

Especificação Comportamental de um subconjunto da Plataforma J2ME

Trabalho de Conclusão de Curso

Engenharia da Computação

Ronaldo Cisneiros Veras.
Orientador: Prof. Dr. Márcio Lopes Cornélio.

Recife, 26 de dezembro de 2005



*ESCOLA POLITÉCNICA
DE PERNAMBUCO*



**Departamento de
Sistemas
Computacionais**

Especificação Comportamental de um subconjunto da Plataforma J2ME

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Ronaldo Cisneiros Veras
Orientador: Prof. Dr. Márcio Lopes Cornélio

Recife, 26 de dezembro de 2005



UNIVERSIDADE
DE PERNAMBUCO

Ronaldo Cisneiros Veras

Especificação Comportamental de um subconjunto da Plataforma J2ME

Resumo

O desenvolvimento de aplicativos para dispositivos com baixa capacidade de recursos, além das limitações impostas pelo hardware, em algumas situações, necessitam de uma implementação que prime pela corretude das aplicações construídas. Além disso, as aplicações precisam ser as mais simples possíveis pelo fato de que as aplicações complexas são geralmente maiores, e com isso, pode ocorrer um baixo desempenho da aplicação durante a sua execução.

Este trabalho aborda a especificação comportamental de um subconjunto da plataforma J2ME, onde foram especificadas a grande maioria das classes do MIDP 2.0 (*Mobile Information Device Profile 2.0*), a qual faz parte da plataforma Java e é bastante utilizada para desenvolvimento de aplicações para telefones celulares, PDA's, dentre outros. A linguagem utilizada para a realização das especificações foi a linguagem JML (*Java Modelling Language*), que é utilizada para realizar especificações comportamentais em aplicações desenvolvidas com a plataforma Java.

Foi utilizado também, nesse trabalho, o verificador estático de aplicações ESC/Java 2, para realizar uma verificação de corretude nas especificações construídas, em detrimento do verificador embutido na distribuição da linguagem JML, que realiza a mesma verificação em tempo de execução.

Abstract

The development of applications for devices with resources of low capacity, in some situations, needs an implementation that take into account the correctness of the constructed applications. Moreover, the applications need to be as simple as possible as complex applications are generally bigger and, consequently, may lead to a low performance.

The objective of this work is to propose a behavioral specification of a subset of the J2ME platform, where had been specified the great majority of the classes of MIDP 2.0 (Mobile Information Device Profile 2.0), which is part of Java platform and is usually used to develop applications for cellular telephones, PDA's, amongst others. The language used for the behavioral specification is the JML language (Java Modelling Language), which is used to specify the behavior of applications developed with the Java language.

It was also used, in this work, the static checker ESC/Java 2 for verifying correctness of the proposed specifications, in detriment of the verifier present in the distribution of the JML language, which is a runtime verifier.

Sumário

Índice de Figuras	vi
Índice de Tabelas	vii
Tabela de Símbolos e Siglas	viii
1 Introdução	10
2 Materiais e Métodos	13
2.1 Plataforma J2ME	13
2.1.1 Arquitetura J2ME	14
2.1.2 CLDC (Connected Limited Device Configuration)	15
2.1.3 CDC (Connected Device Configuration)	16
2.1.4 Máquina Virtual Java	16
2.1.5 MIDP (Mobile Information Device Profile)	17
2.2 Projeto por Contrato	18
2.3 JML (Java Modelling Language)	21
2.3.1 Anotações JML	22
2.3.2 Assertiva <code>requires</code>	22
2.3.3 Assertiva <code>assignable</code>	23
2.3.4 Invariantes em JML	24
2.3.5 A assertiva <code>ensures</code>	24
2.3.6 Assertiva <code>signals</code>	25
2.3.7 Tipos de especificações JML	25
2.3.8 Predicados JML	27
2.4 Ferramentas JML	28
2.4.1 O compilador JML (<code>jmlc</code>)	29
2.4.2 A Ferramenta <code>jmlrac</code>	30
2.4.3 Ferramenta <code>jmldoc</code>	31
2.4.4 As ferramentas de testes unitários	32
2.4.5 ESC/Java 2	33
3 Especificação Comportamental	35
3.1 O subconjunto especificado	35
3.2 O formato das especificações	37
3.3 A especificação formal do MIDP 2.0	38
4 Validação da Proposta	45
4.1 Configurações e uso do ESC/Java 2	45
4.2 Verificação estática das especificações	48
5 Conclusões e Trabalhos Futuros	50
5.1 Contribuições	51

5.2 Limitações	53
5.2.1 Ferramentas JML	53
5.2.2 Máquina Virtual K	54
5.2.3 Tamanho das Aplicações	54
5.3 Dificuldades	55
5.4 Trabalhos Relacionados	55
5.5 Trabalhos Futuros	56
5.5.1 Especificação Formal da plataforma J2ME completa	56
5.5.2 Compilador JML com suporte a KVM	57
5.5.3 Catálogo de componentes J2ME	58

Códigos e Especificações dos Apêndices

Código-Fonte original na Plataforma J2ME	60
Código-Fonte alterado pelo compilador JML	62
Classe TesteSoma.java	67
Classe TesteSoma_JML_Test.java	67
Classe TesteSoma_JML_TestData.java	72
Classes do pacote javax.microedition.lcdui	76
Especificação Alert.jml	76
Especificação AlertType.jml	79
Especificação Choice.jml	80
Especificação ChoiceGroup.jml	83
Especificação Command.jml	87
Especificação CommandListener.jml	90
Especificação DateField.jml	90
Especificação Display.jml	91
Especificação Displayable.jml	95
Especificação Form.jml	97
Especificação Item.jml	100
Especificação List.jml	101
Especificação Screen.jml	105
Especificação TextBox.jml	106
Especificação Ticker.jml	109
Classes do Pacote javax.microedition.rms	110
Especificação DataConverter.jml	110
Especificação InvalidRecordIDException.jml	111
Especificação RecordComparator.jml	112
Especificação RecordEnumeration.jml	112
Especificação RecordEnumerationImpl.jml	114
Especificação RecordFilter.jml	118
Especificação RecordListener.jml	118
Especificação RecordStore.jml	118
Especificação RecordStoreException.jml	125
Especificação RecordStoreFullException.jml	125
Especificação RecordStoreNotFoundException.jml	126
Especificação RecordStoreNotOpenException.jml	126
Classes do Pacote javax.microedition.io	127
Especificação CommConnection.jml	127
Especificação HttpConnection.jml	127
Especificação HttpsConnection.jml	130
Especificação PushRegistry.jml	131
Especificação SecureConnection.jml	132
Especificação SecurityInfo.jml	132
Especificação ServerSocketConnection.jml	133
Especificação SocketConnection.jml	134
Especificação UDPDatagramConnection.jml	135
Classes do Pacote javax.microedition.pki	136
Especificação Certificate.jml	136

Especificação CertificateException.jml	137
Classes do Pacote javax.microedition.midlet	138
Especificação MIDlet.jml	138
Especificação MIDletProxy.jml	140
Especificação MIDletStateChangeException.jml	140
Especificação MIDletStateMapImpl.jml	141
Classes do Pacote java.lang	141
Especificação Class.jml	141
Especificação Runtime.jml	143
Especificação System.jml	144
Especificação IllegalStateException.jml	145
Classes do Pacote java.util	146
Especificação Timer.jml	146
Especificação TimerTask.jml	148
Bibliografia	149

Índice de Figuras

Figura 1. Arquitetura das aplicações J2ME.	15
Figura 2. Código-fonte de uma MIDlet simples.	18
Figura 3. Uso de uma pré-condição em Eiffel	20
Figura 4. Utilização de uma pós-condição em Eiffel.	20
Figura 5. Especificação de Invariantes em Eiffel.	20
Figura 6. Relacionamento linguagem x dubialidade.	21
Figura 7. Uso da assertiva requires.	23
Figura 8. Exemplo do uso de <code>instanceof</code> em uma especificação JML.	23
Figura 9. Utilização da assertiva assignable.	24
Figura 10. Classe Pessoa e a presença de um Invariante.	24
Figura 11. Uso da assertiva ensures.	24
Figura 12. Uso da assertiva signals.	25
Figura 13. O uso de <code>normal_behaviour</code> .	26
Figura 14. Uso da palavra-chave <code>behaviour</code> .	26
Figura 15. Uso da palavra-chave <code>exceptional_behaviour</code> .	27
Figura 16. Utilização do predicado <code>\forall</code> . (Fonte: Manual de Referência de JML [36])	28
Figura 17. Quantificadores aplicados a números. (Fonte: Manual de Referência de JML [36])	28
Figura 18. Utilização do compilador <code>jmlc</code> .	29
Figura 19. Uso da ferramenta <code>jmlrac</code> .	31
Figura 20. Documentação gerada pelo <code>jmldoc</code> .	32
Figura 21. Princípio de funcionamento do ESC/Java 2.	34
Figura 22. Os métodos <code>getProtocol</code> e <code>getHost</code> .	41
Figura 23. Os métodos <code>getFile</code> , <code>getRef</code> e <code>getQuery</code> .	41
Figura 24. Os métodos <code>setRequestMethod</code> e <code>getRequestMethod</code> .	42
Figura 25. Os métodos <code>getRequestProperty</code> e <code>setRequestProperty</code> .	43
Figura 26. Execução da interface gráfica do ESC/Java 2.	46
Figura 27. Ambiente configurado para verificação com MIDP 2.0.	46
Figura 28. Configuração da aba ESC Options para o nosso trabalho.	47
Figura 29. Resultado da verificação estática em <code>HttpConnection</code> .	48
Figura 30. Resultado da verificação estática em <code>AlertType</code> .	49
Figura 31. Falha na compilação de aplicativo J2ME com <code>jmlc</code> .	57

Índice de Tabelas

Tabela 1. Subconjunto da plataforma J2ME escolhido para especificação.

36

Tabela de Símbolos e Siglas

API – Application Programming Interface.
J2ME – Java 2 Mobile Edition.
PDA – Personal Digital Assistant(Assistente Pessoal Digital).
CLDC – Connected Limited Device Configuration.
CDC – Connected Device Configuration.
JML – Java Modelling Language.
HTML – Hypertext Markup Language.
ESC/Java 2 – Extended Static Checker for Java Version 2.
MIDP 2.0 – Mobile Information Device Profile Version 2.
JSR – Java Specification Request.
WAP – Wireless Application Protocol.
MID – Mobile Information Device.
OEM – Original Equipment Manufacturer.
KVM – K Virtual Machine(Máquina Virtual K).
GUI – Graphical User Interface(Interface Gráfica com o Usuário).
RUP – Rational Unified Process.
UML – Unified Modelling Language.
XML – Extensible Markup Language.
URL – Uniform Ressearch Location
GSM – Global System for Mobile Communications.
e-CPF – Cadastro de Pessoa Física Eletrônico.
e-CNPJ – Cadastro Nacional de Pessoa Jurídica Eletrônico.
JAAS – Java Authentication and Authorization.
J2EE – Java 2 Enterprise Edition.

Agradecimentos

Gostaria de agradecer, primeiramente, a Deus pela minha vida, por todos os bons momentos que Ele me deu o prazer de desfrutar e pela paciência e força que me foram dadas nos inúmeros momentos difíceis.

À minha mãe, Maria Dalva, por todo o seu esforço e carinho com que sempre cuidou de mim e soube me entender nos momentos difíceis. À meu pai, Pedro Paiva Veras (*in memoriam*), a quem dedico a conclusão de mais essa etapa da minha vida. À minha irmã Daniela Cisneiros, que torceu por mim ao seu modo, e ao meu cunhado, Danilo Hereda, por sempre estar me incentivando. Dedico também este trabalho à toda minha Família. Obrigado a todos!

Gostaria de agradecer, em especial, ao meu orientador, Márcio Lopes Cornélio, por ter me ajudado bastante durante todo o período em que desenvolvemos este trabalho, não somente com os ensinamentos técnicos, mas também com o incentivo e a compreensão que me fizeram seguir adiante.

Aos professores do meu curso por todos os ensinamentos que pude adquirir durante esses cinco anos de convivência e pela filosofia de trabalho estabelecida por eles de sempre estar perto de seus alunos.

Aos amigos de Universidade que são muitos e por isso, não citarei nomes para evitar o esquecimento de alguns nomes. Mas todos eles são pessoas muito especiais.

Às pessoas que me ajudaram na vida profissional, em especial, aos meus amigos Adilson Arcoverde Júnior e Gabriel Alves de Albuquerque, que muito me ensinaram e me deram a oportunidade de progredir profissionalmente. À todo povo de Tecnologia da Informação da São Mateus Frigorífico: Rivaldo, Chico, Vanessa, Giseli, Rogério, Pedro, Humberto, Fábio, Jorge e Berg. Ao amigos de trabalho da Segsat Rastreamento: Erick, Thyago, Éricles, Frédick, Meira, Newton, Roberto, Luís Carlos.

Obrigado a todos vocês.

Capítulo 1

Introdução

Uma das grandes preocupações existentes na Engenharia de Software diz respeito ao desenvolvimento de aplicações que sejam, ao mesmo tempo, seguras e corretas. Para que as aplicações consigam assegurar esse grau desejado, uma técnica bastante utilizada é uma política intensa de testes de softwares e esses, geralmente, são realizados numa fase posterior ao desenvolvimento.

As aplicações desenvolvidas com J2ME [1][2][3], plataforma Java para dispositivos móveis, são consideradas corretas desde que as suas implementações correspondam, exatamente, às especificações das funcionalidades presentes nessas aplicações. Algumas aplicações J2ME podem exigir um grau mais elevado de corretude como, por exemplo, uma aplicação bancária executada em um telefone móvel. Nestes tipos de aplicações precisamos utilizar técnicas que nos assegurem que o desenvolvimento da aplicação alcançou o grau de corretude solicitado.

Existem várias técnicas que podem ser utilizadas para verificar a corretude de aplicações de software. Os testes de software [4] é uma das conhecidas e muito presente na maioria dos times de desenvolvedores. Existem vários tipos de teste de software. Os testes de unidade, ou testes unitários, são aqueles aplicados a pequenas partes de uma aplicativo. A vantagem dos testes de unidade é podermos projetar apenas uma parte da implementação em um caso de teste e, assim, realizarmos o teste propriamente dito.

Os testes de funcionalidade, ou testes funcionais, são testes que podem ser aplicados em uma porção maior de uma aplicação para verificar se uma determinada funcionalidade desempenha o seu papel corretamente, ou seja, se a tarefa a ser executada pela funcionalidade em teste é realizada da forma como ela foi requisitada.

Outra estratégia ainda pouco utilizada para testar a corretude de programas é a verificação destes através de ferramentas especializadas como, por exemplo, um verificador de programas. A verificação de programas, geralmente, constrói uma representação lógica a partir da especificação de um programa, utilizando predicados lógicos, para representar sentenças da aplicação.

As aplicações para dispositivos móveis como, telefones celulares, PDA's, pagers, por serem aplicações que devem ser projetadas para um hardware específico, possuem algumas limitações com relação a quantidade de recursos disponíveis. Em aplicações móveis utilizando a linguagem J2ME, temos limitações de recursos como, memória, tamanho restrito da área de visualização das aplicações, dentre outras. Essas restrições são impostas pela configuração CLDC

(*Connected Limited Device Configuration*) [1][2][5] ou pelo CDC (*Connected Device Configuration*) [2][6].

Tendo em vista essas limitações de recursos, é preciso que as aplicações projetadas para a plataforma J2ME, ou para quaisquer outras aplicações projetadas para dispositivos com capacidade de recursos limitadas, possuam um elevado grau de corretude e segurança. Além disso, essas aplicações precisam realizar as suas funcionalidades da forma mais simples possível em prol do seu bom desempenho.

A linguagem JML (*Java Modelling Language*) [7][8][9][10] foi desenvolvida para ser utilizada na construção de especificações comportamentais para aplicações desenvolvidas em Java. Através da linguagem JML podemos desenvolver especificações que representam o comportamento de programas, ou partes de programas, desenvolvidos com Java e essas especificações, podem ser posteriormente verificadas pelas ferramentas que oferecem suporte para JML.

Existe uma ampla quantidade de ferramentas que oferecem suporte para a linguagem JML [11]. A própria distribuição da linguagem contém uma variedade de ferramentas para propósitos específicos como, por exemplo, um compilador específico da linguagem, uma ferramenta para verificação de programas em tempo de execução, uma ferramenta para geração de documentação Java no formato HTML para as especificações construídas, uma ferramenta para geração de classes de testes unitários e, por fim, uma outra capaz de executar os testes de unidade para as classes de teste geradas.

Uma outra e importante ferramenta que oferece suporte para a linguagem JML é o ESC/Java 2 (*Extended Static Checker for Java Version 2*) [12]. O ESC/Java 2 é um verificador estático de programas escritos em JML e é uma evolução natural do Projeto ESC/Java mantido pela Compaq Research, onde uma das grandes inovações do projeto é o suporte direto à linguagem JML e o suporte as versões mais novas do J2SE (*Java 2 Standard Edition*) [13]. O ESC/Java 2 realiza uma verificação estática nas especificações JML, ou seja, em tempo de compilação, ao contrário do verificador embutido na distribuição de JML, o `jmlrac`, que só é capaz de fazer a verificação das especificações em tempo de execução.

Uma das grandes vantagens da linguagem JML é que ela é baseada em um conceito bastante importante da Engenharia de Software, o Projeto por Contrato [14][15]. O Projeto por Contrato é uma técnica de desenvolvimento de software na qual é estabelecida uma interface de utilização para um componente de software e as regras de utilização para as funcionalidades desse componente, ou seja, o usuário do componente não precisa conhecer a implementação das funcionalidades, basta conhecer quais funcionalidades podem ser utilizadas e as regras que devem ser obedecidas para a utilização desses componentes. Uma das primeiras linguagens a utilizar conceitos do Projeto por Contrato foi a linguagem Eiffel [16].

No Projeto por Contrato se ao utilizar o componente de software o usuário satisfizer as condições estabelecidas pelo componente, o resultado de tal uso está assegurado, de acordo com o contrato de utilização de software.

O desenvolvimento de especificações de software utilizando uma abordagem baseada no Projeto por Contrato é um dos caminhos mais naturais para um desenvolvimento voltado a componentes de software. Podemos tomar, como exemplo, uma classe Java. Se desenvolvermos especificações comportamentais para os métodos dessa classe estabelecendo as pré-condições para a sua utilização, as pós-condições que devem ser garantidas pelo método após a sua execução, podemos, então, construir uma interface comportamental de software e disponibilizá-la para o usuário da classe. Dessa forma, o usuário não precisará mais conhecer a implementação dos métodos daquela classe, ou seja, ele a utilizará como um componente que realizará determinada funcionalidade desde que o usuário obedeça as condições de utilização.

O objetivo desse trabalho é propor a utilização de Métodos Formais [17], representado pela linguagem JML, no desenvolvimento de aplicações J2ME. Para isso, foi construída uma especificação comportamental para um subconjunto da plataforma J2ME. Foi escolhido um conjunto de classes e interfaces da plataforma J2ME, mais especificamente, a grande maioria das classes do MIDP (*Mobile Information Device Profile*) 2.0 [18], e foi desenvolvida uma especificação comportamental utilizando a linguagem JML para cada uma das classes escolhidas. Essa especificação construída é baseada na especificação informal, disponibilizada pela Sun Microsystems, para o MIDP 2.0 [18], a JSR-118 [19].

Para fazer uma validação nas especificações construídas, utilizamos também o verificador estático ESC/Java 2. A verificação estática teve o objetivo de assegurar a consistência entre a especificação e a implementação para a qual a especificação foi construída.

Esta monografia possui a seguinte estrutura: no Capítulo 2 descrevemos as tecnologias envolvidas para o desenvolvimento das especificações comportamentais. Iniciamos o capítulo descrevendo os conceitos relativos à plataforma J2ME como, as configurações presentes na plataforma, o perfil para os dispositivos, a máquina virtual Java utilizada, a arquitetura das aplicações, dentre outras. Em seguida descreveremos os conceitos relativos à linguagem JML, bem como as ferramentas que oferecem suporte à mesma.

No Capítulo 3 foram mostradas como foram realizadas as especificações JML para este trabalho, como foram selecionadas as classes para serem especificadas e o formato adotado para as especificações.

No Capítulo 4 foram mostradas, em detalhes, como foram realizadas as validações nas especificações construídas utilizando a ferramenta para verificação estática de programas ESC/Java 2. Esse capítulo também mostra os resultados das validações das especificações para uma classe e uma interface.

Finalmente, no Capítulo 5, foi apresentada a conclusão para nosso trabalho, bem como, as limitações, as dificuldades encontradas durante o seu desenvolvimento, trabalhos relacionados, e direções para trabalhos futuros.

Capítulo 2

Materiais e Métodos

Neste capítulo, serão abordados conceitos sobre a plataforma J2ME, a qual tem sido largamente utilizada para desenvolvimento de aplicações para celulares. Descrevemos a plataforma J2ME, o MIDP (*Mobile Information Device Profile*), a configuração CLDC, a máquina virtual K, dentre outros aspectos.

O Projeto por Contrato (*Design by Contract*) também será abordado neste capítulo, com ênfase na sua aplicação na linguagem Eiffel [16], que foi uma das pioneiras no seu uso. Será mostrada a importância do Projeto por Contrato para regulamentar o uso de componentes de software com correteude, assim como a sua aplicação na linguagem JML.

A linguagem JML é descrita logo em seguida. São apresentadas algumas das principais anotações, que são construções sintáticas da linguagem JML bastante semelhantes aos comentários da linguagem Java, utilizadas em JML para implementar uma especificação. Serão abordadas apenas algumas das anotações disponíveis nesta linguagem.

As várias ferramentas que dão suporte ao uso de JML são apresentadas em seguida. Será descrito o objetivo de cada ferramenta e, para algumas delas, mostramos o seu princípio de funcionamento, e seu uso.

2.1 Plataforma J2ME

Dispositivos móveis como celulares, PDA's, *paggers*, dentre outros, têm sido comumente utilizados para realizar tarefas, que no passado, seriam de difícil execução principalmente por decorrência do pouco avanço tecnológico disponível naquele momento para esses dispositivos. Nos dias atuais é bastante comum a comunicação através de mensagens eletrônicas, utilizando o protocolo WAP [20] ou outras tecnologias, via telefones celulares.

Outra aplicação de destaque quanto ao avanço tecnológico dos pequenos dispositivos é a utilização de computadores de mão em diversos ramos de serviços. Nos restaurantes mais modernos, garçons registram e realizam os pedidos dos clientes munidos de um computador de mão, outro exemplo é a realização de vendas por funcionários que trabalham fora da Corporação e precisam se comunicar com ela de qualquer lugar que estejam, essas e muitas outras aplicações utilizam a tecnologia de pequenos dispositivos.

Outra grande aplicação que está presente principalmente em telefones celulares é a ampla gama de jogos distribuídos por fabricantes e operadoras para que os clientes possam utilizá-los como forma de entretenimento.

Alguns fatos justificam o grande avanço no ramo de desenvolvimento para pequenos dispositivos como, por exemplo, propor aos usuários serviços que, até bem pouco tempo atrás, nem se imaginava que poderiam ser realizados através desses dispositivos como consultar informações pessoais e financeiras através do telefone celular, prover conteúdo pra entretenimento de qualquer lugar onde se queira estar como jogos, acesso a Internet, dentre outros.

Um fato bastante relevante a considerar é que a falta de recursos inerentes à baixa capacidade provida pela parte física desses dispositivos como, por exemplo, uma memória reduzida, fazem com que a qualidade dos serviços seja, também, mais baixa que o mesmo tipo de serviço provido por uma máquina com recursos mais abrangentes.

Dentro desse contexto, tais aplicações podem ser implementadas por uma ampla gama de tecnologias e a escolha sobre qual delas utilizar, dependerá, muitas vezes, da finalidade proposta pela aplicação e também de requisitos arquiteturais e temporais que deverão ser respeitados. A plataforma J2ME (*Java 2 Micro Edition*) é uma das possíveis tecnologias que podem ser utilizadas para desenvolvimento de aplicações para vários pequenos dispositivos como telefones celulares, pagers, etc.

O escopo definido para esse trabalho é abordar apenas um subconjunto dessa plataforma de forma que a parte abordada, seja aquele presente na maioria das aplicações para telefones celulares, ficando as outras partes para trabalhos futuros. Para realizar o desenvolvimento de aplicações para telefones celulares utilizam-se dois componentes contidos na plataforma J2ME: o CLDC (*Connected Limited Device Configuration*) e o MIDP (*Mobile Information Device Profile*).

2.1.1 Arquitetura J2ME

Uma aplicação J2ME deve ser projetada para uma arquitetura definida pela própria plataforma J2ME. Na camada mais básica da arquitetura encontramos o componente chamado MID (Dispositivo de Informação Móvel). Esse componente representa o hardware propriamente dito, ou seja, o dispositivo onde será implantada a aplicação. Esse dispositivo pode variar de acordo com a finalidade da aplicação que está sendo desenvolvida, por exemplo, poderá ser um telefone celular onde será implantado um software para consulta de saldos bancários, etc.

Em uma camada mais acima encontramos o sistema operacional nativo. Esse sistema operacional vem embutido no dispositivo desde a sua fabricação. Dentre os mais conhecidos sistemas operacionais para pequenos dispositivos estão o Microsoft Windows CE [21] e o Symbian OS [22].

Acima da camada que representa o sistema operacional nativo temos o CLDC (Configuração de Dispositivo Conectado Limitado). O CLDC, inserido no contexto de uma aplicação J2ME, representa uma configuração que classifica determinados dispositivos como pertencentes a uma classe. Como exemplo de configuração, temos no CLDC dispositivos que recebem cargas elétricas através de baterias. Já os dispositivos que pertencem a configuração CDC possuem, como uma de suas características, conexão de rede.

O MIDP (Perfil de Dispositivo de Informação Móvel) é o componente que, em termos de software, representa as limitações impostas por uma configuração. Em outras palavras, o MIDP é uma API contendo classes Java que satisfazem os critérios de implementação descritos pela especificação do CLDC.

Como complemento ao perfil MIDP, ainda podemos considerar as classes específicas do OEM (Fabricante de Equipamento Original). Essas classes não fazem parte, originalmente, da plataforma J2ME, pois são classes que não podem ser usadas de forma genérica para qualquer aplicação e sim, para aplicações de uma determinada família de dispositivos de um fabricante específico. Considerando apenas os telefones celulares, os inúmeros fabricantes de aparelhos que suportam J2ME em seus dispositivos, disponibilizam componentes de software em seus portais para que a comunidade de desenvolvedores possa utilizar tais componentes com o intuito de desenvolver softwares para os aparelhos onde aqueles componentes podem ser incorporados.

Na última camada dos componentes que representam a arquitetura de um aplicativo J2ME temos três subcomponentes: os aplicativos MIDP, os aplicativos específicos do OEM e os aplicativos nativos. Os aplicativos MIDP são aqueles que utilizam apenas classes específicas da plataforma J2ME como, por exemplo, as classes de interface gráfica do MIDP. Os aplicativos específicos do OEM são aqueles construídos com as API's auxiliares disponibilizadas pelos fabricantes de dispositivos. Os aplicativos nativos são aqueles que já pertencem ao dispositivo e que, em alguns casos, podem ser integrados aos sistemas operacionais nativos. A Figura 1 ilustra a arquitetura das aplicações J2ME .

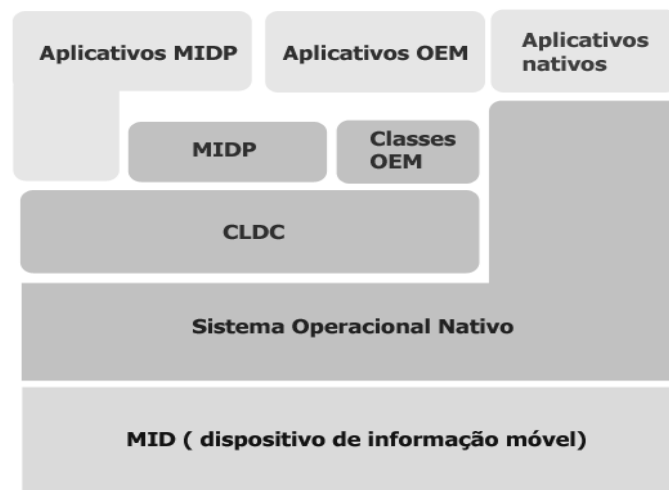


Figura 1. Arquitetura das aplicações J2ME.

2.1.2 CLDC (Connected Limited Device Configuration)

O CLDC [1][2][5] tem o papel de, ao mesmo tempo, definir uma especificação para uma máquina virtual Java, chamada de KVM, para a plataforma J2ME e também de definir um conjunto de classes Java que proporcione o suporte necessário para desenvolver aplicações voltadas para a plataforma J2ME. Ressaltamos que o CLDC não é usado especificamente para aplicações para telefones celulares, mas para dispositivos que possuem recursos limitados de memória e capacidade de vídeo, por exemplo.

O CLDC especifica algumas restrições de configurações que classificam alguns dispositivos como pertencentes a uma mesma classe de dispositivos. Algumas delas são listadas abaixo:

- 128 kbytes de memória para executar o Java.
- 32 kbytes para alocação de memória em tempo de execução.
- Interface restrita com o usuário.
- Normalmente alimentado por bateria.
- Conectividade de rede, normalmente dispositivos sem fio com largura de banda baixa e acesso intermitente.

2.1.3 CDC (Connected Device Configuration)

O CDC [2][6], assim como o CLDC, têm o papel de estabelecer uma configuração específica para determinados tipos de dispositivos, além de prover um conjunto de classes Java que tornem viável a implementação de aplicações para a plataforma J2ME.

Considerando o papel que o CDC desempenha no contexto da plataforma J2ME, pode-se perceber que é bem semelhante ao papel desempenhado pelo CLDC, porém o CDC especifica algumas configurações, que listamos abaixo, que o diferenciam do CLDC:

- 512 kbytes de memória para execução do Java
- 256 kbytes de memória para alocação de memória em tempo de execução
- Conectividade de rede, largura de banda geralmente persistente e alta.

Percebe-se claramente, se compararmos as duas especificações dos recursos inerentes a cada uma das configurações, que o CDC possui uma quantidade maior de recursos e por isso deve agrupar dispositivos muitas vezes maiores e geralmente mais eficientes que os dispositivos que possuem uma configuração compatível com o CLDC.

2.1.4 Máquina Virtual Java

Um conceito bastante comum para os desenvolvedores que utilizam a plataforma Java é o conceito de máquina virtual Java. Uma das diretrizes principais, quando do processo de concepção da linguagem Java, era a idéia de desenvolver uma linguagem onde uma aplicação desenvolvida dentro de uma plataforma, fossem portáveis para qualquer outra plataforma, ou seja, uma aplicação desenvolvida utilizando a plataforma Windows, por exemplo, executaria normalmente utilizando a plataforma Linux.

Considerando a plataforma J2ME e suas configurações, temos alguns pontos relevantes a considerar com relação à máquina virtual. Primeiramente, se considerarmos a configuração CDC, a máquina virtual utilizada é a mesma utilizada por aplicações utilizando J2SDK (a versão Java para aplicações desktop).

Para dispositivos com uma configuração CLDC, por questões de limitações de recursos, a máquina virtual Java foi modificada. A KVM [26] (máquina virtual K), como ela é chamada, possui algumas diferenças relativas à máquina virtual tradicional como, por exemplo, não suporta operações com número em ponto flutuante, a API de reflexão sofre algumas alterações que limitam o seu uso dentro de alguns contextos como, por exemplo, para obtenção dos arquivos de classes carregados.

A máquina virtual K foi desenvolvida para alcançar dois objetivos: ser menor que a máquina virtual do J2SE e ser mais eficiente. Em termos de memória, a KVM executa em apenas 60 quilobytes de memória, por isso a origem da letra K no nome KVM representando o k de quilobytes. A KVM foi escrita em C para tornar-se portátil para uma ampla gama de plataformas.

2.1.5 MIDP (Mobile Information Device Profile)

O MIDP [1][2][18] tem o papel de prover uma API com a implementação de classes que devem satisfazer os requisitos de limitações de recursos dos pequenos dispositivos, ou seja, o MIDP especifica, através de software, as restrições de recursos definidas pelo CLDC.

As implementações presentes no perfil juntamente com aquelas presentes no CLDC, provêm um ambiente para construção de aplicações para, por exemplo, telefones celulares. As classes providas pelo MIDP servem para regular o uso de interfaces gráficas com o usuário de uma forma eficiente, disponibilizam um *framework* genérico capaz de realizar conexões de software, realizar armazenamento persistente de dados, dentre outros.

Os pacotes de classes mais importantes do MIDP agrupam classes que servem para implementação de um determinado objetivo específico. Para exemplificar, serão descritos os nomes de alguns deles e a sua utilização dentro de um projeto de software para pequenos dispositivos:

- **java.util.***: define classes que são utilizadas para realização de tarefas de temporização, classes que representam estruturas de dados, classes de datas, etc.;
- **java.io.***: onde se encontram classes para realização de operações de entrada e saída;
- **javax.microedition.lcdui.***: onde estão presentes as classes que podem ser utilizadas para a construção de interfaces gráficas com o usuário. Neste pacote podemos encontrar as classes que representam os componentes de GUI (*Graphical User Interface*) pré-definidos como `TextBox`, `TextField`, dentre outros, e as classes que propiciam ao usuário construir seus próprios componentes de GUI como as classes `Canvas`, `Graphics`, etc. Dentro deste pacote encontramos um outro pacote contendo classes específicas para a implementação de jogos;
- **javax.microedition.rms.***: neste pacote estão contidas classes que proporcionam uma forma de armazenamento persistente para estes dispositivos com baixa capacidade de recursos;
- **javax.microedition.midlet.***: contém classes que juntas são capazes de representar todo o ciclo de vida das MIDlets, a qual é a forma mais comum de entender uma aplicação J2ME e será abordada posteriormente.

Uma aplicação J2ME é comumente composta de uma ou várias MIDlets. Uma MIDlet é uma unidade de software que pode ser disponibilizada para utilização por um dispositivo e que pode realizar um objetivo específico. Uma calculadora, por exemplo, presente na maioria dos telefones celulares pode ser composta de várias MIDlets, onde a tela principal, na qual o usuário solicita o tipo da operação e informa os operandos, é um exemplo de MIDlet.

O conceito de MIDlet obedece a um ciclo de vida estabelecido. Uma MIDlet pode alcançar três estados durante o seu ciclo de vida:

- **Pausa**: a MIDlet encontra-se nesse estado após uma chamada ao seu construtor e antes que o gerenciador de aplicativos seja iniciado.
- **Ativa**: quando a MIDlet é posta em execução.
- **Destruída**: quando a MIDlet libera os recursos adquiridos durante a sua execução e é desligada pelo gerenciador de aplicativos.

A Figura 2 mostra um exemplo de MIDlet simples, onde são visualizados os métodos que expressam o ciclo de vida de uma aplicação J2ME baseada em MIDlet. O método `startApp` define como o gerenciador de aplicativos inicia a execução de uma MIDlet. Esse método poderá ser chamado várias vezes. O método `pauseApp` é utilizado pelo gerenciador de aplicativos, após o construtor da MIDlet ter sido executado, com a intenção de levar a MIDlet para um estado de pausa. O método `destroyApp` é utilizado pelo gerenciador de aplicativos para terminar a execução de uma MIDlet. No corpo desse método podem ser liberados os recursos que, porventura, tenham sido alocados dentro do método `startApp`.

```
public class MidletSimples extends MIDlet {
    //O construtor da MIDlet
    public MidletSimples( ){

    }
    //Chamado pelo gerenciador de aplicativos para iniciar uma
//MIDlet
    public void startApp( ){

    }
    //Chamado pelo gerenciador de aplicativos antes da pausa da
//MIDlet
    public void pauseApp( ){

    }
    //Chamado pelo gerenciador de aplicativos antes do
//desligamento
    public void destroyApp( boolean unconditional ){

    }
}
```

Figura 2. Código-fonte de uma MIDlet simples.

2.2 Projeto por Contrato

As metodologias de desenvolvimento de software, como o RUP (*Rational Unified Process*) [27], muitas vezes presam por um aparato enorme de documentação, como, por exemplo, diagramas UML (*Unified Modelling Language*) [28][29], documentos de requisitos, de projeto, etc. No entanto, essa documentação muitas vezes não é precisa e completa o suficiente para que desenvolvedores inseridos no projeto possam entendê-la corretamente. Mesmo em uma especificação formal, um especificador pode esquecer alguma funcionalidade.

Um dos objetivos dos documentos em um projeto de software é fornecer informações suficientes para que um novo membro de uma equipe possa, rapidamente, se inteirar sobre as decisões de projeto tomadas por outros membros da equipe antes de sua integração. Outro objetivo é fazer com que os próprios desenvolvedores possam lembrar decisões antigas que foram tomadas há um tempo atrás, durante o processo de desenvolvimento.

Os grandes problemas relacionados às documentações de projeto são a incompletude, a falta de corretude e o tipo de linguagem utilizada para documentar as decisões de projeto. Incompletude diz respeito a funcionalidades não serem documentadas totalmente, podendo representar um grande problema para o leitor que tenta compreender o documento que pode ser,

por exemplo, uma especificação de software. A corretude, em termos de uma especificação de software, diz respeito a verificar se aquilo que foi requisitado para a construção do software corresponde, exatamente, ao que está contido na especificação.

A linguagem utilizada para construção dos documentos deve possuir, dentre outros requisitos, uma forma não-ambígua de especificação, ou seja, a informação documentada deverá ser entendida da mesma forma por todos os leitores que examinarem o documento. A grande dificuldade para uma documentação não-ambígua encontra-se quando a linguagem escolhida para a documentação é o mais semelhante possível da linguagem natural. Linguagens naturais, ao mesmo tempo em que podem aumentar o número dos usuários que estão aptos a lerem o documento, podem levar os mesmos a terem interpretações diferentes para o mesmo conceito documentado.

Para tentar minimizar o problema, surgiram várias linguagens para especificação de documentos. A linguagem UML (*Unified Modelling Language*) é uma linguagem muito conhecida para realizar a documentação de modelos de software, para construção de diagramas de componentes, diagramas de interação, entre outros. As linguagens formais também são utilizadas como ferramentas para especificação de software, documentando decisões de projetos tomadas durante o processo de desenvolvimento. Dentre as linguagens formais, podemos citar a linguagem Z [30][31], cuja base é a teoria dos conjuntos e Lógica de Primeira Ordem, a linguagem Circus [32], a qual é uma linguagem baseada na linguagem Z e no cálculo de refinamentos [33], a linguagem OhCircus, que representa uma extensão da linguagem Circus implementada sobre o paradigma de Orientação a Objetos, etc. A linguagem JML (*Java Modelling Language*) também é utilizada principalmente pela comunidade de desenvolvedores Java para a especificação formal do comportamento e interface de classes e métodos escritos na linguagem Java.

O Projeto por Contrato é uma técnica utilizada para construção de aplicações baseadas em componentes de software. Um contrato de software é composto por uma interface, contendo as operações que podem ser realizadas pela implementação do componente, e as condições de utilização para cada uma dessas operações. A interface do componente torna visível, para quem quer utilizá-lo, o leque de operações que o componente pode realizar. As condições de uso para os componentes, por sua vez, especificam o que deve ser satisfeito pelo usuário do componente para que este possa realizar uma de suas operações corretamente.

Para termos o componente de software funcionando corretamente é preciso obedecer às condições impostas pela especificação da interface e, se as condições forem satisfeitas, o componente garante que os resultados das operações realizadas ocorram corretamente. Fazendo um paralelo entre um componente de software e um componente de hardware, teríamos como componente o hardware em si, o qual não precisa explicitar ao usuário como suas tarefas são realizadas, e como as condições de uso do componente, as restrições como, por exemplo, a voltagem que poderá ser utilizada para o correto funcionamento do hardware.

Uma das primeiras linguagens a adotar o projeto por contrato foi a linguagem Eiffel. Para especificar as regras de utilização, Eiffel tem algumas asserções que regulam as condições de uso. Uma pré-condição estabelece tudo que deve ser satisfeito para o correto funcionamento daquilo que está sendo especificado, ou seja, se quisermos utilizar uma função que tenha como retorno a raiz quadrada de um número positivo, a pré-condição de uso para essa função seria que a função em questão deve receber, como argumento, um número positivo. A Figura 3 ilustra o uso de uma pré-condição utilizada na linguagem Eiffel. Essa pré-condição especifica que o argumento que deverá ser enviado para a função de extração da raiz quadrada deverá ser positivo.


```
require
    argumento_funcao_sqrt: arg >= 0
```

Figura 3. Uso de uma pré-condição em Eiffel

Um contrato especifica uma relação onde ambas as partes contribuem para um objetivo comum. Se o usuário satisfizer uma pré-condição de utilização, algo em troca deverá ser garantido. Essa “garantia” é chamada, no projeto por contrato, de pós-condição. A pós-condição estabelece o que será retornado, se a pré-condição for satisfeita. Utilizando o mesmo exemplo da função que retorna a raiz quadrada de um número positivo, a pós-condição seria o fato de que o retorno dessa função também seria um número positivo. A Figura 4 ilustra a utilização de uma pós-condição especificada na linguagem Eiffel. Nessa especificação vemos que o resultado da função que retorna a raiz quadrada de um número positivo, deve ser, também, um número positivo.

```
ensure
    retorno_funcao_sqrt: retorno > 0
```

Figura 4. Utilização de uma pós-condição em Eiffel.

Invariantes também são relevantes quando se trata de especificações formais. Os invariantes são predicados que descrevem fatos que devem ser sempre verdadeiros durante todo o escopo de uma especificação. A Figura 5 mostra a utilização de invariantes em Eiffel. Essa especificação poderia ser utilizada para validar requisitos de tempo em uma função que o utilizasse como argumento.

```
invariant
    hora_valida: 0 <= hora and hora <= 23
    minuto_valido: 0 <= minuto and minuto <= 59
    segundo_valido: 0 <= segundo and segundo <= 59
```

Figura 5. Especificação de Invariantes em Eiffel.

A linguagem Eiffel possui ainda outras asserções que podem ser utilizadas para especificar uma ampla gama de situações. As asserções acima servem apenas para exemplificar as situações mais casuais em um projeto por contrato. Situações excepcionais também são especificadas em um projeto por contrato e um exemplo desse tipo de situação será ilustrado mais a frente, quando apresentarmos a linguagem JML.

Uma especificação de software é importante pelo fato de prover aos desenvolvedores duas contribuições: a primeira delas é a documentação de decisões de implementação realizadas durante um processo de desenvolvimento, o que poderá ser bastante útil no futuro, quando for preciso realizar manutenção da mesma, e a segunda contribuição é o fato de através da implementação do raciocínio sobre como realizar uma operação na forma de especificação, podermos utilizar, a depender da linguagem de especificação utilizada, a especificação formal como entrada para um verificador estático, por exemplo, e esse poderá comprovar a sua correteza. Dentre os verificadores mais conhecidos temos o ESC/Java [34] que foi construído pela Compaq Research. Esse verificador recebe como entrada um arquivo contendo uma classe Java na qual estão presentes alguns métodos especificados com uma linguagem bastante semelhante a JML. A especificação presente em cada classe utilizada pelo ESC/Java, antes das

implementações dos métodos ou como invariantes, regem as condições de utilização de tais métodos.

2.3 JML (Java Modelling Language)

Na Seção 2.2 apresentamos conceitos de grande relevância para o desenvolvimento de software com ênfase na qualidade do mesmo. Os componentes de software reutilizáveis proporcionam redução no tempo de desenvolvimento de um projeto. O projeto por contrato estabelece as regras para utilização desses componentes e a sua garantia de correteude.

Em termos de uma especificação de software temos, como mencionado anteriormente, o agravante de que quanto mais próxima a especificação estiver do nível de linguagem natural, mais dúbida poderá ser. A Figura 6 mostra o relacionamento entre os níveis de linguagem de especificação de software e o seu grau de informações dúbias que podem ser causadas pela má interpretação do que está especificado. Na figura vemos claramente que a linguagem matemática, por ser uma linguagem precisa, causa menos dubialidade ao usuário da especificação, porém, é uma linguagem que pode ser entendida por uma quantidade menor de pessoas, apesar de ser universal, pois uma especificação puramente matemática pode não ser facilmente compreensível por todos os usuários. Por outro lado, as linguagens naturais são bastante compreensíveis, porém, podem ser pouco precisas com relação ao que se deseja expressar.

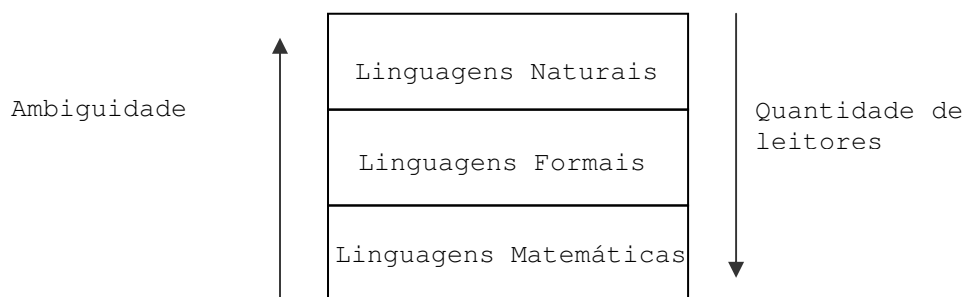


Figura 6. Relacionamento linguagem x dubialidade.

A construção de uma especificação de software necessita de uma linguagem que seja capaz de expressar o correto comportamento da parte do software que está sendo especificado e, se possível, possuir uma sintaxe semelhante àquela utilizada para a construção do software. A linguagem para especificação JML [7][8][9][10] segue exatamente esses requisitos. JML é uma linguagem utilizada para a especificação de contratos a respeito de interfaces e comportamento de aplicações Java, e possui uma sintaxe semelhante a Java, com uma grande quantidade de ferramentas capazes de realizar verificação estática, compilação, geração automática de classes de teste para o JUnit [35], geração de javadocs, entre outras.

Em uma especificação formal que utiliza a linguagem JML podemos especificar atributos, invariantes de classe, comportamento de métodos, e tipos abstratos de dados. Descreveremos cada uma dessas especificações mais adiante.

2.3.1 Anotações JML

As anotações presentes na linguagem JML permitem construir aplicações que sejam, ao mesmo tempo, próximas da linguagem utilizada para a especificação do software (que para a JML é a linguagem Java) e que garantam a qualidade das aplicações construídas. Quando se utiliza o termo “próximo da linguagem” estamos nos referindo a utilização de estruturas dessa linguagem dentro da linguagem de especificação para realizar a especificação do aplicativo. Por exemplo, quando é preciso estabelecer a pós-condição de um método que tem como retorno um objeto que é uma instância da classe `java.lang.String` podemos utilizar a sintaxe: `//@ ensures \result instanceof java.lang.String`. Como podemos observar, estamos utilizando uma forma de especificação que possui uma sintaxe mista entre a linguagem Java, que fica caracterizada pela presença do operador `instanceof`, e a presença de assertivas e operadores inerentes à linguagem JML, como a assertiva `ensures` e o operador `\result`.

Com relação à integração de anotações da linguagem JML em códigos-fonte implementados em Java, novas versões do J2SE, iniciando a partir do Java 5 (também conhecido como *Tiger*), estão implementando e recomendando, fortemente, a utilização de *Annotations* em substituição aos arquivos contendo metadados (informações auxiliares ao bom funcionamento de um *framework*, por exemplo), que geralmente eram expressos em arquivos no formato XML. A presença de uma linguagem de especificação de software baseada em anotações pode representar uma grande colaboração para os desenvolvedores que já estão familiarizados com essas anotações. Com a integração das *Annotations* Java e uma linguagem de especificação baseada em anotações podemos considerar uma especificação, não apenas como uma especificação de software, mas como um repositório de decisões de projeto que podem ser úteis para o desenvolvedor, principalmente aqueles que estão tendo o primeiro contato com o código-fonte.

O amplo conjunto de anotações JML, a grande maioria baseada no Projeto por Contrato de Eiffel, são utilizadas para realizar especificações de comportamento dos métodos e classes. Nas próximas seções descreveremos algumas das assertivas.

2.3.2 Assertiva `requires`

A assertiva `requires` é utilizada por uma especificação para estabelecer as condições que devem ser satisfeitas para a utilização de um método. Em uma abordagem voltada a componentes de software utilizando o Projeto por Contrato, a assertiva `requires` é utilizada para estabelecer as pré-condições de utilização daquele componente.

A sintaxe para utilização da assertiva `requires` é a presença da palavra chave `requires` seguida de uma lista de operações lógicas que devem ser satisfeitas para o correto funcionamento do bloco de código-fonte especificado.

A Figura 7 mostra a utilização da assertiva `requires` em um método Java que calcula a soma de dois valores inteiros e não-negativos. De acordo com a especificação desse método, os dois argumentos, neste exemplo, `a` e `b` devem ser inteiros não-negativos. A construção sintática `//@` indica o início de uma declaração JML e deve ser reconhecida pelas ferramentas que oferecem suporte à linguagem.

```
//@ requires a >= 0 && b >= 0;  
public int soma( int a, int b ){  
  
    return a + b;  
  
}
```

Figura 7. Uso da assertiva `requires`.

A assertiva `requires`, geralmente, é utilizada em conjunção com outras assertivas JML de forma que elas, juntas, especifiquem o correto funcionamento de um determinado componente.

Dentro da definição de uma assertiva `requires` podemos também utilizar operadores específicos da linguagem da implementação a fim de construir uma especificação que descreva, com o máximo de precisão, o correto funcionamento do bloco de código cujo comportamento está sendo descrito. A Figura 8 ilustra a presença de uma dessas construções. Na especificação da pré-condição do método temos a presença do operador `instanceof`, da linguagem Java. Na especificação o uso do operador mostra uma condição que deve ser estabelecida na qual o argumento de um método, que retorna um intervalo de uma `String`, seja uma instância da classe `java.lang.String`.

```
//@ requires (str != null) && (str instanceof String);  
public String retornaIntervaloString(String str){  
  
    return str.substring(0,2);  
  
}
```

Figura 8. Exemplo do uso de `instanceof` em uma especificação JML.

2.3.3 Assertiva `assignable`

A utilização da assertiva `assignable` está associada à alteração de valores de variáveis que podem ocorrer dentro de um bloco de código no qual a especificação precede. Em termos práticos, estabelece quais variáveis que podem ser alvo de atribuição durante a execução do método. Se dentro do bloco de código-fonte que está sendo especificado tivermos a atribuição de um valor a uma variável, a assertiva `assignable` é utilizada para definir a variável que receberá o valor.

O uso da assertiva `assignable` pode também estar associado à utilização do predicado `\nothing`. Quando as duas construções são utilizadas juntas, a semântica equivalente à construção diz que, dentro do bloco de código-fonte especificado, não temos atribuição de valores a quaisquer variáveis. A Figura 9 ilustra o uso da assertiva `assignable` no mesmo método ilustrado na Figura 8 o qual retorna um intervalo de uma `String`. A assertiva `assignable`, mostrada na figura, diz que neste método nenhuma atribuição é realizada.

```
//@ requires (str != null) && (str instanceof String);  
//@ assignable \nothing;  
public String retornaIntervaloString(String str){  
  
    return str.substring(0,2);  
  
}
```

Figura 9. Utilização da assertiva assignable.

2.3.4 Invariantes em JML

Os invariantes descrevem propriedades que são sempre verdadeiras em todo estado visível de um bloco de código. Um invariante estabelece uma condição global que deve ser sempre preservada em todos os estados possivelmente alcançados por uma instância de uma classe.

A Figura 10 mostra a definição de um invariante de classe. O invariante definido especifica a propriedade de imutabilidade de uma pessoa ter um peso que sempre é maior que 0.

```
public class Pessoa{  
    private int peso;  
    ...  
    //@ invariant peso > 0;  
    ...  
}
```

Figura 10. Classe Pessoa e a presença de um Invariante.

2.3.5 A assertiva ensures

A assertiva ensures especifica uma pós-condição normal, ou seja, uma propriedade que é verdadeira ao final da chamada de método específica, caso uma exceção não seja levantada.

Para definirmos uma especificação de uma pós-condição em JML devemos utilizar a cláusula ensures seguida de uma lista de operações lógicas que devem ser satisfeitas ao final da execução do bloco de código que está sendo especificado.

A Figura 11 ilustra o uso da assertiva ensures. Na figura podemos visualizar a utilização da assertiva em uma especificação de um método que retorna a soma de 2 inteiros positivos. A pós-condição que deve ser satisfeita por esse método é que o valor retornado pelo método, representado pelo predicado `\result` seja igual à soma dos dois argumentos.

```
//@ ensures \result == a + b;  
public int soma(int a, int b){  
    return a + b;  
}
```

Figura 11. Uso da assertiva ensures.

2.3.6 Assertiva `signals`

Como dito anteriormente, uma boa especificação de software deve ser o mais completa possível no sentido de poder representar todos os possíveis resultados que a execução do bloco de código-fonte que está sendo especificado possa alcançar.

Um dos possíveis estados que um bloco de código-fonte pode alcançar é uma terminação anormal devido a alguma situação inesperada que, por ventura, possa ter ocorrido. Na linguagem Java uma situação excepcional é representada por uma instância da classe `Exception` onde essa classe possui métodos capazes de nos informar a causa que motivou a execução a entrar em um estado excepcional.

Na linguagem de especificação JML uma situação excepcional ou anormal pode ser especificada através da utilização da cláusula `signals`. A sintaxe para utilização da assertiva `signals` é a utilização da palavra-chave `signals` seguida pelo tipo de exceção a ser lançada caso a situação ocorra e uma lista de operações lógicas que devem ser satisfeitas para que a situação excepcional possa acontecer.

A Figura 12 mostra o uso da assertiva `signals` que determina que uma instância da classe `Exception` deverá ser lançada se o valor retornado pelo método não for igual ao valor especificado pela pós-condição, ou seja, o valor representado pelo predicado `\result` for diferente da soma dos 2 argumentos.

```
//@ ensures \result == a + b;  
//@ signals (Exception) \result != a + b;  
public int soma( int a, int b ){  
    return a + b;  
}
```

Figura 12. Uso da assertiva `signals`.

2.3.7 Tipos de especificações JML

As especificações de software utilizando a linguagem JML podem ser classificadas de várias maneiras, de acordo com o tipo de palavra-chave utilizada na especificação para identificar o tipo de comportamento.

O primeiro tipo de especificação diz respeito às especificações “leves” (*lightweight*). Neste tipo de especificação não é utilizada qualquer palavra-chave que identifique o comportamento desejado. É a forma mais simples de especificação, cuja principal característica é a sua incompletude, ou seja, a especificação conterá apenas o que se deseja especificar. Todas as especificações presentes nas figuras até o presente momento podem ser consideradas “leves”, tendo em vista que foram especificadas apenas as situações necessárias para ilustrar os exemplos.

O outro tipo de especificação presente na linguagem JML é uma especificação “pesada” (*heavyweight*). As especificações “pesadas” são caracterizadas por apresentar um conteúdo mais completo do que as especificações “leves”. Elas utilizam, também, palavras-chave que identificam o tipo de comportamento que está sendo especificado.

As palavras-chave podem ser de três tipos: `normal_behaviour`, `behaviour`, `exceptional_behaviour`. A palavra-chave `normal_behaviour` é utilizada para definirmos uma especificação de software onde as situações excepcionais que o software pode assumir não estejam presentes na especificação. A Figura 13 mostra a definição de uma especificação “pesada” de um comportamento sem situações excepcionais. Como podemos ver na figura, esse tipo de especificação não deve conter a presença de uma assertiva `signals` que

especifica uma situação excepcional. A presença de uma assertiva `signals` invalida o uso de `normal_behaviour` em uma especificação de software.

```
/*@ normal_behaviour
   @ ensures \result == a + b;
   @*/
public int soma( int a, int b ){
    return a + b;
}
```

Figura 13. O uso de `normal_behaviour`.

A segunda palavra-chave utilizada para especificar o comportamento de um bloco de código-fonte é a palavra `behaviour`. Essa palavra é utilizada para construir a forma mais completa de uma especificação de software que envolve as propriedades que constituem as condições normais de funcionamento e as condições excepcionais que podem vir a ocorrer. As condições normais podem ser caracterizadas pelas pré e pós-condições que devem ser satisfeitas. As condições anormais podem ser representadas pelas exceções que podem, por ventura, ser lançadas quando alguma condição não for satisfeita.

A Figura 14 ilustra o uso da palavra-chave `behaviour` em uma especificação de um método que retorna a soma de dois valores inteiros e positivos. Na especificação em questão temos a presença das assertivas `requires` e `ensures`, as quais definem, respectivamente, a pré e pós-condição de utilização do método. A assertiva `signals` é usada para especificar situações anormais que, possam vir a acontecer. Na Figura 14, o uso de `signals` estabelece que o método lançará uma exceção se o valor retornado pelo método não for igual a soma dos seus argumentos.

```
/*@ behaviour
   @ requires ( a >= 0 ) && ( b >= 0 );
   @ assignable \nothing;
   @ ensures \result == a + b;
   @ signals (Exception) \result != a + b;
   @*/
public int soma( int a, int b ){
    return a + b;
}
```

Figura 14. Uso da palavra-chave `behaviour`.

A última palavra-chave utilizada para definir uma especificação “pesada” é a palavra `exceptional_behaviour`. Com a presença dessa palavra-chave a especificação deverá conter apenas o comportamento assumido pelo bloco de código-fonte que está sendo especificado quando uma situação excepcional ocorre.

A Figura 15 mostra a utilização da palavra-chave `exceptional_behaviour`. Na figura, a situação especificada mostra o que pode acontecer quando uma situação excepcional ocorre, ou seja, é lançada uma instância da classe `Exception` quando a pós-condição não é satisfeita.

```
/*@ exceptional_behaviour
   @ signals (Exception) \result != a + b;
   @*/
public int soma( int a, int b ){
    return a + b;
}
```

Figura 15. Uso da palavra-chave `exceptional_behaviour`.

O tipo de palavra-chave a ser utilizada quando estamos definindo uma especificação para um bloco de código-fonte depende apenas do nível de detalhamento que queremos expressar em uma especificação. Se a especificação for a mais completa possível, ou seja, deve conter as condições normais que uma aplicação deve alcançar durante a sua execução e as condições anormais, utilizamos a palavra-chave `behaviour`. Se o objetivo for construir uma especificação que relate apenas o comportamento normal daquilo que está sendo especificado, a palavra-chave que deve ser utilizada é `normal_behaviour`. Quando é preciso especificar somente as situações excepcionais que podem acontecer em algum estado da execução de uma aplicação, deve ser utilizada a palavra-chave `exceptional_behaviour`.

2.3.8 Predicados JML

Para complementar as anotações utilizadas para estabelecer uma especificação de software que obedeça as regras do Projeto por Contrato, temos a presença de vários predicados JML. Esses predicados são utilizados para representar uma série de situações. Temos predicados utilizados para representar o resultado de um método, para representar o valor antigo de uma variável, predicados que definem a existência de uma variável, dentre outros.

Nesta seção abordaremos, resumidamente, alguns desses predicados com o objetivo de mostrar as possibilidades de realizar uma especificação JML utilizando as assertivas básicas da linguagem, juntamente com os predicados que serão explorados nesta seção. Como o objetivo desse trabalho é o de realizar uma especificação simples de um subconjunto da API J2ME, imposição estabelecida pela pequena quantidade de memória presente nos dispositivos, não entraremos em detalhes sobre todas as construções JML. O objetivo é definir uma especificação simples, correta, e com o menor número de comentários JML possível.

Um dos predicados, já utilizado anteriormente em exemplos de especificação JML, é o `\result`. Esse predicado representa o retorno de um método e pode ser utilizado em pós-condições. Um outro predicado utilizado é o `\old`. Esse predicado representa o antigo valor de uma expressão em um estado anterior ao estado atual. Esse predicado também é comumente utilizado em pós-condições.

O predicado `\not_modified` é utilizado para especificar uma situação onde os valores de uma expressão se mantêm inalterados em dois estados subseqüentes, ou seja, o valor assumido por uma expressão em um estado anterior permanece inalterado no próximo estado.

Os quantificadores universais e existenciais também estão presentes na linguagem JML. O quantificador universal `\forall` é utilizado para definir propriedades que acontecem em um conjunto de valores. A Figura 16 mostra a utilização do predicado `\forall` que estabelece a propriedade de ordenação de um array `a`. Na figura, temos especificado que para todo o conjunto de índices do array, todo elemento posterior é maior que o seu antecedente. O quantificador existencial é representado na linguagem JML pela palavra-chave `\exists`. Esse predicado especifica uma situação que acontece, pelo menos, uma vez. A utilização do predicado existencial está comumente associada ao uso do quantificador universal.

```
(\forall int i,j; 0<=i && i<j && j<10;a[i]<a[j])
```

Figura 16. Utilização do predicado `\forall`. (Fonte: Manual de Referência de JML [36])

Uma outra importante construção da linguagem de especificação JML é a presença de quantificadores que podem ser utilizados para especificação de expressões numéricas. O quantificador `\sum` representa a soma de uma seqüência de valores. O quantificador `\product` é utilizado para representar o produto de uma seqüência de valores. Dois outros quantificadores relevantes para a linguagem JML são aqueles que devolvem o máximo e o mínimo valor contido em uma seqüência de números. Eles são representados pelas palavras-chaves `\max` e `\min`, respectivamente. A Figura 17 ilustra o uso de cada um desses quantificadores aplicados a seqüências de números.

```
(\sum int i; 0 <= i && i < 5; i) == 0 + 1 + 2 + 3 + 4  
(\product int i; 0 < i && i < 5; i) == 1 * 2 * 3 * 4  
(\max int i; 0 <= i && i < 5; i) == 4  
(\min int i; 0 <= i && i < 5; i-1) == -1
```

Figura 17. Quantificadores aplicados a números. (Fonte: Manual de Referência de JML [36])

Como a linguagem JML possui uma ampla variedade de anotações, usadas para construir especificações formais de softwares, não temos como descrever cada uma delas em seus detalhes completos. O objetivo dessa seção foi mostrar aos leitores a quantidade de situações que podem ser especificadas utilizando-se assertivas, predicados e quantificadores da linguagem JML. As construções da linguagem JML mostradas até este momento são suficientes para a implementação de uma especificação de software J2ME simples, correta e pequena. Para uma maior compreensão das outras anotações JML o manual de referência da linguagem é a fonte primária de consultas[36].

2.4 Ferramentas JML

O projeto JML não foi construído apenas para prover anotações capazes de desenvolver uma especificação formal de software. Aliado as anotações temos uma suíte de ferramentas utilizadas para uma série de finalidades específicas. Todas essas ferramentas recebem como entrada, códigos-fonte Java especificados com as anotações JML [11].

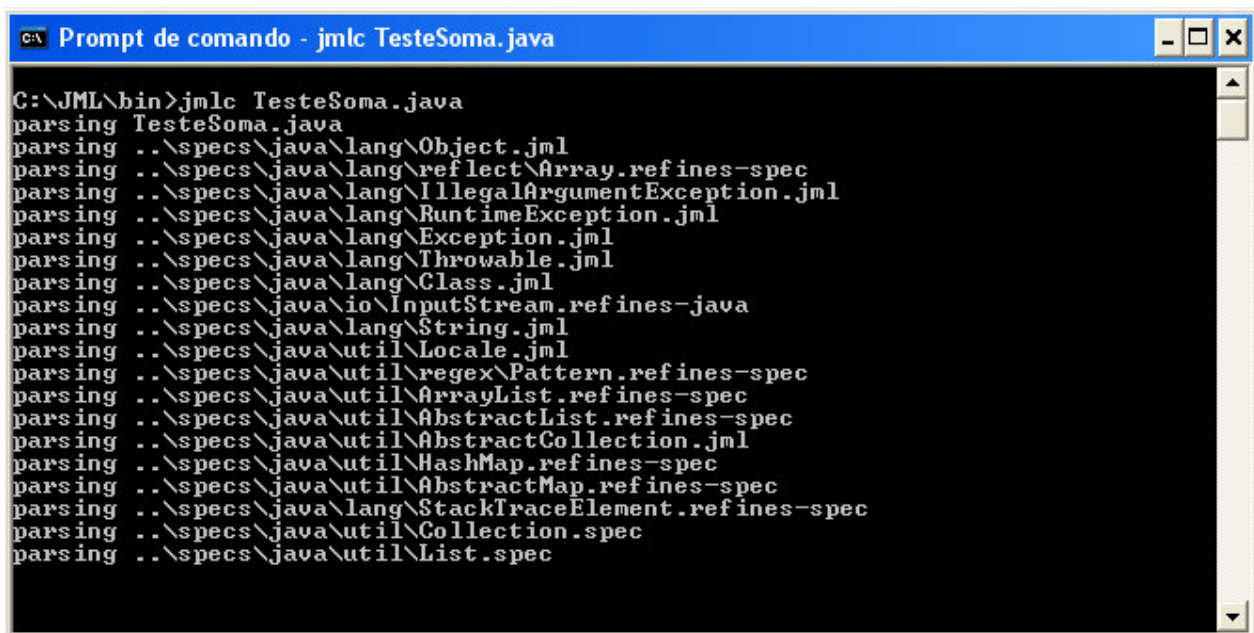
Dentro desse conjunto de ferramentas temos aquelas responsáveis por realizar um processo de compilação do código-fonte anotado com JML, ferramentas para realizar a execução do código-fonte anotado e verificar se as pré e pós-condições foram satisfeitas ou não e uma ferramenta para geração de documentação Java a partir de uma especificação JML escrita. Outras duas ferramentas que merecem destaque são aquelas responsáveis pela geração e execução de classes de teste para o JUnit [24]. Uma das ferramentas é utilizada para gerar o código que deve estar contido dentro das classes de teste e a outra é responsável pela execução dessas classes de teste em conjunto com o *framework* para testes unitários JUnit. Nas próximas seções descreveremos um pouco de cada uma dessas ferramentas, bem como o cenário ideal para utilização das mesmas.

2.4.1 O compilador JML (jmlc)

Para que uma especificação de software usando as anotações da linguagem JML sejam reconhecidas como construções da própria linguagem JML é preciso que o código-fonte seja compilado utilizando um compilador diferente daquele utilizado para compilar código Java puro. Se o compilador utilizado for o `javac`, o compilador Java, as anotações JML serão descartadas, ou seja, serão reconhecidas como blocos de comentários Java.

O processo de compilação empregado na linguagem JML além de reconhecer as anotações inerentes à linguagem, adiciona, aos *bytecodes*, código específico da linguagem JML. Esse fato acarreta em um arquivo de classe com um tamanho maior que se fosse compilado apenas com o compilador Java. Esse arquivo de classe de tamanho maior, em J2ME, é um grande problema que pode ser justificado pela quantidade de memória reduzida da maioria dos dispositivos móveis.

A Figura 18 mostra a execução do compilador JML para o processo de compilação de uma classe Java que tem o objetivo de somar dois números inteiros positivos. Essa compilação é realizada a partir do código-fonte que contém um método para somar 2 valores e como saída desse processo, temos um arquivo de classe (*.class*) Java composto pelas modificações realizadas pelo compilador JML durante o processo de compilação.



```

C:\JML\bin>jmlc TesteSoma.java
parsing TesteSoma.java
parsing ..\specs\java\lang\Object.jml
parsing ..\specs\java\lang\reflect\Array.refines-spec
parsing ..\specs\java\lang\IllegalArgumentException.jml
parsing ..\specs\java\lang\RuntimeException.jml
parsing ..\specs\java\lang\Exception.jml
parsing ..\specs\java\lang\Throwable.jml
parsing ..\specs\java\lang\Class.jml
parsing ..\specs\java\io\InputStream.refines-java
parsing ..\specs\java\lang\String.jml
parsing ..\specs\java\util\Locale.jml
parsing ..\specs\java\util\regex\Pattern.refines-spec
parsing ..\specs\java\util\ArrayList.refines-spec
parsing ..\specs\java\util\AbstractList.refines-spec
parsing ..\specs\java\util\AbstractCollection.jml
parsing ..\specs\java\util\HashMap.refines-spec
parsing ..\specs\java\util\AbstractMap.refines-spec
parsing ..\specs\java\lang\StackTraceElement.refines-spec
parsing ..\specs\java\util\Collection.spec
parsing ..\specs\java\util>List.spec
  
```

Figura 18. Utilização do compilador jmlc.

Apesar de o compilador JML reconhecer as anotações presentes no código-fonte a ser compilado, a checagem de corretude das aplicações não acontece na fase da compilação. Durante essa fase, todo trabalho realizado pelo compilador é adicionar as construções JML que serão verificadas em tempo de execução aos *bytecodes* e também alertar ao desenvolvedor sobre algum erro que possa impossibilitar a compilação. O Apêndice A fornece os códigos-fontes de uma pequena aplicação desenvolvida em J2ME que funciona como uma calculadora simples e o *bytecodes* (*.class*) gerado pelo compilador `jmlc` em sua versão para J2ME. Percebe-se que o código-fonte original é extremamente diferente do seu correspondente quando decompilamos o arquivo de classe gerado pelo compilador `jmlc`. Além disso, o tamanho do código-fonte é bem maior que o original.

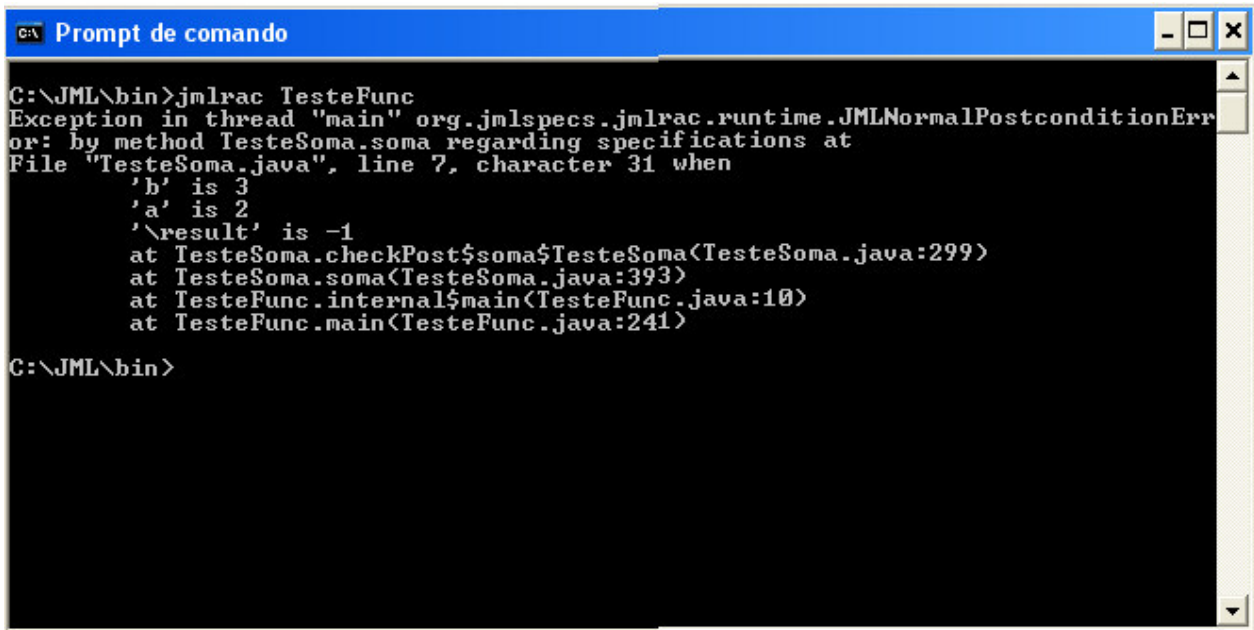
2.4.2 A Ferramenta `jmlrac`

A ferramenta `jmlrac` é responsável por verificar as inconsistências entre o código-fonte especificado e a especificação escrita em JML. Essa verificação é realizada em tempo de execução. Enquanto o bloco de código-fonte que foi especificado encontra-se em execução, o `jmlrac` verifica se aquilo que foi especificado corresponde exatamente ao que ocorre na execução corrente. Se todas as condições forem satisfeitas e nenhuma inconsistência for encontrada, nenhuma mensagem de erro é retornada. Caso contrário, a ferramenta mostra a existência de um erro de inconsistência entre o código-fonte e a especificação.

Esta ferramenta não somente informa que ocorreu uma violação entre o código-fonte e a especificação, mas também disponibiliza informações, estáticas e dinâmicas, que podem levar o usuário a rapidamente solucionar a inconsistência. Como informações estáticas podemos considerar o posicionamento do erro dentro do código-fonte. As informações dinâmicas são, por exemplo, o valor das variáveis. Se tomarmos como exemplo o método que calcula a soma de dois inteiros positivos e passarmos como argumentos os valores 2 e 3, a informação dinâmica seria o retorno da função quando comparado ao resultado esperado pela especificação. A informação estática seria, se houvesse alguma violação que invalidasse a pós-condição do método, a localização de onde ocorreu o erro.

Como o `jmlrac` realiza essas verificações dinamicamente, a ferramenta não é capaz de identificar todo tipo de violação. O fato que justifica essa afirmação é o princípio de funcionamento da ferramenta. Como a verificação de corretude é realizada a partir das contra-provas, a ferramenta poderá não utilizar todas as possíveis contra-provas e, com isso, não informar alguns erros.

A Figura 19 ilustra o uso da ferramenta de verificação dinâmica de asserções `jmlrac` para verificar o método que realiza a soma de dois inteiros positivos. Nesse método, propositadamente, alteramos o resultado retornado pelo método de forma a forçar uma violação da pós-condição, ao invés de retornar a soma dos argumentos `a` e `b`, o valor retornado é a diferença entre eles. Como os valores passados como argumentos para o método foram 2 e 3, respectivamente, teríamos como resultado esperado o valor igual a 5 que corresponde à soma dos dois argumentos. No entanto, o valor retornado pelo método, representado pela presença do predicado `\result`, foi igual a -1. O valor -1 foi obtido da violação que, propositadamente, forçamos a acontecer dentro da implementação do método que realiza a soma. O método agora, calcula a diferença entre os argumentos `a` e `b` que receberam, respectivamente, os valores inteiros e positivos 2 e 3, obtendo o valor de retorno igual a -1, violando a pós-condição, que garante que o resultado seja igual à soma de `a` com `b`.



```

C:\JML\bin>jmlrac TesteFunc
Exception in thread "main" org.jmlspecs.jmlrac.runtime.JMLNormalPostconditionError: by method TesteSoma.soma regarding specifications at
File "TesteSoma.java", line 7, character 31 when
    'b' is 3
    'a' is 2
    '\result' is -1
    at TesteSoma.checkPost$soma$TesteSoma<TesteSoma.java:299>
    at TesteSoma.soma<TesteSoma.java:393>
    at TesteFunc.internal$main<TesteFunc.java:10>
    at TesteFunc.main<TesteFunc.java:241>

C:\JML\bin>

```

Figura 19. Uso da ferramenta jmlrac.

2.4.3 Ferramenta jmldoc

Os desenvolvedores que utilizam a linguagem Java estão bastante acostumados a buscar informações sobre como utilizar as classes presentes na plataforma através das APIs da linguagem. Esse tipo de documentação contém toda a informação necessária para a perfeita utilização de uma classe. As informações presentes nas APIs relatam os graus de dependências das classes, quais as suas interfaces, quais os seus atributos, quais métodos podem ser usados, quais as exceções que podem ser lançadas por cada um desses métodos, dentre outras.

As documentações *javadocs* disponíveis nas distribuições da plataforma Java estão disponíveis no formato HTML, o que possibilita um formato de documentação estruturado, permitindo que a mesma esteja eletronicamente disponível.

A linguagem JML também suporta esse tipo de documentação. A presença da ferramenta *jmldoc* torna possível a construção de uma documentação baseada em HTML a partir de uma especificação JML. Essa documentação é útil porque proporciona um meio mais comum, para desenvolvedores que já tiveram contato com os *javadocs*, de colher informações a respeito de uma determinada especificação de software.

A Figura 20 ilustra uma parte da documentação gerada pelo *jmldoc* para uma classe que contém um método para realizar a soma de dois valores inteiros e positivos. Podemos observar na figura que a especificação está contida também na documentação gerada.

Method Detail

soma

```
public int soma(int a,
               int b)
```

Specifications:

```
public normal_behavior
requires (a > 0 && b > 0);
assignable \nothing;
ensures \result == (a+b);
```

Package [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

Figura 20. Documentação gerada pelo `jml.doc`.

2.4.4 As ferramentas de testes unitários

Uma das práticas mais comuns na Engenharia de Software, principalmente para equipes que adotam uma metodologia de desenvolvimento dirigido a testes ou métodos ágeis de desenvolvimento, é a realização de testes de software.

Os testes de software podem ser classificados de acordo com a sua finalidade. Existem os testes funcionais que são realizados para verificar a qualidade do software no realizar de alguma tarefa, ou seja, verificar se o software realiza corretamente a funcionalidade para a qual foi projetado. Outro tipo de teste bastante comum, principalmente em aplicações em rede, são os teste de carga, onde os servidores são postos a operar em seus limites a fim de verificar a robustez dos mesmos perante a execução de uma aplicação.

Dentre as várias modalidades de teste, temos uma que pode ser beneficiada através das anotações JML: os testes de unidade ou unitários. Os teste unitários são realizados em pequenas unidades de um aplicativo com o objetivo de testar a corretude dessas pequenas partes isoladamente. A grande vantagem desse tipo de teste é que eles podem acontecer a qualquer momento do processo de desenvolvimento além da vantagem de reduzir grandes erros em um projeto devido aos outros testes de unidade anteriores serem completados com êxito.

Um dos grandes conflitos existentes na Engenharia de Software diz respeito à quantidade de tempo que pode ser empregado na escrita de classes de teste em relação a produtividade. Algumas vertentes argumentam que o tempo que, por ventura, possa ser empregado escrevendo-se classes de teste para garantir um desenvolvimento incremental e sem acúmulo de erros, deve ser utilizado para desenvolver o próprio software em prol de testes finais. Outra vertente defende a idéia de que os testes de unidade são importantes para um bom projeto mesmo que esses “tomem” um tempo extra do desenvolvimento [37].

Um projeto que conta com a presença de anotações JML dentro de seu código-fonte pode ajudar a equacionar um pouco esse conflito. Na suíte de ferramentas da linguagem JML temos duas ferramentas que possuem uma integração com um *framework* para realização de testes unitários chamado JUnit. O JUnit é um *framework* totalmente implementado em Java e também,

um dos mais utilizados para realização de testes unitários, sendo parte integrante dos maiores ambientes de desenvolvimento para a linguagem Java como o Eclipse e o JBuilder, por exemplo.

A nova implementação do JUnit é fortemente baseada no novo recurso de anotações Java. Dessa forma, podemos ter em uma mesma classe Java, anotações para serem executadas no JUnit e também anotações para ferramentas JML .

A ferramenta *jmlunit* é utilizada para realizar a geração de classes de teste para o JUnit. Essa geração é feita a partir do código-fonte anotado com JML, resultando em duas classes: uma onde devem ser adicionados os dados para o teste e a outra classe, que herda da primeira, é responsável pela execução do teste. No Apêndice B mostramos um exemplo das classes geradas pelo *jmlunit* para uma classe que contém um método para realizar a soma de dois inteiros positivos.

A outra ferramenta para testes unitários da suíte JML é o *jml-junit*. O papel atribuído a essa ferramenta é realizar a comunicação com o JUnit a fim de iniciar a execução da classe de teste gerada. Dessa forma, podemos observar que o desenvolvimento apoiado nas anotações JML tem a possibilidade de ser verificado em tempo de execução, através de ferramentas como o *jmlrac*, ou também através da realização de testes unitários e não é preciso privar tanto tempo de desenvolvimento para implementação de classes de teste.

2.4.5 ESC/Java 2

A ferramenta ESC/Java 2 [12][41], uma continuação do projeto ESC/Java da Compaq Research, foi projetada para realização de verificação estática e estendida a partir de um código-fonte Java. Ela é uma ferramenta estática pelo fato de não realizar a verificação do código em tempo de execução, ou seja, a verificação é feita utilizando a especificação sem precisar executar qualquer código-fonte.

Por outro lado, ESC/Java 2 é considerada uma ferramenta de verificação estendida pelo fato de conseguir realizar verificações que vão além das que podem ser realizadas pelos compiladores convencionais, como o compilador Java. Essa verificação diferenciada será descrita quando apresentarmos o princípio de funcionamento dessa ferramenta.

O projeto ESC/Java 2 tem algumas vantagens com relação ao seu antecessor, o projeto ESC/Java da Compaq Research. A primeira vantagem é o suporte a versões mais novas do J2SDK, como a versão 1.4. O ESC/Java apenas suportava versões mais antigas do ambiente Java.

A outra grande vantagem, e a que mais trouxe benefícios para a aplicação dos Métodos Formais a projetos de software, é o suporte direto a código-fonte anotado com JML. As ferramentas que pertencem à suíte JML realizam a verificação de código-fonte anotado com JML, mas, apenas em tempo de execução, os possíveis erros podem ser visualizados pelo usuário. Com o suporte provido pelo ESC/Java 2 à linguagem JML, essa verificação pode ser realizada estaticamente, inclusive com a exibição de possíveis erros pertencentes ao código. A grande contribuição desse fato é que podemos verificar propriedades dentro de um sistema antes de colocarmos o sistema em um ambiente de produção.

Por ESC/Java 2 ser uma ferramenta que realiza uma grande quantidade de cálculos lógicos em tempo de compilação, uma de suas decisões de projeto, foi o fato de deixar com que alguns possíveis erros não sejam verificados pelo programa. Essa decisão, propositadamente, foi considerada em função do desempenho da ferramenta, ou seja, se a ferramenta tivesse o objetivo de verificar todas as situações que pudessem gerar erros em um aplicativo, teríamos um tempo enorme de compilação onde seriam verificadas todas as possíveis histórias de execução de um programa.

O princípio de funcionamento do ESC/Java 2 é semelhante ao processo do seu antecessor. Temos como entrada para a ferramenta, O código-fonte em que pretendemos verificar é entrada para a ferramenta. Como o objetivo da ferramenta é verificar estaticamente o comportamento do código-fonte, geralmente é utilizada a linguagem JML para definição da especificação equivalente ao comportamento inerente ao código-fonte.

O próximo passo, após entrarmos com o código-fonte Java para a ferramenta, é uma conversão de formato realizado pelo compilador do ESC/Java 2. O código-fonte Java, especificado com JML, é transformado em estruturas lógicas que representem, com precisão, o comportamento das funcionalidades especificadas no código original. Após esse processo de compilação, temos o início do processo de verificação estática.

Uma verificação estática é realizada checando todas as histórias de execuções que podem ser realizadas durante o processo de execução de um programa. Uma história de execução de um programa é uma seqüência de comandos que pode ser executada pelo mesmo. A ferramenta de verificação estática analisa todas as possíveis combinações de comandos que podem ser disparadas e assim, verifica as possibilidades de erros acontecerem.

O último passo de uma verificação estática usando o ESC/Java 2 é a geração da saída da verificação. Se o ESC/Java 2 não encontrar possibilidades de erros em cada história de execução do código-fonte, a verificação não levanta erros. Quando algum erro, por ventura, é encontrado dentro do código-fonte, o ESC/Java 2 apresenta informações que descrevem o tipo do erro, a causa do erro e a localização do mesmo dentro do código.

A Figura 21 mostra as etapas realizadas durante uma verificação estática com o ESC/Java 2. Primeiramente o código-fonte especificado com a linguagem JML é passado como entrada para o verificador. Em seguida o ESC/Java realiza uma transformação a partir código-fonte para obter uma representação lógica da especificação JML. Essa representação lógica é verificada através de um provador de teoremas interno ao ESC/Java 2 e, por fim, o resultado da verificação é informado ao usuário.

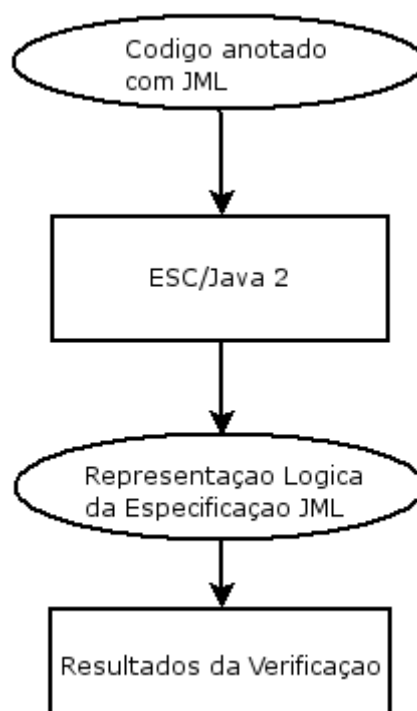


Figura 21. Princípio de funcionamento do ESC/Java 2.

Capítulo 3

Especificação Comportamental

O objetivo deste capítulo é mostrar como foram selecionadas as classes da plataforma J2ME para serem especificadas com a linguagem JML, as decisões de projeto que foram tomadas durante o decorrer deste trabalho e como foram realizadas as especificações.

O objetivo deste trabalho é definir uma especificação formal para um subconjunto da plataforma J2ME em contrapartida à especificação informal proposta pela Sun, através das JSR's (*Java Specification Request*) para a plataforma J2ME. As próximas seções desse capítulo abordarão a forma de implementação desse trabalho, mostrando alguns exemplos da integração das anotações JML em classes da plataforma J2ME.

3.1 O subconjunto especificado

Devido à quantidade relativamente grande de classes contidas na API J2ME e a limitação do tempo de finalização desse trabalho, foi preciso estabelecer um subconjunto de classes da plataforma a fim de realizarmos uma especificação formal utilizando a linguagem JML.

A estratégia utilizada para a escolha dessas classes foi selecionar um conjunto de classes que estivessem presentes na maioria das aplicações J2ME. O MIDP 2.0, que já foi discutido no capítulo anterior, é uma parte da plataforma J2ME que, na grande maioria das aplicações, tem participação no código-fonte.

Uma outra estratégia foi escolher classes que possuíssem pouca dependência em relação a projetos utilizando APIs de fabricantes específicos. A justificativa para essa estratégia é que, quando se desenvolvem projetos J2ME existem grandes possibilidades de que uma aplicação que funcione perfeitamente para um determinado modelo de equipamento móvel, não obtenha o mesmo desempenho ou funcione em outro modelo. Essa limitação faz com que aplicações J2ME

precisem se tornar sistema parametrizados, ou seja, é necessário, além da implementação, que é comum aos modelos de equipamentos móveis, realizar implementações específicas para os equipamentos envolvidos.

A API MIDP 2.0 foi a parte da plataforma J2ME escolhida para ser especificada formalmente utilizando a linguagem JML. Essa API pode estar presente, a depender do tipo de aplicação desenvolvida, em mais de 50% do código-fonte da aplicação sendo, assim, relevante a sua especificação comportamental. O MIDP contém classes de interface gráfica, as quais nessa API representam componentes que serão exibidos na tela de aparelhos celulares, contém mecanismos de persistência de dados, uma API genérica para conexão, dentre outras. O MIDP é uma parte com poucas dependências e capaz de implementar um núcleo para o desenvolvimento de aplicativos J2ME.

Ressaltamos que não especificamos todas as classes do MIDP 2.0. Das classes de interface gráfica, contidas no pacote `javax.microedition.lcdui`, foram especificadas apenas aquelas que representam componentes já construídos, ou seja, aqueles que o desenvolvedor utiliza sem modificar a sua construção. Com isso, as classes `javax.microedition.lcdui.Canvas`, a qual possibilita ao usuário definir novos componentes de interface gráfica, e `javax.microedition.lcdui.Graphics`, a qual funciona como o objeto personalizador dos componentes criados com `Canvas`, não serão especificadas nesse momento.

O motivo de especificar apenas algumas classes de interface gráfica é que o foco está na especificação de classes que possam levar o desenvolvedor a identificar erros críticos em um projeto. Os erros causados pelo mau uso de componentes de interface gráfica não representam erros graves em um projeto, tendo em vista que as interfaces gráficas, geralmente, propiciam o fornecimento de entradas para as alicações por um usuário e, outras vezes, exibem informações da aplicação. As classes que não são de interface gráfica, geralmente, realizam operações mais complexas e, por isso, podem acarretar erros mais graves para uma aplicação.

As classes contidas no pacote `javax.microedition.games` também não foram especificadas. O motivo é que elas são utilizadas para fins muito específicos como o desenvolvimentos de jogos com J2ME e não estão presentes na grande maioria das aplicações J2ME. A Tabela 1 contém todas as classes e interfaces da API MIDP2.0 que foram especificadas durante o nosso trabalho. Algumas classes como, `java.lang.String`, não foram especificadas porque as suas especificações já são distribuídas com a linguagem JML.

Tabela 1. Subconjunto da plataforma J2ME escolhido para especificação.

<code>javax.microedition.lcdui</code>	Alert AlertType Choice ChoiceGroup Command CommandListener DateField Display Displayable Form Item List Screen TextBox Ticker
<code>javax.microedition.rms</code>	DataConverter InvalidRecordIDException RecordComparator

	RecordEnumeration RecordEnumerationImpl RecordFilter RecordListener RecordStore RecordStoreException RecordStoreFullException RecordStoreNotFoundException RecordStoreNotOpenException
javax.microedition.io	CommConnection HttpConnection HttpsConnection PushRegistry SecureConnection SecurityInfo ServerSocketConnection SocketConnection UDPDatagramConnection
javax.microedition.pki	Certificate CertificateException
javax.microedition.midlet	MIDlet MIDletProxy MIDletStateChangeException MIDletStateMapImpl
java.lang	Class Runtime System IllegalStateException
java.util	Timer TimerTask

3.2 O formato das especificações

Durante o desenvolvimento desse trabalho realizamos, também, um estudo detalhado das formas como devem ser construídas as especificações JML. Analisamos as especificações que estão disponíveis na página oficial da Linguagem, relativas ao trabalho relacionado ao nosso que é a implementação da API de JavaCards utilizando a linguagem JML [38][39] em busca de entender a forma como foram desenvolvidas as especificações.

Existem várias formas de especificarmos um código-fonte utilizando a linguagem JML. A primeira dessas formas é realizarmos a implementação dentro do arquivo fonte, ou seja, no arquivo contendo a implementação Java. Essa forma é mais comum quando pretendemos realizar uma política de testes e verificação na aplicação que está sendo desenvolvida. A desvantagem desse formato de especificação é que ela desfavorece as práticas do Projeto por Contrato, onde não é preciso que o desenvolvedor conheça detalhes de implementação. É apenas necessário conhecer o que ele pode fazer com aquele componente e quais as condições para utilizá-los.

Uma outra forma de realizarmos uma especificação é separar as anotações JML da implementação em si. Esse método é bastante favorável ao Projeto por Contrato, uma vez que a única informação disponibilizada ao desenvolvedor são as operações que eles podem realizar com o componente e as condições que devem ser satisfeitas para o correto funcionamento do mesmo. Em JML esse formato de especificação é disponibilizada em um arquivo que possui a extensão .jml. O conteúdo desse arquivo é bastante semelhante a definição de uma interface Java. A única

diferença é que não utilizamos a palavra-chave `interface` e, também, a presença das anotações JML que definem as condições de uso dos métodos.

Esse segundo formato de especificação formal de software com JML tem uma vantagem bastante relevante: a separação estabelecida entre o código-fonte do aplicativo e a especificação JML. Com isso, não é necessário alterar o código-fonte da aplicação especificada. A especificação pode estar completamente desacoplada do código-fonte. Esse foi o formato de especificação adotado no nosso trabalho e também, o formato adotado para a especificação da API JavaCard.

3.3 A especificação formal do MIDP 2.0

O nosso trabalho de especificar formalmente o comportamento de um subconjunto da plataforma J2ME tem um caráter propriamente científico e um de seus objetivos é proporcionar uma motivação para a integração dos métodos formais em projetos de software, com a finalidade de desenvolver software de maior qualidade e com a menor quantidade de erros possíveis. Essa seção descreve a implementação das especificações JML para a maioria das classes do MIDP 2.0.

Devido às limitações de recursos dos equipamentos móveis que suportam a plataforma J2ME, principalmente as limitações de memória disponível, tivemos que assumir o desafio de implementar especificações que sejam as mais simples possíveis, e que representassem, com completude, o comportamento do aplicativo que queremos especificar. A justificativa para essas especificações de tamanho limitado é seguir a filosofia das limitações impostas pela plataforma J2ME. Como existem limitações de recursos, as classes precisam implementar as suas funcionalidades de forma simples e correta. Se as especificações JML para as classes J2ME forem muito complexas, isso significa um aumento no tamanho dos arquivos contendo as especificações, e isso pode causar problemas de desempenho para a aplicação. Esse problema fica bem caracterizado quando a especificação JML encontra-se junto ao código-fonte especificado pois, se compilarmos o código-fonte com o compilador JML, o compilador modifica o arquivo de classe e esse arquivo fica bem maior por causa da presença das anotações JML que são inseridas. Esse problema está referenciado no Apêndice A.

Antes de iniciarmos a descrição de como foram realizadas as especificações, é preciso esclarecer algumas decisões de projeto que foram tomadas para a implementação desse trabalho. A primeira decisão diz respeito à definição do que deverá ser especificado com as anotações JML dentro de uma classe J2ME. Como o nosso objetivo maior é definir uma especificação formal que sirva como benefício para o desenvolvimento de software J2ME com maior qualidade, é extremamente importante definir o que será especificado.

Com a construção das especificações JML para o subconjunto de classes do MIDP em nosso trabalho, pensaremos nas classes especificadas como repositórios de funcionalidades onde as anotações JML regem as condições para o perfeito funcionamento das funcionalidades presentes no repositório. Nas implementações desenvolvidas para as classes da plataforma J2ME, mais especificamente para o MIDP 2.0, encontramos várias construções sintáticas presentes no código-fonte como, por exemplo, a presença de blocos estáticos, classes internas (também conhecidas, na língua inglesa, como *inner classes*), os atributos das classes, os métodos públicos, os métodos privados, etc.

Este trabalho preocupa-se em especificar apenas as partes presentes nas classes visíveis ao desenvolvedor, ou seja, as funcionalidades que o desenvolvedor pode dispor quando está desenvolvendo as suas aplicações J2ME. Logo, os blocos de código estáticos, os vários métodos

privados que não são visualizados pelo desenvolvedor e são utilizados apenas por outros métodos e as classes internas, não serão especificados com a linguagem JML.

A fim de esclarecer como realizamos as especificações JML para classes do MIDP, utilizaremos, como exemplo, o processo de especificação da interface `javax.microedition.io.HttpConnection`. A interface `HttpConnection` faz parte do *framework* de conexão genérica implementado pela plataforma J2ME. O objetivo dessa interface é estabelecer as funcionalidades para realização de conexão estabelecida utilizando o protocolo HTTP [40], um dos tipos de conexão suportados pela plataforma J2ME.

A implementação foi organizada em uma estrutura de diretórios bastante semelhante àquela utilizada para o empacotamento das classes providas pelos implementadores do MIDP 2.0. Assim, as nossas especificações estarão localizadas no mesmo lugar em que se encontram os códigos-fontes originais disponibilizados pela Sun Microsystems.

Anteriormente foi dito que as especificações deveriam ser as mais simples possíveis por causa das limitações de memória dos dispositivos móveis. Pode-se observar, agora na prática e ao longo desse seção, que as especificações construídas seguem um padrão de simplicidade mas, sem abrir mão de especificar todo o comportamento proposto pela implementação dos métodos.

O procedimento utilizado para realizar a especificação das classes, utilizadas nesse trabalho, inicia-se a partir do código-fonte original da API J2ME. Foi realizada uma cópia da implementação original da classe, e adicionada a um arquivo contendo a extensão `.jml`, que é característica para o formato da especificação que decidimos utilizar.

Tomamos como base para a definição das anotações JML que correspondem à especificação comportamental das classes, a documentação informal da API MIDP 2.0 [19] disponibilizada pela Sun Microsystems. Se a nossa opção fosse não utilizar a documentação do MIDP 2.0, também poderíamos ter realizado as especificações. A decisão de construir as especificações de acordo com a documentação pode ser justificada por um dos objetivos do nosso trabalho, que é a construção de uma especificação formal para um subconjunto da API J2ME em concordância com a sua especificação informal, como a entendemos.

A nossa primeira construção sintática utilizando a linguagem JML inserida na especificação da interface `HttpConnection` é uma declaração de importação como podemos verificar abaixo:

```
package javax.microedition.io;
//@ model import org.jmlspecs.*;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.String;
import javax.microedition.io.ContentConnection;
```

Iniciando a nossa especificação da interface `HttpConnection`, foi adicionada, primeiramente, uma das anotações da linguagem JML que é utilizada para definirmos que a especificação comportamental pode utilizar as classes pertencentes à linguagem JML, onde essas classes podem ser utilizadas dentro de pré e pós-condições. Na especificação, a anotação `//@ model import org.jmlspecs.*;`, indica que as classes da linguagem JML estão sendo utilizadas na especificação mas, essa importação só faz sentido para as ferramentas que suportam a linguagem JML. O compilador Java não reconhece essa anotação e por isso, as classes JML não podem ser utilizadas. A palavra reservada `model` é utilizada apenas para informar que essa importação é válida somente para especificações.

Uma vez estabelecida a importação das classes contidas na linguagem JML, é preciso adicionar um tipo de comentário JML na linha que define a assinatura de uma classe conforme pode ser visto abaixo:

```
public /*@ pure @*/ interface HttpConnection extends ContentConnection {
```

A anotação `/*@ pure @*/` é utilizada para informar que a nossa especificação não deverá conter qualquer implementação para seus métodos e construtores. Como estamos tratando da especificação de uma interface, é natural que essa não contenha implementação alguma, apenas as assinaturas dos métodos e os atributos. Porém, nas classes da API MIDP 2.0 que foram especificadas, a presença desse tipo de anotação indica que não deve haver qualquer implementação no escopo da classe.

A seguir, temos uma lista de constantes que representam informações específicas do protocolo HTTP. Essas constantes são utilizadas pelos vários métodos da interface. Abaixo podemos visualizar algumas dessas constantes. As outras constantes que foram omitidas representam códigos de estado de uma conexão HTTP, assim como a constante `HTTP_OK`.

```
public final static String HEAD = "HEAD";  
public final static String GET = "GET";  
public final static String POST = "POST";  
public static final int HTTP_OK = 200;  
...
```

O trabalho de especificação comportamental propriamente dito da interface `HttpConnection`, tem início neste momento com o desenvolvimento das especificações para os métodos públicos da interface.

O primeiro método a ser especificado é o método `getURL`, o qual é utilizado para devolver a URL (*Uniform Research Location*) completa utilizada na conexão, como podemos visualizar abaixo:

```
/*@ behaviour  
@ assignable \nothing;  
@ ensures (\result instanceof String);  
@*/  
public String getURL();
```

Iniciamos a especificação informando o tipo de comportamento que queremos especificar. Essa especificação é um pouco redundante com relação ao tipo de retorno do método mas, esse tipo de construção JML também está presente no trabalho relacionado ao esse, que será discutido no Capítulo 5. Utilizando a palavra reservada `behaviour`, estamos declarando que a especificação é a mais completa possível, ou seja, englobando as condições normais e anormais de terminação da execução. Em seguida utilizamos a cláusula `assignable`, a qual indica se ocorre alguma atribuição dentro do método e qual a expressão que realiza a atribuição. Nesse caso, como estamos tratando de um método de acesso a dados, não temos nenhuma atribuição realizada. Esse fato é representado pela presença do quantificador `\nothing`, o qual informa que nenhuma atribuição estará sendo realizada.

Como o método `getURL` não contém parâmetros e não lança exceções, não utilizaremos na especificação as assertivas `requires` e `signals`, utilizadas para representar pré-condições

e o lançamento de exceções, respectivamente. A pós-condição desse método, representada pela presença da assertiva `ensures`, informa que o resultado da execução do método, que é referenciado implicitamente pelo quantificador `\result`, deve ser do tipo `java.lang.String`.

Os métodos `getProtocol` e `getHost` são especificados da mesma forma que `getURL`. Então, para evitar a repetição de conceitos, mostraremos apenas as especificações construídas sem explicarmos, novamente. A Figura 22 ilustra as duas especificações.

```
/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result instanceof String);
@*/
public String getProtocol();

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result instanceof String);
@*/
public String getHost();
```

Figura 22. Os métodos `getProtocol` e `getHost`.

O método `getProtocol` é utilizado para obter o protocolo contido na URL. Através do método `getHost` podemos ter acesso ao endereço ou ao nome atribuído ao computador hospedeiro. As informações são recuperadas em um objeto `java.lang.String`.

As especificações desenvolvidas para os métodos `getFile`, `getRef` e `getQuery` também são idênticas. O método `getFile` é utilizado para obter o nome de arquivo do URL. O método `getRef` é utilizado para obter a parte relativa à referência do URL e o método `getQuery` é utilizado para obter a string de consulta, mas só é válido se a requisição for do tipo GET. As especificações relativas a esses três métodos encontram-se na Figura 23, mas, serão comentadas apenas a especificação do método `getFile`:

```
/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result == null) ||
  @ (\result instanceof String);
@*/
public String getFile();

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result == null) ||
  @ (\result instanceof String);
@*/
public String getRef();

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result == null) ||
  @ (\result instanceof String);
@*/
public String getQuery();
```

Figura 23. Os métodos `getFile`, `getRef` e `getQuery`.

Para o método `getFile`, e também para os outros dois métodos, as especificações diferem, daquelas que foram mostradas antes, apenas na pós-condição. Agora, a assertiva `ensures` vem acompanhada de uma lista de condições lógicas. O motivo para esse fato, é que, tomando como exemplo o método `getFile`, pode não haver um nome de arquivo associado à URL e com isso, o retorno do método pode ser um valor nulo. O mesmo acontece para os outros dois métodos. Dessa forma, poderemos obter dois valores de retorno para os métodos: um valor nulo ou um objeto da classe `java.lang.String`.

O método `getPort` é utilizado em um aplicativo J2ME para obter a porta utilizada quando do estabelecimento de uma conexão HTTP. Abaixo segue a especificação para tal método:

```
/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result == 80) || ( \result >= 0 );
  */
public int getPort();
```

Podemos verificar, porém, que o método `getPort` fornece, como retorno, um tipo primitivo `int`. Na especificação da pós-condição do método, estamos garantindo que o valor inteiro retornado pelo método é maior ou igual a 0, onde esse valor representa o número da porta, ou então, é retornado o valor *default* da porta que, para uma conexão utilizando o protocolo HTTP, é a porta de número 80.

As duas próximas especificações que iremos apresentar dizem respeito aos métodos acessores do método utilizado para uma requisição HTTP. Os dois métodos são `getRequestMethod` e `setRequestMethod` e podemos visualizar as especificações na Figura 24.

```
/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result.equals(HEAD)) || (\result.equals(GET)) ||
  @ ( \result.equals(POST));
  */
public String getRequestMethod();

/*@ behaviour
  @ requires (method.equals(HEAD)) || (method.equals(GET)) ||
  @ method.equals(POST));
  @ signals ( IOException ex )
  @ ex.getMessage() != null && ((!method.equals(HEAD)) &&
  @ (!method.equals(GET)) && ( !method.equals(POST)));
  */
public void setRequestMethod(String method) throws IOException;
```

Figura 24. Os métodos `setRequestMethod` e `getRequestMethod`.

Na especificação do método `getRequestMethod` um ponto que merece ser comentado é a sua pós-condição. De acordo com a especificação do MIDP 2.0 o retorno desse método é um objeto `java.lang.String` que indica qual o método utilizado na requisição HTTP. Dessa forma, a assertiva `ensures` garante que o retorno desse método terá valor igual a uma das três

opções de constantes, presentes na interface `URLConnection`, que representam os três tipos de métodos para requisições HTTP.

Para o método `setRequestMethod`, como não temos nada sendo retornado pelo método, não utilizamos a assertiva `ensures`. A presença do parâmetro `method` no método nos informa que deve ser estabelecida uma pré-condição com relação a esse parâmetro. No caso do nosso método, a restrição de uso imposta pela pré-condição nos diz que devemos passar como argumento para esse método, um objeto `java.lang.String` que deve possuir um valor igual a uma das três constantes presentes na interface `URLConnection` e que representam, cada uma delas, um tipo de método para requisição HTTP.

O método `setRequestMethod` lança uma exceção do tipo `IOException`. Esse é um tipo de terminação anormal de uma execução e é representado em uma especificação comportamental JML pela assertiva `signals`. Na especificação do método, a assertiva `signals` identifica que o método lançará uma exceção do tipo `IOException` e logo em seguida, os motivos que podem ter ocorrido para que a exceção fosse disparada. No caso do nosso exemplo, para que este tipo de exceção ocorra, o argumento que o método recebeu não teve o valor igual a uma das três constantes presentes na interface `URLConnection` e que representam os possíveis métodos para uma requisição HTTP.

Em seguida, temos a presença dos dois métodos acessores para as propriedades de uma requisição HTTP, o método `getRequestProperty` e `setRequestProperty`. As propriedades de uma requisição HTTP são pares, nomes e valores, que são, geralmente, configurados pelos usuários para obter as informações através de uma requisição. Os vários campos de um formulário HTML, por exemplo, podem ser considerados propriedades de uma requisição. Seguem, na Figura 25, as especificações para esses métodos.

```

/*@ behaviour
  @ assignable \nothing;
  @ requires (key instanceof String);
  @ ensures (\result instanceof String) || ( \result == null );
  @*/
public String getRequestProperty(String key);

/*@ behaviour
  @ requires (key instanceof String) && ( value instanceof String );
  @ signals (IOException ex)
  @   ex.getMessage() != null;
  @*/
public void setRequestProperty(String key, String value)
  throws IOException;

```

Figura 25. Os métodos `getRequestProperty` e `setRequestProperty`.

Podemos ver, nas especificações, que elas seguem o mesmo padrão das especificações anteriores.

Nosso próximo método a ser especificado é o método `getResponseCode`. O papel desse método é obter o código de resposta de uma mensagem. É bastante comum, para nós usuários, requisitarmos através de um *browser* a visualização de uma página HTML e, se a página não for encontrada, recebermos uma mensagem de erro junto a uma numeração. Para esse exemplo, o código de erro é o 404. O nosso método `getResponseCode` retornaria, para o nosso exemplo, o valor inteiro 404. Abaixo, podemos visualizar a especificação comportamental para esse método:

```
/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result == -1) || ( \result >= 200);
  @ signals (IOException ex)
  @   ex.getMessage() != null;
  @*/
public int getResponseCode() throws IOException;
```

Verificando a especificação do método acima, como não há nenhum parâmetro formal na assinatura do método, não há necessidade de estabelecer uma pré-condição. Com relação à pós-condição, temos duas possibilidades de retorno para o método: o valor inteiro -1 , se houver algum erro de conexão, ou um valor inteiro maior ou igual a 200 . Um valor de retorno maior ou igual a 200 nos indica que tal valor de retorno, será igual a uma das constantes da interface `HttpConnection` que possuem o nome iniciando com `HTTP`.

Para evitar a repetição, as especificações para os outros métodos da interface `HttpConnection` seguem os mesmos conceitos que foram apresentados até o presente momento e por isso, decidimos não explicar novamente. A especificação completa da interface `HttpConnection` pode ser encontrada no Apêndice C, onde também podem ser encontradas as demais especificações.

Capítulo 4

Validação da Proposta

Neste capítulo, apresentamos como foram realizadas as verificações de correção das especificações desenvolvidas durante o nosso trabalho. Mostramos, também, como foi possível realizar essas verificações utilizando o verificador estático ESC/Java 2.

Como as verificações seguem todas o mesmo padrão, mostramos aqui apenas as verificações realizadas para uma classe, e uma interface. A interface que será utilizada é a mesma que foi tomada para exemplificar a construção das especificações no Capítulo 3, a interface `javax.microedition.io.HttpConnection`. A classe utilizada nesse capítulo, será a classe `javax.microedition.lcdui.AlertType`.

4.1 Configurações e uso do ESC/Java 2

Utilizamos, o verificador estático estendido ESC/Java 2 [12] para verificar a correção das especificações construídas utilizando a linguagem JML[8].

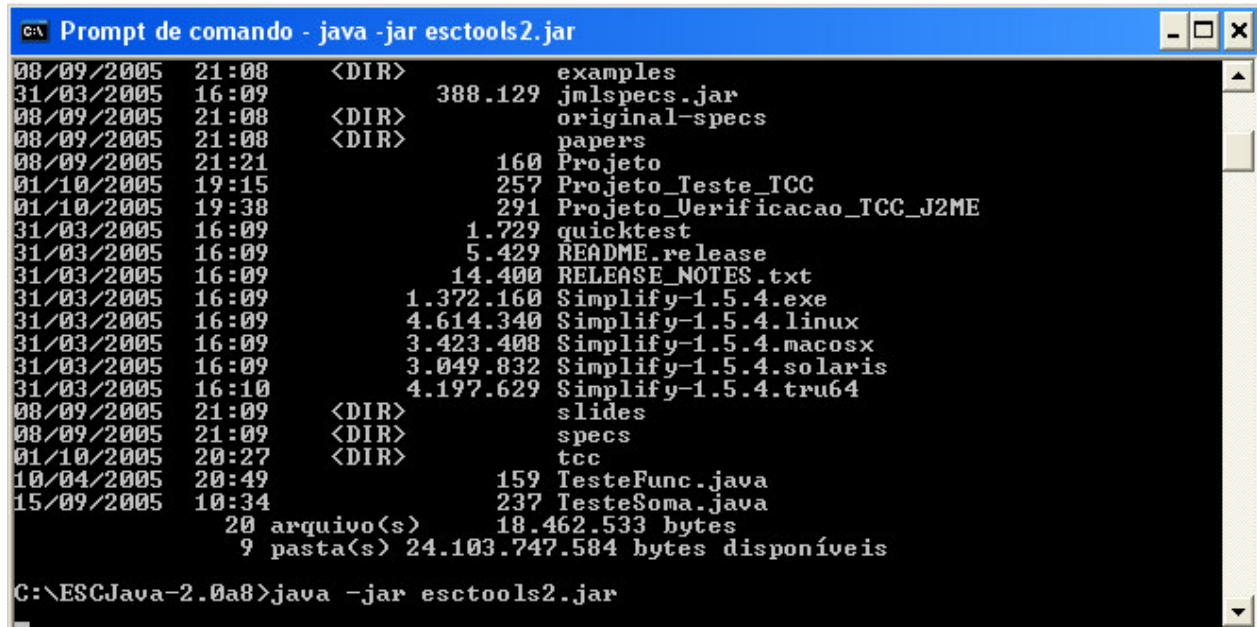
O ESC/Java 2 [12] é uma ferramenta totalmente implementada em Java e uma das suas grandes vantagens, é o suporte oferecido para verificação de especificações comportamentais JML. A versão utilizada nesse trabalho foi a versão 2.0a8 [41].

A ferramenta ESC/Java 2 pode ser utilizada com a sua interface gráfica ou na sua versão console. Em ambas as opções os mesmos resultados devem ser obtidos. No nosso trabalho utilizamos a opção com interface gráfica pela sua facilidade de configuração.

É importante salientar, que essa seção não possui a intenção de ser uma documentação detalhada para uso da ferramenta. O propósito dela é servir como uma rápida referência para

trabalhos futuros que tenham como ferramenta alvo esse verificador. Um outro propósito é tornar simples o uso da ferramenta uma vez que, a sua documentação deixa a desejar nesse quesito.

Para utilização da interface gráfica do ESC/Java 2 devemos acessar o diretório onde a versão da ferramenta foi extraída e executar o arquivo `esctools2.jar`. A Figura 26 ilustra o acesso ao diretório e a execução desse arquivo.



```

C:\> Prompt de comando - java -jar esctools2.jar
08/09/2005 21:08 <DIR>          examples
31/03/2005 16:09          388.129  jmlspecs.jar
08/09/2005 21:08 <DIR>          original-specs
08/09/2005 21:08 <DIR>          papers
08/09/2005 21:21          160      Projeto
01/10/2005 19:15          257      Projeto_Teste_TCC
01/10/2005 19:38          291      Projeto_Verificacao_TCC_J2ME
31/03/2005 16:09          1.729   quicktest
31/03/2005 16:09          5.429   README.release
31/03/2005 16:09          14.400  RELEASE_NOTES.txt
31/03/2005 16:09          1.372.160 Simplify-1.5.4.exe
31/03/2005 16:09          4.614.340 Simplify-1.5.4.linux
31/03/2005 16:09          3.423.408 Simplify-1.5.4.macosx
31/03/2005 16:09          3.049.832 Simplify-1.5.4.solaris
31/03/2005 16:10          4.197.629 Simplify-1.5.4.tru64
08/09/2005 21:09 <DIR>          slides
08/09/2005 21:09 <DIR>          specs
01/10/2005 20:27 <DIR>          tcc
10/04/2005 20:49          159     TesteFunc.java
15/09/2005 10:34          237     TesteSoma.java
          20 arquivo(s) 18.462.533 bytes
          9 pasta(s) 24.103.747.584 bytes disponíveis

C:\ESCJava-2.0a8>java -jar esctools2.jar
  
```

Figura 26. Execução da interface gráfica do ESC/Java 2.

O próximo passo para configurarmos a ferramenta é a configuração do seu caminho de classes. Esse passo é realizado através da interface gráfica na aba *Project Files*. A Figura 27 ilustra como ficou configurado o nosso ambiente para verificações após a configuração do caminho de classes para o nosso trabalho.

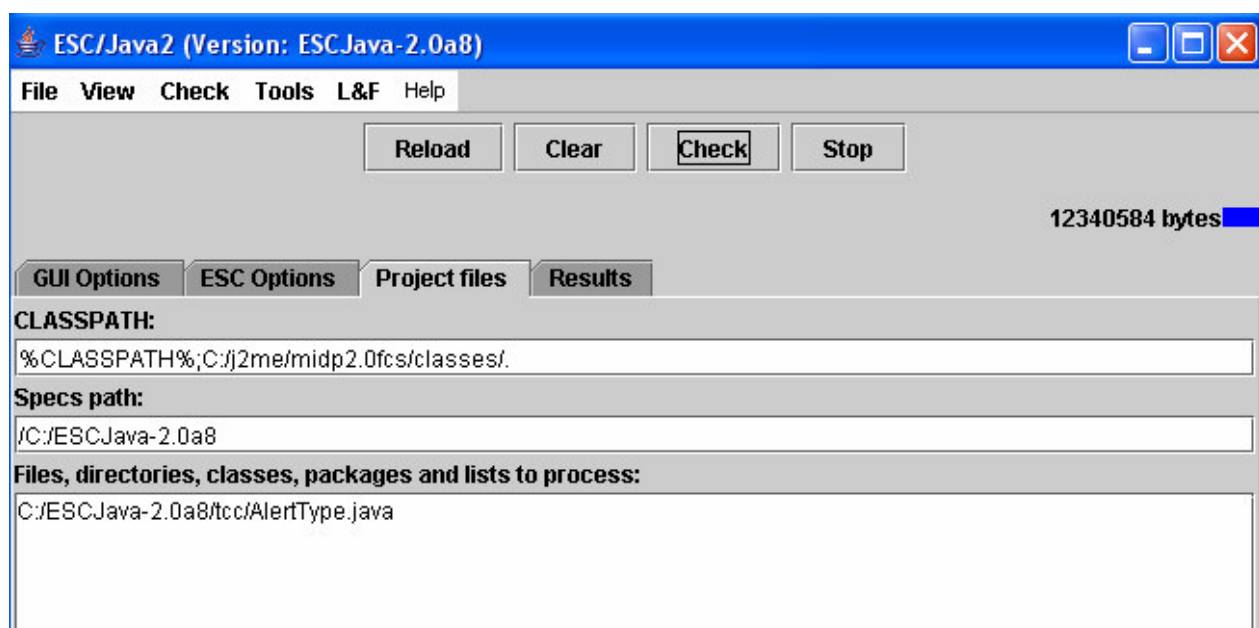


Figura 27. Ambiente configurado para verificação com MIDP 2.0.

Como podemos ver na Figura 27, foi preciso editar o campo de texto CLASSPATH. O conteúdo desse campo deverá referenciar a variável de ambiente CLASSPATH configurada para o seu sistema operacional, e incluímos também as classes do MIDP 2.0.

Visualizando a Figura 27 podemos perceber, ainda, que temos a presença de várias abas. Uma delas, a aba *ProjectFiles*, foi utilizada para a configuração do caminho de classes. A primeira aba, chamada *GUI Options*, possui, basicamente, opções para configurar a exibição de informações para o usuário. Nesse trabalho, nada configuramos. A próxima aba, chamada *ESC Options*, serve para configuração geral da ferramenta. Através dessa aba, a ferramenta reconhece o grau de verificação que o usuário quer realizar sobre suas especificações. A Figura 28 ilustra a configuração da aba *ESC Options* utilizada nesse trabalho.

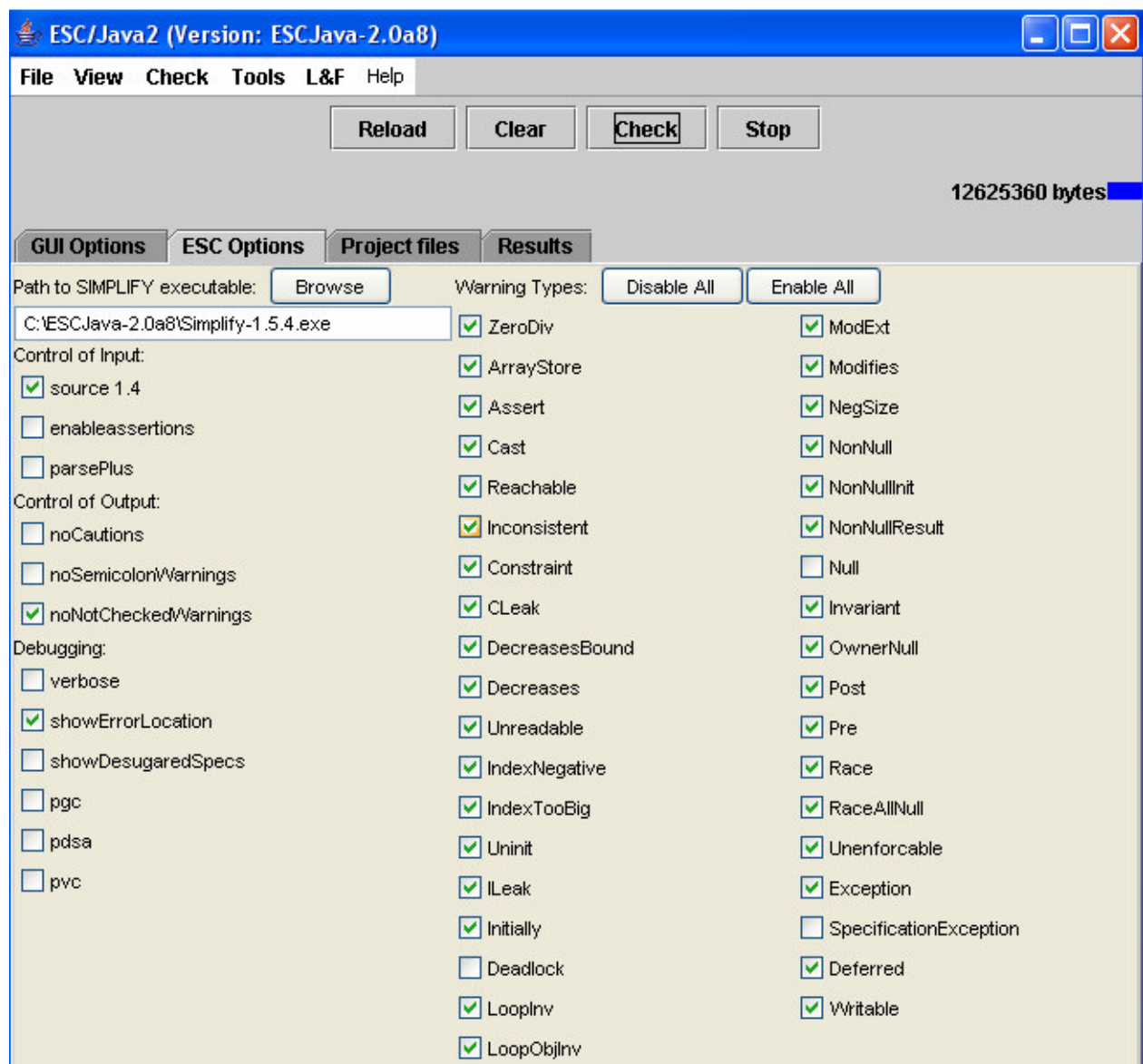


Figura 28. Configuração da aba ESC Options para o nosso trabalho.

Como podemos verificar na Figura 28, através da aba ESC Options, é permitido configurar as opções sobre os tipos de avisos que a ferramenta pode utilizar para informar aos usuários. Para esse trabalho é extremamente importante que opções como, Pre, Post, Invariant, Assert, estejam selecionadas.

A última aba da ferramenta diz respeito aos resultados da verificação. O seu conteúdo será mostrado nas próximas duas seções onde será discutida a verificação estática realizada para algumas das especificações.

4.2 Verificação estática das especificações

Nessa seção serão mostrados os resultados para as verificações estáticas de uma classe e uma interface presente na plataforma J2ME. Inicialmente, será realizada a verificação de corretude para a interface `javax.microedition.io.HttpConnection`.

A ferramenta ESC/Java 2 não suporta arquivos com a extensão `.jml`. Dessa forma, para realizar esse trabalho foi preciso transferir o conteúdo desses arquivos para um arquivo com a extensão `.java`. Um outro problema encontrado quanto ao processo de verificação foi que o ESC/Java 2 não suporta o uso da assertiva `behaviour`, a qual é utilizada para descrever o tipo de comportamento que será especificado.

Tudo que é preciso ser feito para iniciar a verificação formal da especificação JML é informar para o ESC/Java 2 o caminho do arquivo que será testado. Essa informação deve ser inserida dentro do campo de texto nomeado *files, directories, classes, packages, and lists to process*: da aba *Project Files*. Para a especificação da interface `javax.microedition.io.HttpConnection`, foi utilizado o seguinte caminho `C:/ESCJava-2.0a8/tcc/HttpConnection.java`.

A Figura 29 ilustra o resultado da verificação estática da interface `HttpConnection`. Para iniciar a verificação estática basta clicar no botão *Check*.

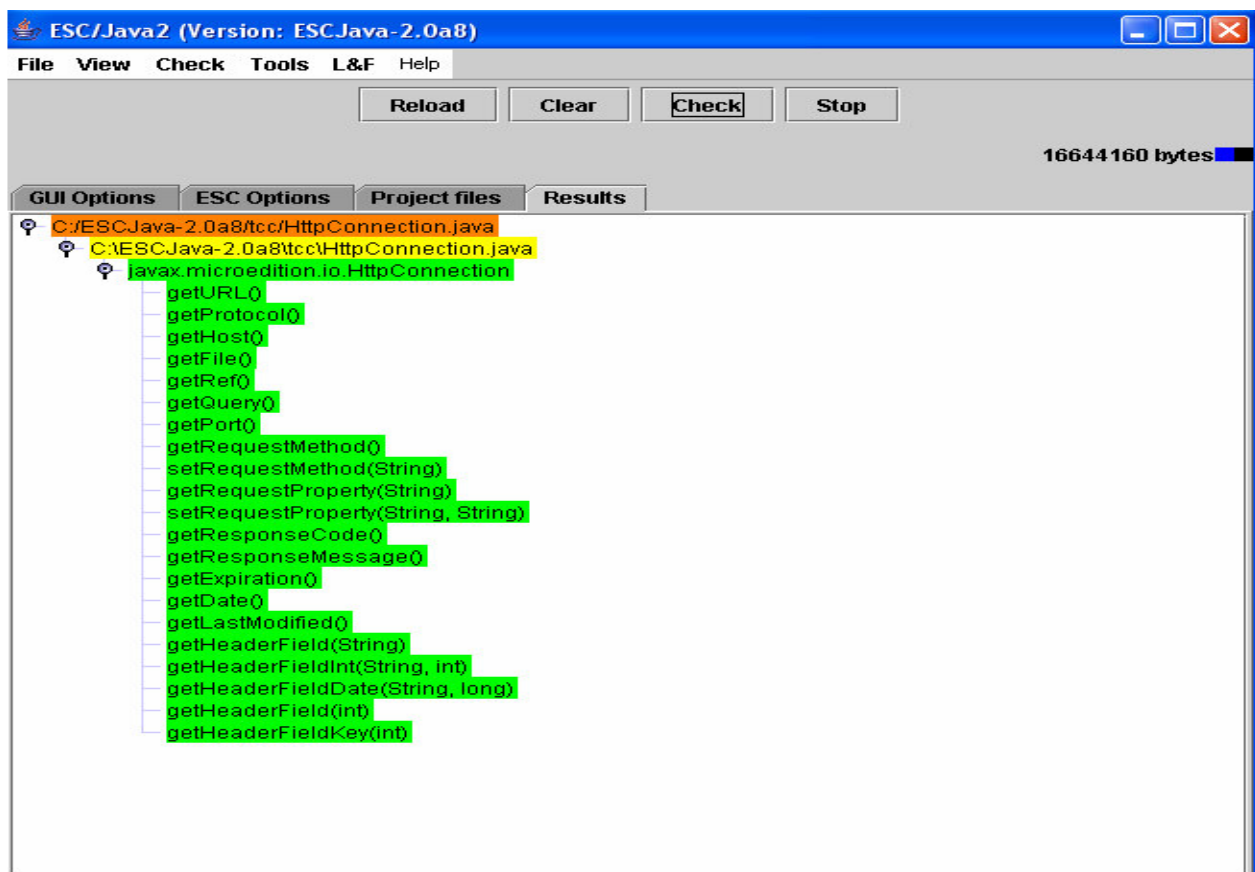


Figura 29. Resultado da verificação estática em `HttpConnection`.

Pode ser observado, na Figura 29, que a verificação da especificação obteve sucesso em todos os métodos dessa interface. Esse fato é evidenciado pela cor verde presente em cada um dos métodos da interface. Caso alguma situação de erro fosse encontrada na especificação, o método onde esse erro foi encontrado apareceria na cor vermelha e uma outra janela seria aberta, trazendo informações sobre o erro encontrado e a sua localização dentro do arquivo.

Para a especificação da classe `javax.microedition.lcdui.AlertType`, a única alteração que deve ser feita no ambiente é o caminho da classe a ser verificada. A nova localização mudará para `C:/ESCJava-2.0a8/tcc/AlertType.java`. A Figura 30 ilustra o resultado para a verificação estática para essa classe. Podem ser observadas na Figura 30 que as especificações obtiveram êxito para todos os métodos da classe.

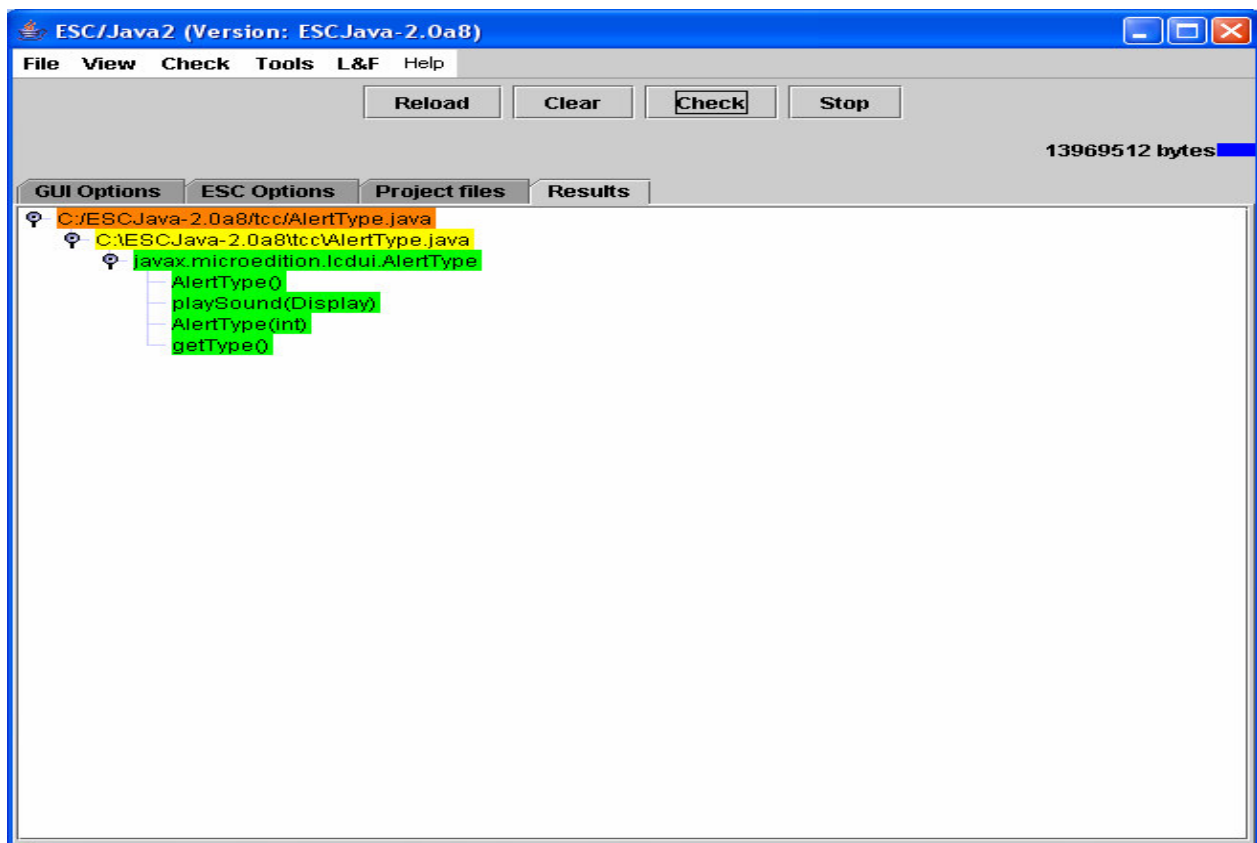


Figura 30. Resultado da verificação estática em `AlertType`.

As demais verificações estáticas para as outras especificações construídas, foram omitidas do texto porque seguem o mesmo princípio das duas acima exemplificadas. As verificações para as especificações construídas obtiveram sucesso.

Capítulo 5

Conclusões e Trabalhos Futuros

Este trabalho teve o objetivo de desenvolver uma especificação formal de software para um subconjunto da plataforma J2ME, o MIDP 2.0. A linguagem escolhida para construir as especificações foi JML, uma linguagem para especificação comportamental de software, baseada em assertivas que descrevem o comportamento da implementação.

A linguagem JML é utilizada para especificar código-fonte construído em Java e é uma linguagem, inerentemente, baseada em Projeto por Contrato e com uso de pré-condições, que devem ser satisfeitas para a execução dos métodos, o estabelecimento de pós-condições que devem ser garantidas pelo método, se as pré-condições forem satisfeitas. A especificação pode ser limitada a um determinado tipo de comportamento, ou seja, podem ser especificadas somente as condições normais de comportamento como, por exemplo, as pré e pós-condições, ou podem ser especificadas as terminações anormais como, por exemplo, as exceções que podem ser lançadas e as causas para que as exceções ocorram.

A parte da plataforma J2ME escolhida para ser especificada foi o MIDP 2.0, com a restrição de não implementar os componentes Canvas e Graphics, do pacote `javax.microedition.lcdui`, porque essas classes servem para propósitos mais específicos, como construção de componentes de interface gráfica customizados. A escolha do MIDP 2.0 foi baseada em dois critérios específicos. O primeiro critério era especificar, como um ponto de partida para o uso de métodos formais na plataforma J2ME, um conjunto dessa plataforma que estivesse presente na grande maioria dos projetos J2ME. O MIDP obedece a esse critério. O segundo critério era que esse subconjunto possuísse baixo grau de dependência em relação às outras partes da plataforma.

O amplo conjunto de ferramentas disponíveis para a linguagem JML tornam viáveis a construção de especificações de software baseadas em JML. Com essas ferramentas podemos realizar verificação em tempo de execução do comportamento do software especificado, pode ser gerada documentação no formato javadocs para uma especificação, e também podem ser geradas classes de testes unitários para serem executadas pelo JUnit. A ferramenta ESC/Java 2 também oferece suporte a JML e seu objetivo é realizar uma verificação estática sobre as especificações presentes no código-fonte, ou seja, para obter informações sobre o comportamento da execução de um método não é preciso executar tal método. O ESC/Java 2 realiza essas verificações em tempo de compilação.

Pode ser concluído para esse trabalho, apesar das limitações impostas pela plataforma J2ME e pela falta de suporte de algumas ferramentas JML para a plataforma como, por exemplo,

o compilador JML, apresentou uma inovação em relação à aplicação de métodos formais, representado pela linguagem JML, para a construção de especificações voltadas a dispositivos móveis de capacidade limitada. Esse trabalho teve muitas limitações que serão discutidas na Seção 5.2, mas cumpriu o seu papel de abrir uma discussão sobre a importância das especificações formais em software que precisam de um cuidado especial quanto à sua correção e segurança, e sua consequência imediata que é a construção de aplicações que atendem fatores de qualidade de software.

5.1 Contribuições

A aplicação de métodos formais a projetos de software tem o objetivo de proporcionar especificações mais precisas, muitas vezes, baseadas em uma notação lógica, como é o caso de linguagens como Z, JML, dentre outras, a fim de permitir a verificação de propriedades dos sistemas apoiados em fundamentos matemáticos.

A principal contribuição é uma especificação formal para uma API J2ME, o MIDP 2.0, em contrapartida à especificação informal disponibilizada pela Sun Microsystems através da JSR (*Java Specification Request*) 118. A forma de utilização de uma especificação informal provida para alguma API Java se dá através de consultas a mesma, com o objetivo de verificar detalhes de implementação, condições de uso dos componentes, decisões de projeto, dentre outras.

Tendo em vista a utilização de uma linguagem formal para especificação comportamental de software e se essa linguagem possuir ferramentas que permitam a verificação mecanizada de correção das especificações, essas considerações representam uma contribuição para o processo de desenvolvimento, pois a garantia da qualidade do software desenvolvido não está mais baseada em modelos semi-formais e que muitas vezes precisam passar por etapas de otimizações, como os modelos gerados pela linguagem UML. Os modelos UML possuem a sua importância em qualquer projeto de software, porém, as especificações utilizando linguagens formais possuem base matemática que permitem verificar propriedades sobre as funcionalidades de um software e com isso, possuem maiores chances de ter como resultado final, um software com maior qualidade.

Outro fator que justifica o ganho na qualidade de um software desenvolvido utilizando-se uma linguagem formal é a premissa de que quanto mais perto de uma linguagem universal, como a lógica, estiver uma especificação de software, menor o grau de ambigüidade presente na especificação. Supondo uma equipe de desenvolvimento de software J2ME que tenha recebido um novo integrante, a possibilidade de o novo membro começar a desenvolver software com correção, partindo de especificações lógicas formais, é bem maior do que se o mesmo novo integrante utilizasse especificações baseadas na linguagem natural como um artefato de desenvolvimento. As chances serão ainda maiores, se a linguagem de especificação possuir uma sintaxe mais próxima da linguagem de implementação.

Esse trabalho estimula a utilização de linguagens formais para implementação de aplicações de um modo geral, não apenas em aplicações para a plataforma J2ME. Essa contribuição poderá levar o desenvolvedor a iniciar o raciocínio antes mesmo da implementação fazendo com que, quando bem elaborada a especificação, a implementação torna-se rápida e com a presença da especificação comportamental, podemos verificar o comportamento da implementação com relativa simplicidade.

A utilização da linguagem JML para especificação formal de aplicações J2ME é um caso onde a qualidade das aplicações pode ser beneficiada. O primeiro grande motivo é que aplicações para dispositivos móveis, como aparelhos celulares, necessitam de um elevado grau de correção

e precisam ser simples. O elevado grau de corretude pode ser justificado por um fator principal: é que se torna custoso para o fornecedor das aplicações disponibilizar a aplicação e essas, futuramente, apresentarem erros, pois as aplicações estão, quase sempre, associadas a um hardware. Exemplificando tal situação, basta imaginar que um grande fabricante de aparelhos celulares encontra um erro e uma aplicação de um dos seus modelos. A forma mais simples de corrigir o erro é tornar disponível a nova versão do software, geralmente através da página do fornecedor, para que os usuários do modelo possam fazer uma atualização, se o modelo possuir tal funcionalidade. Esse tipo de fato é custoso para os dois lados: para o desenvolvedor, que de certa forma perde um pouco do prestígio e diminui o conceito de qualidade junto aos seus usuários, e para o usuário, que se torna quase obrigado a corrigir um erro que não deve haver. Para evitar tal situação, os grandes fornecedores de equipamentos e aplicações móveis estão sempre em busca de profissionais especializados em realização de testes em suas aplicações. Além da procura por esses profissionais, elas estão sempre investindo em pesquisa na busca de novas técnicas que permitam implementar aplicações de maior qualidade.

A linguagem JML beneficia aplicações J2ME em termos de corretude porque pode, através de ferramentas de verificação estática, como o ESC/Java 2, realizar verificações de funcionalidades em uma aplicação, em tempo de compilação, antes mesmo de simular a execução da aplicação nos simuladores J2ME. Os possíveis erros encontrados podem ser corrigidos antes mesmo de realizarmos a simulação em tempo de execução.

A simplicidade que deve ser inserida em especificações para a plataforma J2ME deve ser justificada pela baixa capacidade dos recursos do hardware, principalmente o tamanho de memória disponível. As implementações J2ME precisam ser pequenas para que possam ser executadas nos dispositivos. Se, além da implementação da aplicação, forem construídas especificações complexas, o código-fonte pode ser tornar maior e, conseqüentemente, limitar ainda mais os recursos de memória disponível. A linguagem JML, através de suas assertivas, é capaz de definir especificações simples que descrevam as funcionalidades presentes em um aplicativo J2ME, ou seja, a presença da especificação não aumenta demasiadamente o tamanho do código-fonte.

Na especificação JML para o MIDP 2.0 gerada por esse trabalho, serão disponibilizadas nos arquivos contendo as especificações para as classes do MIDP, apenas as informações necessárias para utilização dessas classes por parte do desenvolvedor J2ME. Através de construções de especificações baseadas no Projeto por Contrato, serão informadas ao desenvolvedor J2ME, as condições que devem ser satisfeitas para utilização de um determinado método de uma classe do MIDP, as condições que serão garantidas pelo método ao final de sua execução se a pré-condição para a sua chamada for satisfeita, dentre outras. O desenvolvedor pode entender um arquivo contendo especificações JML para uma classe do MIDP como um repositório de funcionalidades que ele poderá utilizar e as anotações JML acima da assinatura de cada método, representam o contrato para utilização do método.

A presença de uma especificação formal para o MIDP 2.0 torna-se uma excelente alternativa, ou um complemento, para os desenvolvedores J2ME, principalmente para aqueles que, além de desenvolvedores J2ME, possuem afinidade com notações formais e estejam dispostos a utilizá-las. Essa contribuição já foi anteriormente proporcionada por um trabalho relacionado ao nosso e que será discutido na Seção 5.4.

5.2 Limitações

Durante todo o nosso trabalho de especificação do MIDP utilizando a linguagem JML foram observadas uma série de limitações, algumas já eram esperadas, outras foram descobertas durante o desenvolvimento do projeto. As limitações esperadas foram provenientes das próprias restrições impostas pela plataforma J2ME. Já as limitações encontradas foram decorrentes das ferramentas contidas na suíte de ferramentas JML.

5.2.1 Ferramentas JML

A primeira, e talvez a que mais tem impacto no nosso trabalho, é a falta de suporte para J2ME das ferramentas JML desenvolvidas na Universidade de Iowa. As ferramentas que são disponibilizadas para a linguagem JML, aquelas que estão disponíveis para download na página oficial da linguagem, foram projetadas inicialmente para aplicações Java baseadas no J2SE, ou seja, todo o conjunto de ferramentas funciona perfeitamente para códigos-fonte especificados na linguagem JML com base no J2SE.

Quando trabalhamos com a plataforma J2ME existe um processo de compilação e execução diferentes daqueles realizados com o J2SE. Somente exemplificando o fato acima citado, para compilar uma aplicação Java convencional, é necessário apenas compilar o código-fonte para gerar os *bytecodes* que serão executados pela Máquina Virtual Java (JVM). Durante o processo de compilação e execução de um aplicativo J2ME, existe uma seqüência de passos que possuem funções diferentes. O primeiro passo é a compilação do código-fonte J2ME para a geração dos *bytecodes*, assim como acontece com o J2SE.

Em seguida, é realizada uma pré-verificação nos *bytecodes* gerados, com o objetivo de verificar se os *bytecodes* obedecem aos critérios de restrições de recursos do J2ME (neste caso, as restrições são impostas pelo CLDC). A etapa final, antes de fazer o transferência da aplicação para o dispositivo, é realizar a simulação da aplicação com o objetivo de verificar a corretude das funcionalidades implementadas. Essa simulação realizada possui uma interface gráfica que emula o dispositivo móvel para onde a aplicação será transferida.

Devido às limitações impostas pelos recursos do hardware utilizado, as configurações J2ME como a CLDC, por exemplo, definem implementações de classes Java que são bastante limitadas em relação àquelas contidas no J2SE. Se for comparado o subconjunto do MIDP capaz de realizar operações de entrada e saída com arquivos e realizar conexões genéricas, ao conjunto de classes do J2SE responsáveis por essas mesmas funções, é possível perceber que poucas classes do MIDP cumprem o papel da maioria das classes dos pacotes *java.io* e *java.net*. Essas diferenças entre a plataforma J2SE e a J2ME possuem grande impacto sobre as ferramentas JML.

O primeiro impacto causado para as ferramentas, acontece durante o processo de compilação. Como para realizar a compilação de um aplicativo J2ME é preciso tornar disponível, através do caminho de classes da aplicação, as classes do J2ME (nesse caso, principalmente, as classes do MIDP 2.0) , quando é utilizada a ferramenta de compilação *jmlc* é bem provável que ocorra um conflito de classes causado pelo compilador JML. Isso acontece porque, por exemplo, o compilador JML utiliza uma parte do J2SE para realizar a compilação e também uma parte do J2ME. Um possível erro que pode ser observado é que, se ao compilar um código-fonte J2ME com as classes do MIDP no caminho de classes e utilizando a ferramenta de compilação JML, o compilador JML poderá gerar um erro informando que a classe não está completa. Como exemplo, pode ser argumentado que no pacote *java.lang* do J2SE existe a classe *Object* e no J2ME também existe a mesma classe e no mesmo pacote. Se for realizada a compilação

usando o *jmlc*, a procura pela classe `Object` é realizada, primeiramente, no MIDP e após isso, nas classes do J2SE. Como a classe do MIDP é restrita em relação à mesma classe do J2SE, o compilador JML exibirá uma mensagem de erro informando o acontecimento.

Ainda sobre o compilador JML, existem grandes chances de, se conseguirmos compilar com o *jmlc* algum aplicativo anotado com JML, ao código-fonte será adicionada uma grande quantidade de linhas de código. O impacto causado para a aplicação é que o compilador JML pode adicionar referências a algumas classes do J2SE que não existem dentro das classes do MIDP, por exemplo. Esse fato acarreta uma falha no procedimento que segue a compilação, a pré-verificação, pois quando os *bytecodes* estiverem sendo verificados, erros são gerados porque as classes adicionadas durante o processo de compilação JML não estão presentes no J2ME. Como a pré-verificação não conseguiu ser concluída com êxito, não pode ser realizada a simulação da aplicação. Uma das classes que o compilador JML adiciona aos *bytecodes* é a classe `HashSet`. Essa classe é utilizada para encapsular alguns parâmetros da linguagem.

Como não é possível ainda realizar uma verificação dinâmica nas aplicações J2ME utilizando anotações JML, a única forma encontrada de validar a corretude das aplicações é através do verificador estático estendido ESC/Java 2.

5.2.2 Máquina Virtual K

Como as aplicações J2ME, assim como todas as aplicações Java, precisam de uma máquina virtual para funcionar, uma das decisões de projeto da plataforma J2ME foi a implementação de uma máquina virtual específica, a KVM, capaz de funcionar perfeitamente em dispositivos com recursos limitados. A máquina virtual K, ou simplesmente KVM, desempenha o mesmo papel da máquina virtual Java do J2SE porém, sua implementação é reduzida para atender aos requisitos de desempenho de uma aplicação J2ME. Este fato faz com que muitas das classes disponíveis no J2SE não estejam presentes para a KVM.

O fato de utilizar uma máquina virtual diferente da máquina virtual utilizada pelo J2SE, leva aplicações desenvolvidas em J2ME a não serem suportadas por algumas ferramentas JML como foi discutido acima. Um ponto que deve ficar claro para essa limitação imposta pelo uso de uma máquina virtual diferente é que as especificações JML para aplicativos J2ME não se tornam inviáveis. A verificação comportamental não poderá ser realizada em tempo de execução, mas utilizando-se um verificador estático, ou seja, em tempo de compilação, pode ser realizada a verificação normalmente.

5.2.3 Tamanho das Aplicações

Durante o desenvolvimento de especificações comportamentais para aplicações Java, existe a possibilidade de torná-las complexas com o objetivo de simular ao máximo o comportamento do código-fonte especificado. Quanto mais simples e precisa for a especificação, melhor será para o projeto.

No desenvolvimento de aplicações J2ME temos que ter um cuidado especial com relação ao tamanho das implementações. As implementações precisam ser simplificadas ao máximo em prol do seu bom desempenho. Isso significa que, muitas vezes, conceitos importantes da Engenharia de Software como muitas das boas práticas de programação, encapsulamento, dentre outras, precisam ser deixadas em segundo plano. Esse fato é justificado, também, pela quantidade de memória disponível nos dispositivos para comportar as aplicações.

Ao desenvolvermos especificações JML para a plataforma J2ME, é preciso se preocupar com o tamanho das especificações. Se forem adicionadas uma grande quantidade de assertivas

JML dentro da especificação, o tamanho dos arquivos presentes dentro da aplicação será maior e com isso, ocorrerá uma perda de desempenho na execução da aplicação.

Existe, porém, uma solução alternativa para esse problema em que sua adoção depende da forma como serão disponibilizadas as especificações. A solução seria separar a especificação da implementação. Se essa solução for adotada, a especificação pode ser posta em um arquivo diferente do arquivo da implementação, e pode ser verificado estaticamente o arquivo contendo as especificações. Após validada a especificação, a aplicação poderá ser simulada em tempo de execução sem as anotações pois, essas já foram verificadas estaticamente.

Se o objetivo for disponibilizar a especificação junto com a implementação, o melhor a ser realizado é simplificar ao máximo as especificações para não que não haja perda de desempenho durante a execução. Esse fato torna-se mais crítico em aplicações contendo muitas classes especificadas.

5.3 Dificuldades

Algumas dificuldades foram encontradas durante o desenvolvimento desse trabalho. A carência de documentação para a maioria das ferramentas que suportam a linguagem JML foi a principal delas. A maioria das ferramentas que oferecem suporte à linguagem não disponibilizam informações sobre como podemos utilizá-las. As próprias ferramentas contidas na distribuição JML não são bem documentadas.

A grande maioria dos documentos sobre a linguagem JML e suas ferramentas são artigos científicos que não possuem o propósito de explorar o uso de ferramentas. Alguns desses artigos falam sobre algumas ferramentas como, por exemplo, o artigo que introduz a integração de JML e o JUnit, porém, a informação não possibilita uma compreensão que facilite a utilização.

Uma outra dificuldade encontrada é que não existe uma comunidade ativa de desenvolvedores JML no Brasil. Esse fato pode ser justificado pela baixo índice de utilização de linguagens formais em projetos de software. É bastante difícil estabelecer uma comunicação com pessoas que tenham utilizado a linguagem JML e assim, tirar proveito de seus conhecimentos. Durante o desenvolvimento desse trabalho precisamos entrar em contato duas vezes com membros que trabalharam fortemente no projeto da linguagem JML. O primeiro contato foi estabelecido com o Professor Gary Leavens, um dos criadores do JML, com o objetivo de entender como poderíamos estender as ferramentas JML para suportar aplicativos J2ME da melhor forma possível. O segundo contato foi estabelecido com o Professor Erik Poll, um dos implementadores da API JavaCard [38] com anotações JML, com o objetivo de entender como tal API pode ser melhor utilizada.

Apesar das dificuldades de comunicação com os desenvolvedores, tivemos uma resposta bastante positiva de Gary Leavens com relação a esse trabalho e a intenção de, se for alcançado êxito no suporte a J2ME pelas ferramentas JML, incluir as novas ferramentas no conjunto de ferramentas contidos na distribuição JML.

5.4 Trabalhos Relacionados

Um trabalho relacionado ao descrito por esse texto foi a especificação comportamental da API JavaCard utilizando a linguagem JML [38][39]. Esse trabalho foi desenvolvido por uma equipe liderada por Erik Poll. O trabalho desenvolvido por eles utilizou duas ferramentas para verificar

as especificações construídas: a ferramenta LOOP e o ESC/Java 2, que também foi utilizado no nosso trabalho.

A API JavaCard é uma implementação completamente construída em Java para o desenvolvimento de aplicações para *Smart Cards*. Os *smart cards*, por sua vez, são utilizados em muitos tipos de dispositivos dentre eles, alguns cartões GSM para telefonia celular. Em aplicações que merecem um nível de segurança elevado em termos de integração com um hardware, a API JavaCard tem sido empregada por sua facilidade de utilização e a garantia de que possam ser desenvolvidas aplicações realmente seguras.

No Brasil, existem alguns exemplos do uso de *smart cards* como o e-CPF e o e-CNPJ [42]. O e-CPF é um serviço para declaração de imposto de renda pela Internet. Já o e-CNPJ é utilizado para evitar enfrentar filas em busca de documentações por uma empresa. Os dois serviços, são mantidos pelo Ministério da Fazenda com o objetivo de atender com maior rapidez e sem burocracia, os seus usuários.

O trabalho desenvolvido por Erik Poll e sua equipe, encontra-se disponível para download na página oficial da linguagem JML. No trabalho foram especificadas todas as classes da API JavaCard, que é bem menor que a API J2ME.

As especificações formais, ou seja, a formalização de aspectos de uma aplicação JavaCard também pode ser encontrada no projeto Verificard [43]. O objetivo do projeto Verificard é checar formalizações de aplicações construídas com a API JavaCard e de *bytecodes* JavaCard. Um projeto que é parte integrante do projeto Verificard construiu um provador de teoremas denominado Isabelle [44] que também foi utilizado para a API JavaCard.

5.5 Trabalhos Futuros

Nesta seção apresentamos algumas propostas de trabalhos que surgiram a partir das limitações que foram encontradas durante o desenvolvimento deste trabalho.

5.5.1 Especificação Formal da plataforma J2ME completa

Devido a limitações de prazo para finalização desse trabalho e da grande quantidade de classes contidas na plataforma J2ME, não houve tempo hábil de especificarmos formalmente toda a plataforma J2ME. A escolha de grande parte do MIDP para especificação já foi justificada no Capítulo 3. Aliada as justificativas, é preciso considerar que a escolha também foi motivada pelo tamanho da API justamente pelo fato de o um prazo para terminação do trabalho ser curto.

Como a plataforma J2ME é relativamente grande e possui vertentes diferentes para o tipo de configuração que o dispositivo possui, um dos trabalhos futuros seria a especificação das outras partes da plataforma como, por exemplo, as classes do CLDC e do CDC. A implementação de toda a plataforma J2ME proporcionaria uma grande contribuição para a comunidade de desenvolvedores para pequenos dispositivos e que se interessam por especificações formais de software.

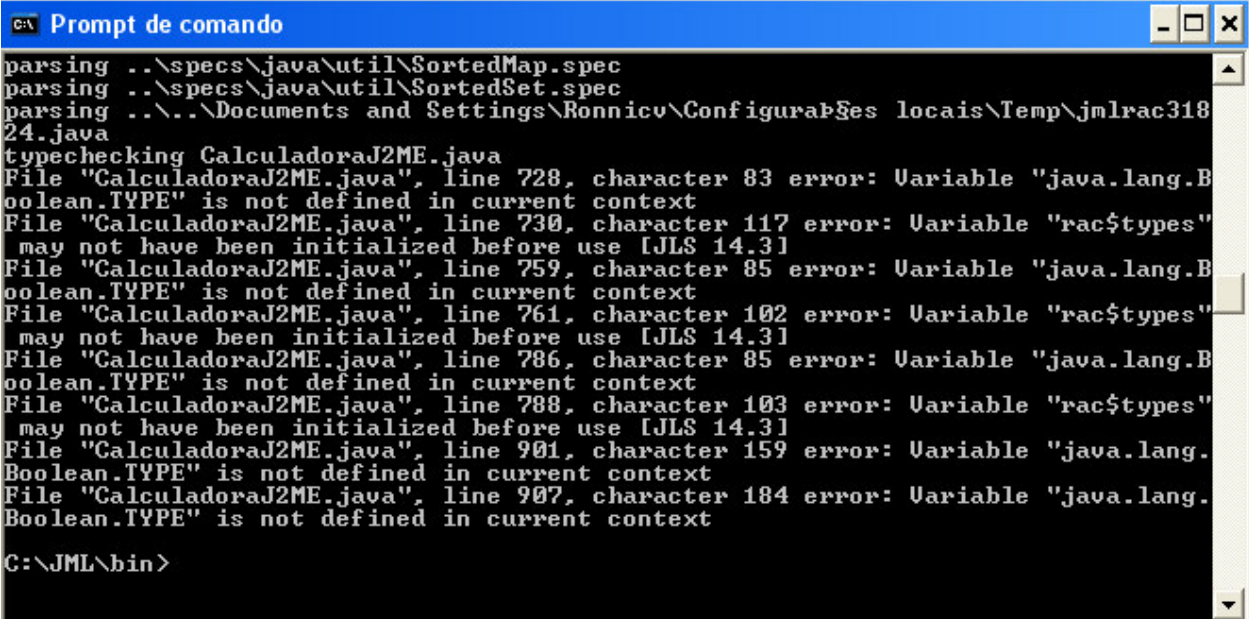
Pode ser investigada, também, a possibilidade de especificação de APIs J2ME proprietárias, ou seja, aquelas API que não fazem parte do núcleo da linguagem e são disponibilizadas pelos fabricantes de equipamentos. Deve-se levar em consideração se os fabricantes fazem objeções para realizar modificações em seus códigos-fonte, ou até mesmo sobre a distribuição das especificações comportamentais que, por ventura, venham a ser construídas.

5.5.2 Compilador JML com suporte a KVM

Uma das grandes possibilidades de trabalhos futuros é um estudo detalhado sobre o compilador utilizado pela linguagem JML e a sua extensão para suportar aplicações J2ME. O grande problema relativo à compilação das aplicações J2ME com o compilador JML é que, basicamente, como o compilador foi projetado para aplicações J2SE, foram definidas especificações JML para suportar as classes do J2SE. Essas especificações fazem parte de um repositório de especificações utilizadas pelo compilador e não são manipuladas pelo usuário.

Com essas especificações pré-definidas, durante a compilação, o compilador JML busca nessas especificações informações sobre o comportamento das classes utilizadas na aplicação sendo compilada e realiza também uma checagem de tipos. Como as classes do J2ME são diferentes daquelas do J2SE como, por exemplo, a classe `java.lang.Boolean` do J2SE possui o atributo `TYPE` enquanto a mesma classe, `java.lang.Boolean`, do J2ME não possui esse atributo e com isso o processo de compilação falha. Esse fato ocorre porque é preciso ter no caminho de classes para a compilação, as classes do J2SE juntamente às classes do J2ME. Ocorre um conflito de versões em tempo de execução e com isso a compilação falha.

A Figura 31 ilustra um processo de compilação de um pequeno aplicativo J2ME utilizando-se o compilador JML. Pode ser observado que a compilação falha porque o compilador JML não encontra o atributo `TYPE` na classe `java.lang.Boolean` do J2ME. Nesse momento o compilador JML procurava a classe `Boolean` do J2SE.



```

C:\> Prompt de comando
parsing ..\specs\java\util\SortedMap.spec
parsing ..\specs\java\util\SortedSet.spec
parsing ..\..\Documents and Settings\Ronnico\Configurações locais\Temp\jmlrac318
24.java
typechecking CalculadoraJ2ME.java
File "CalculadoraJ2ME.java", line 728, character 83 error: Variable "java.lang.B
oolean.TYPE" is not defined in current context
File "CalculadoraJ2ME.java", line 730, character 117 error: Variable "rac$types"
 may not have been initialized before use [JLS 14.3]
File "CalculadoraJ2ME.java", line 759, character 85 error: Variable "java.lang.B
oolean.TYPE" is not defined in current context
File "CalculadoraJ2ME.java", line 761, character 102 error: Variable "rac$types"
 may not have been initialized before use [JLS 14.3]
File "CalculadoraJ2ME.java", line 786, character 85 error: Variable "java.lang.B
oolean.TYPE" is not defined in current context
File "CalculadoraJ2ME.java", line 788, character 103 error: Variable "rac$types"
 may not have been initialized before use [JLS 14.3]
File "CalculadoraJ2ME.java", line 901, character 159 error: Variable "java.lang.
Boolean.TYPE" is not defined in current context
File "CalculadoraJ2ME.java", line 907, character 184 error: Variable "java.lang.
Boolean.TYPE" is not defined in current context
C:\JML\bin>

```

Figura 31. Falha na compilação de aplicativo J2ME com `jmlc`.

Uma solução para resolver o processo de compilação acima, embora não seja uma solução mais adequada, é a definição do atributo `TYPE` na classe `Boolean` do J2ME seguida da compilação da classe após a alteração.

A solução de reimplementar a classe `Boolean`, por exemplo, resolveria o problema da compilação, mas um outro problema subsequente aconteceria. O fato é que quando é realizada a compilação de qualquer aplicação Java utilizando o compilador JML, os *bytecodes* gerados pela compilação apresentam estruturas, adicionadas durante a compilação, pelo compilador JML. Se for utilizado um decompilador Java para fazer uma engenharia reversa e gerar o código-fonte a partir dos *bytecodes* podemos perceber que o código-fonte foi totalmente modificado.

A modificação realizada também adiciona referências às classes do J2SE que não são implementadas na plataforma J2ME. Esse fato faz com que na fase seguinte de pré-verificação da aplicação, esta não aconteça por causa da presença de algumas classes adicionadas durante a fase anterior, o de compilação.

Outro ponto que pôde ser observado é que após a compilação com o `jmlc`, o tamanho das aplicações cresce demasiadamente. Como as aplicações J2ME precisam ter um tamanho limitado, é preciso realizar um processo de compilação simples que não aumente bastante o tamanho do código.

Portanto, como trabalho futuro, além do estudo e extensão do compilador JML para suportar J2ME, é preciso avaliar a possibilidade de implementar no compilador, alguma técnica de otimização de compiladores que reduza de maneira significativa os *bytecodes* gerados. O projeto de um compilador JML com suporte a J2ME seria uma grande contribuição para os desenvolvedores de aplicativos móveis que precisam realizar verificações de comportamento em seus aplicativos.

5.5.3 Catálogo de componentes J2ME

Um outro trabalho futuro que surgiu durante o desenvolvimento desse foi a idéia da criação de componentes de software J2ME anotados com a linguagem JML. Os componentes de software reutilizáveis possuem uma importância bastante relevante na Engenharia de Software porque através deles podemos nos beneficiar de suas funcionalidades sem ter que implementá-las novamente, devido ao reuso delas.

Para utilização de componentes de software em um processo de desenvolvimento, não é preciso que o desenvolvedor conheça os detalhes de implementação do componente. É suficiente para o desenvolvedor conhecer apenas como ele pode se beneficiar das funcionalidades contidas dentro do componente.

O reuso de componentes de software nos projetos, possuem um caráter extremamente econômico e seguro. A economia se dá pelo fato de não haver necessidade de uma reimplementação da funcionalidade que já se encontra pronta para ser reutilizada. A segurança pode ser justificada pelo fato de os componentes de software, antes de serem disponibilizados para reuso, terem sido largamente testados e com isso, o risco de desempenhar suas funções inadequadamente seja minimizado.

Visando o desenvolvimento rápido e garantido de aplicações J2ME, o desenvolvimento desse trabalho nos levou à idéia de desenvolver um catálogo de componentes J2ME, seguindo a mesma estratégia dos componentes desenvolvidos em circuitos digitais. Componentes eletrônicos, geralmente, são disponibilizados juntos com suas especificações, chamadas de *datasheets*, que descrevem as características dos componentes incluindo as suas condições de uso.

No catálogo de componentes, deverão ser desenvolvidos componentes de software J2ME reutilizáveis, que nos catálogos de circuitos eletrônicos seriam os *ipcores*, e que conteriam toda a implementação. As especificações comportamentais das funcionalidades do componentes representariam os *datasheets* e conteriam as condições que regem o bom uso das funcionalidades presentes nos componentes. Então, cada componente do catálogo seria disponibilizado junto com a sua especificação JML e essa seria utilizada pelos desenvolvedores para que obedecam as condições de uso do componente em troca do seu correto funcionamento.

A possibilidade de implementar componentes anotados com JML e testados para as outras plataformas Java também seria de grande relevância para a comunidade da Engenharia de Software em geral. Como possibilidades, podem ser desenvolvidos componentes de segurança

com a API JAAS [45], componentes reutilizáveis para a API JavaCard, componentes J2EE [46], componentes para aplicações desktop com J2SE, além dos vários componentes para as várias outras APIs Java ou para *frameworks opensource*.

Apêndice A

Códigos-Fonte Calculadora J2ME

Apresentaremos neste apêndice os códigos fontes de uma pequena aplicação J2ME que funciona como uma calculadora básica, ou seja, realiza as 4 operações matemáticas básicas: soma, subtração, multiplicação e divisão.

O objetivo desse apêndice é mostrar ao leitor como o compilador `jmlc` modifica o código fonte original implementado na plataforma J2ME. No código fonte da Calculadora não estão presentes qualquer tipo de anotação JML. O compilador JML foi aplicado ao código fonte J2ME, apenas para analisarmos quais as modificações acrescentadas por ele.

Essa aplicação do compilador `jmlc` ao código fonte original e implementado com a plataforma J2ME também nos permite, e aliás foi esse o motivo principal para mostrarmos essas implementações nesse capítulo, visualizar uma das limitações das ferramentas JML para a plataforma J2ME: a inclusão de referências a objetos Java que não estão presentes na plataforma J2ME.

Abaixo podemos visualizar, separadamente, as duas implementações: o código fonte original implementado na plataforma J2ME e o código-fonte obtido após a decompilação dos *bytecodes* gerados pelo compilador JML, respectivamente.

Código-Fonte original na Plataforma J2ME

```
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

public class CalculadoraJ2ME extends MIDlet implements CommandListener{
```

```

private Display display = null;
private Form form = null;
private TextField textFieldOperand1 = null;
private TextField textFieldOperador= null;
private TextField textFieldOperando2 = null;
private Command commandOperar = null;
private Command commandVoltar = null;
private Command commandSair = null;
private Alert alert = null;

public CalculadoraJ2ME () {

    display = Display.getDisplay(this);
    this.form = new Form( "Calculadora J2ME" );
    this.textFieldOperand1 = new
TextField("Operand1:", "", 3, TextField.NUMERIC);
    this.textFieldOperador = new
TextField("Operador:", "", 1, TextField.ANY);
    this.textFieldOperando2 = new
TextField("Operando2:", "", 3, TextField.NUMERIC);

    this.commandOperar = new Command("Operar", Command.SCREEN, 1);
    this.commandVoltar = new Command("Voltar", Command.SCREEN, 1);
    this.commandSair = new Command("Sair", Command.SCREEN, 1);
    this.form.addCommand(commandOperar);
    this.form.addCommand(commandSair);
    this.form.append(textFieldOperand1);
    this.form.append(textFieldOperador);
    this.form.append(textFieldOperando2);

    this.form.setCommandListener(this);

}

protected void startApp() throws MIDletStateChangeException {
    // TODO Auto-generated method stub
    this.display.setCurrent(form);

}

protected void pauseApp() {
    // TODO Auto-generated method stub

}

protected void destroyApp(boolean arg0) throws
MIDletStateChangeException {
    // TODO Auto-generated method stub

}

public void commandAction( Command c, Displayable s){

    if( c == commandOperar ){

        String operand1 = textFieldOperand1.getString();
        int valor1 = Integer.parseInt(operand1);

        char operador = textFieldOperador.getString().charAt(0);

```



```
String operando2 = textFieldOperando2.getString();
int valor2 = Integer.parseInt(operando2);

String resultado = "O Resultado é: ";
switch (operador) {
case '+':
    resultado += ( valor1 + valor2 );
    break;
case '-':
    resultado += ( valor1 - valor2 );
    break;
case '*':
    resultado += ( valor1 * valor2 );
    break;
case '/':
    if( valor2 != 0){
        resultado += ( valor1 / valor2 );
    } else{
        resultado = "Não é permitida divisão por zero";
    }
    break;

default:
    break;
}
this.alert = new
Alert("Resultado", resultado, null, AlertType.INFO);
this.alert.addCommand(commandVoltar);
this.display.setCurrent(alert);
}
if( c == commandSair ){
    try{
        destroyApp(false);
        notifyDestroyed();
    }catch( Exception ex){
        ex.printStackTrace();
    }
}
if( c == commandVoltar ){

    this.display.setCurrent(form);

}
}
}
```

Código-Fonte alterado pelo compilador JML

```
import java.io.PrintStream;
import java.lang.reflect.*;
import java.util.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
```

```

import org.jmlspecs.jmlrac.runtime.*;

public class CalculadoraJ2ME extends MIDlet
    implements CommandListener, JMLCheckable
{
    public CalculadoraJ2ME()
    {
        Block$();
        rac$dented = true;
        if(!rac$ClassInitialized || !rac$initialized)
        {
            internal$$init$();
            return;
        }
        if(!JMLChecker.isActive(1) || !rac$dented())
        {
            internal$$init$();
            return;
        }
        if(JMLChecker.isActive(3) && rac$dented())
        {
            JMLChecker.enter();
            checkInv$static("<init>@pre<File \"CalculadoraJ2ME.java\", line
26, character 15>");
            JMLChecker.exit();
        }
        if(JMLChecker.isActive(1))
        {
            JMLChecker.enter();
            checkPre$$init$$CalculadoraJ2ME();
            JMLChecker.exit();
        }
        if(JMLChecker.isActive(4) && rac$dented())
        {
            JMLChecker.enter();
            evalOldExprInHC$static();
            JMLChecker.exit();
        }
        boolean flag = true;
        boolean flag1 = true;
        try
        {
            internal$$init$();
            if(JMLChecker.isActive(2) && rac$dented())
            {
                JMLChecker.enter();
                checkPost$$init$$CalculadoraJ2ME(null);
                JMLChecker.exit();
            }
        }
        catch(JMLEntryPreconditionError jmlentrypreconditionerror)
        {
            flag = false;
            throw new JMLInternalPreconditionError(jmlentrypreconditionerror);
        }
        catch(JMLAssertionError jmlassertionerror)
        {
            flag = false;
            throw jmlassertionerror;
        }
    }
}

```

```

    }
    catch(Throwable throwable)
    {
        flag1 = false;
        try
        {
            if(JMLChecker.isActive(2) && rac$dented())
            {
                JMLChecker.enter();
                checkXPost$$init$$CalculadoraJ2ME(throwable);
                JMLChecker.exit();
            }
        }
        catch(JMLAssertionError jmlassertionerror1)
        {
            flag = false;
            throw jmlassertionerror1;
        }
    }
    finally
    {
        if(flag && flag1 && (JMLChecker.isActive(3) && rac$dented()))
        {
            JMLChecker.enter();
            checkInv$instance$CalculadoraJ2ME("<init>@post<File
\"CalculadoraJ2ME.java\", line 26, character 15>");
            JMLChecker.exit();
        }
        if(flag)
        {
            if(JMLChecker.isActive(3) && rac$dented())
            {
                JMLChecker.enter();
                checkInv$static("<init>@post<File
\"CalculadoraJ2ME.java\", line 26, character 15>");
                JMLChecker.exit();
            }
            if(JMLChecker.isActive(4) && rac$dented())
            {
                JMLChecker.enter();
                checkHC$static("<init>@post<File \"CalculadoraJ2ME.java\",
line 26, character 15>", "<init>", new Class[0]);
                JMLChecker.exit();
            }
        }
    }
}

    public void checkHC$instance$CalculadoraJ2ME(String s, boolean flag,
String s1, Class aclass[])
    {
        s1 = flag ? null : s1;
        rac$check("javax.microedition.midlet.MIDlet", this,
"checkHC$instance$MIDlet", new Class[] {
            Class.forName("java.lang.String"),
Class.forName("java.lang.String"), Class.forName("[Ljava.lang.Class;")
        }, new Object[] {
            s, s1, aclass
        });
    }
}

```

```

        rac$check("javax.microedition.lcdui.CommandListener$JmlSurrogate",
this, "checkHC$instanceS$CommandListener", new Class[] {
        Class.forName("java.lang.String"),
Class.forName("java.lang.String"), Class.forName("[Ljava.lang.Class;")
        }, new Object[] {
            s, sl, aclass
        });
    }

    public void checkHC$instanceS$CalculadoraJ2ME(String s, String sl, Class
aclass[])
    {
        rac$check("javax.microedition.midlet.MIDlet", this,
"checkHC$instanceS$MIDlet", new Class[] {
            Class.forName("java.lang.String"),
Class.forName("java.lang.String"), Class.forName("[Ljava.lang.Class;")
        }, new Object[] {
            s, sl, aclass
        });
        rac$check("javax.microedition.lcdui.CommandListener$JmlSurrogate",
this, "checkHC$instanceS$CommandListener", new Class[] {
            Class.forName("java.lang.String"),
Class.forName("java.lang.String"), Class.forName("[Ljava.lang.Class;")
        }, new Object[] {
            s, sl, aclass
        });
    }

    public void checkHC$instanceW$CalculadoraJ2ME(String s, String sl, Class
aclass[])
    {
        rac$check("javax.microedition.midlet.MIDlet", this,
"checkHC$instanceS$MIDlet", new Class[] {
            Class.forName("java.lang.String"),
Class.forName("java.lang.String"), Class.forName("[Ljava.lang.Class;")
        }, new Object[] {
            s, sl, aclass
        });
        rac$check("javax.microedition.lcdui.CommandListener$JmlSurrogate",
this, "checkHC$instanceS$CommandListener", new Class[] {
            Class.forName("java.lang.String"),
Class.forName("java.lang.String"), Class.forName("[Ljava.lang.Class;")
        }, new Object[] {
            s, sl, aclass
        });
    }

    public static void checkHC$static(String s, String sl, Class aclass[])
    {
    }

    public void checkInv$instance$CalculadoraJ2ME(String s)
    {
        HashSet hashset = new HashSet();
        boolean flag = true;
        boolean flag1 = true;
        if(flag)
        {
        .
        .
    }

```

```
.  
}  
} //fim da classe
```

Podemos perceber a partir desse exemplo acima, dois grande problemas que teriam um impacto relativamente grande quando da utilização dessa classe compilada em um ambiente J2ME.

O primeiro deles podemos visualizar, por exemplo, na página 65 no método de nome `checkInv$instance$CalculadoraJ2ME`. Dentro do corpo desse método encontramos a criação de um objeto `java.util.HashSet`, o qual não está presente na plataforma J2ME.

Podemos perceber, claramente, que o tamanho do código-fonte decompilado a partir do *bytecode* gerado pelo compilador JML é bem maior que o original e dessa forma, não é viável para ser implantado em um dispositivo com limitação de memória como, por exemplo, um telefone celular.

Não entraremos em detalhes sobre essas limitações, pois o Capítulo 5 desse trabalho abordará todas elas.

Apêndice B

Classes de Testes geradas pelo *jmlunit*

O objetivo desse apêndice é ilustrar as classes de teste de unidade geradas pela ferramenta *jmlunit*. Essas classes foram geradas a partir da implementação da classe `TesteSoma.java`.

Apresentaremos, primeiramente, a implementação da classe `TesteSoma.java`. Em seguida, apresentaremos as implementações das classes geradas com o objetivo de realizar testes de unidade com o JUnit.

Classe `TesteSoma.java`

```
public class TesteSoma {  
  
    /*@ public normal_behavior  
       @ requires ( a > 0 && b > 0 );  
       @ assignable \nothing;  
       @ ensures \result == ( a + b );  
    @*/  
    public int soma ( int a , int b ) {  
        return a + b;  
    }  
}
```

Classe `TesteSoma_JML_Test.java`

```
// This file was generated by jmlunit on Sun Mar 20 16:04:57 BRT 2005.  
  
/** Automatically-generated test driver for JML and JUnit based  
 * testing of TesteSoma. The superclass of this class should be edited  
 * to supply test data. However it's best not to edit this class  
 * directly; instead use the command
```

```

* <pre>
*  jmlunit TesteSoma.java
* </pre>
* to regenerate this class whenever TesteSoma.java changes.
*/
public class TesteSoma_JML_Test
    extends TesteSoma_JML_TestData
{
    /** Initialize this class. */
    public TesteSoma_JML_Test(java.lang.String name) {
        super(name);
    }

    /** Run the tests. */
    public static void main(java.lang.String[] args) {
        org.jmlspecs.jmlunit.JMLTestRunner.run(suite());
        // You can also use a JUnit test runner such as:
        // junit.textui.TestRunner.run(suite());
    }

    /** Test to see if the code for class TesteSoma
     * has been compiled with runtime assertion checking (i.e., by jmlc).
     * Code that is not compiled with jmlc would not make an effective test,
     * since no assertion checking would be done. */
    public void test$IsRACCompiled() {
        junit.framework.Assert.assertTrue("code for class TesteSoma"
            + " was not compiled with jmlc"
            + " so no assertions will be checked!",
org.jmlspecs.jmlrac.runtime.JMLChecker.isRACCompiled(TesteSoma.class)
        );
    }

    /** Return the test suite for this test class. This will have
     * added to it at least test$IsRACCompiled(), and any test methods
     * written explicitly by the user in the superclass. It will also
     * have added test suites for each testing each method and
     * constructor.
     */
    //@ ensures \result != null;
    public static junit.framework.Test suite() {
        TesteSoma_JML_Test testobj
            = new TesteSoma_JML_Test("TesteSoma_JML_Test");
        junit.framework.TestSuite testsuite = testobj.overallTestSuite();
        // Add instances of Test found by the reflection mechanism.
        testsuite.addTestSuite(TesteSoma_JML_Test.class);
        testobj.addTestSuiteForEachMethod(testsuite);
        return testsuite;
    }

    /** A JUnit test object that can run a single test method. This
     * is defined as a nested class solely for convenience; it can't
     * be defined once and for all because it must subclass its
     * enclosing class.
     */
    protected static abstract class OneTest extends TesteSoma_JML_Test {

        /** Initialize this test object. */
        public OneTest(String name) {
            super(name);
        }
    }
}

```

```

}

/** The result object that holds information about testing. */
protected junit.framework.TestResult result;

/**@ also
/**@ requires result != null;
public void run(junit.framework.TestResult result) {
    this.result = result;
    super.run(result);
}

/* Run a single test and decide whether the test was
 * successful, meaningless, or a failure. This is the
 * Template Method pattern abstraction of the inner loop in a
 * JML/JUnit test. */
public void runTest() throws java.lang.Throwable {
    try {
        // The call being tested!
        doCall();
    }
    catch (org.jmlspecs.jmlrac.runtime.JMLEntryPreconditionError e) {
        // meaningless test input
        addMeaningless();
    } catch (org.jmlspecs.jmlrac.runtime.JMLAssertionError e) {
        // test failure
        int l = org.jmlspecs.jmlrac.runtime.JMLChecker.getLevel();
        org.jmlspecs.jmlrac.runtime.JMLChecker.setLevel
            (org.jmlspecs.jmlrac.runtime.JMLOption.NONE);
        try {
            java.lang.String failmsg = this.failMessage(e);
            junit.framework.AssertionFailedError err
                = new junit.framework.AssertionFailedError(failmsg);
            err.setStackTrace(new java.lang.StackTraceElement[]{});
            err.initCause(e);
            result.addFailure(this, err);
        } finally {
            org.jmlspecs.jmlrac.runtime.JMLChecker.setLevel(l);
        }
    } catch (java.lang.Throwable e) {
        // test success
    }
}

/** Call the method to be tested with the appropriate arguments. */
protected abstract void doCall() throws java.lang.Throwable;

/** Format the error message for a test failure, based on the
 * method's arguments. */
protected abstract java.lang.String failMessage
    (org.jmlspecs.jmlrac.runtime.JMLAssertionError e);

/** Inform listeners that a meaningless test was run. */
private void addMeaningless() {
    if (result instanceof org.jmlspecs.jmlunit.JMLTestResult) {
        ((org.jmlspecs.jmlunit.JMLTestResult) result)
            .addMeaningless(this);
    }
}
}

```



```

/** Create the tests that are to be run for testing the class
 * TesteSoma. The framework will then run them.
 * @param overallTestSuite$ The suite accumulating all of the tests
 * for this driver class.
 */
//@ requires overallTestSuite$ != null;
public void addTestSuiteForEachMethod
    (junit.framework.TestSuite overallTestSuite$)
{
    try {
        this.addTestSuiteFor$TestSoma(overallTestSuite$);
    } catch (java.lang.Throwable ex) {
        overallTestSuite$.addTest
            (new org.jmlspecs.jmlunit.strategies.ConstructorFailed(ex));
    }
}

/** Add tests for the soma method
 * to the overall test suite. */
private void addTestSuiteFor$TestSoma
    (junit.framework.TestSuite overallTestSuite$)
{
    junit.framework.TestSuite methodTests$
        = this.emptyTestSuiteFor("soma");
    try {
        org.jmlspecs.jmlunit.strategies.IndefiniteIterator
            receivers$iter
            = new org.jmlspecs.jmlunit.strategies.NonNullIteratorDecorator
                (this.vTesteSomaIter("soma", 2));
        this.check_has_data
            (receivers$iter,
             "new NonNullIteratorDecorator(this.vTesteSomaIter(\"soma\",
2))");
        while (!receivers$iter.atEnd()) {
            org.jmlspecs.jmlunit.strategies.IntIterator
                vint$1$iter
                = this.vintIter("soma", 1);
            this.check_has_data
                (vint$1$iter,
                 "this.vintIter(\"soma\", 1)");
            while (!vint$1$iter.atEnd()) {
                org.jmlspecs.jmlunit.strategies.IntIterator
                    vint$2$iter
                    = this.vintIter("soma", 0);
                this.check_has_data
                    (vint$2$iter,
                     "this.vintIter(\"soma\", 0)");
                while (!vint$2$iter.atEnd()) {
                    final TesteSoma receiver$
                        = (TesteSoma) receivers$iter.get();
                    final int a
                        = vint$1$iter.getInt();
                    final int b
                        = vint$2$iter.getInt();
                    methodTests$.addTest
                        (new TestSoma(receiver$, a, b));
                    vint$2$iter.advance();
                }
                vint$1$iter.advance();
            }
        }
    }
}

```

```

        }
        receivers$iter.advance();
    }
} catch (org.jmlspecs.jmlunit.strategies.TestSuiteFullException e$) {
    // methodTests$ doesn't want more tests
}
overallTestSuite$.addTest(methodTests$);
}

/** Test for the soma method. */
protected static class TestSoma extends OneTest {
    /** The receiver */
    private TesteSoma receiver$;
    /** Argument a */
    private int a;
    /** Argument b */
    private int b;

    /** Initialize this instance. */
    public TestSoma(TesteSoma receiver$, int a, int b) {
        super("soma" + ":" + a + ", " + b);
        this.receiver$ = receiver$;
        this.a = a;
        this.b = b;
    }

    protected void doCall() throws java.lang.Throwable {
        receiver$.soma(a, b);
    }

    protected java.lang.String failMessage
        (org.jmlspecs.jmlrac.runtime.JMLAssertionError e$)
    {
        java.lang.String msg = "\n\tMethod 'soma' applied to";
        msg += "\n\tReceiver: " + this.receiver$;
        msg += "\n\tArgument a: " + this.a;
        msg += "\n\tArgument b: " + this.b;
        return msg;
    }
}

/** Check that the iterator is non-null and not empty. */
private void
check_has_data(org.jmlspecs.jmlunit.strategies.IndefiniteIterator iter,
                String call)
{
    if (iter == null) {
        junit.framework.Assert.fail(call + " returned null");
    }
    if (iter.atEnd()) {
        junit.framework.Assert.fail(call + " returned an empty iterator");
    }
}

/** Converts a char to a printable String for display */
public static String charToString(char c) {
    if (c == '\n') {
        return "NL";
    } else if (c == '\r') {
        return "CR";
    }
}

```

```

    } else if (c == '\t') {
        return "TAB";
    } else if (Character.isISOControl(c)) {
        int i = (int)c;
        return "\\u"
            + Character.forDigit((i/2048)%16,16)
            + Character.forDigit((i/256)%16,16)
            + Character.forDigit((i/16)%16,16)
            + Character.forDigit((i)%16,16);
    }
    return Character.toString(c);
}
}
}

```

A classe `TesteSoma_JML_Test.java` é a classe que será executada pela ferramenta JML, `jml-junit`. Essa ferramenta possui, internamente, uma chamada para o *framework* JUnit o qual é o responsável por executar um teste unitário.

A próxima classe, `TesteSoma_JML_TestData.java`, representa um caso de teste do JUnit e tem como objetivo, encapsular os dados utilizados para realização dos testes.

Classe `TesteSoma_JML_TestData.java`

```

/** Supply test data for the JML and JUnit based testing of
 * TesteSoma.
 *
 * <p>This class is also the place to override the <code>setUp()</code>
 * and <code>tearDown()</code> methods if your testing needs some
 * actions to be taken before and after each test is executed.
 *
 * <p>This class is never rewritten by jmlunit.
 */
public abstract class TesteSoma_JML_TestData
    extends junit.framework.TestCase
{
    /** Initialize this class. */
    public TesteSoma_JML_TestData(java.lang.String name) {
        super(name);
    }

    /** Return the overall test suite for accumulating tests; the
     * result will hold every test that will be run. This factory
     * method can be altered to provide filtering of test suites, as
     * they are added to this overall test suite, based on various
     * criteria. The test driver will first call the method
     * addTestSuite to add a test suite formed from custom programmed
     * test methods (named testX for some X), which you can add to
     * this class; this initial test suite will also include a method
     * to check that the code being tested was compiled with jmlc.
     * After that, for each method to be tested, a test suite
     * containing tests for that method will be added to this overall
     * test suite, using the addTest method. Test suites added for a
     * method will have some subtype of TestSuite and that method's
     * name as their name. So, if you want to control the overall
     * suite of tests for testing some method, e.g., to limit the
     * number of tests for each method, return a special-purpose
     * subclass of {@link junit.framework.TestSuite} in which you override the

```

```

* addTest method.
* @see junit.framework.TestSuite
*/
//@ assignable objectState;
//@ ensures \result != null;
public junit.framework.TestSuite overallTestSuite() {
    return new junit.framework.TestSuite("Overall tests for TesteSoma");
}

/** Return an empty test suite for accumulating tests for the
 * named method. This factory method can be altered to provide
 * filtering or limiting of the tests for the named method, as
 * they are added to the test suite for this method. The driver
 * will add individual tests using the addTest method. So, if you
 * want to filter individual tests, return a subclass of TestSuite
 * in which you override the addTest method.
 * @param methodName The method the tests in this suite are for.
 * @see junit.framework.TestSuite
 * @see org.jmlspecs.jmlunit.strategies.LimitedTestSuite
 */
//@ assignable objectState;
//@ ensures \result != null;
public junit.framework.TestSuite emptyTestSuiteFor
    (java.lang.String methodName)
{
    return new junit.framework.TestSuite(methodName);
}

// TEST DATA SUPPLY SECTION

// You should edit the following code to supply test data. In the
// skeleton originally supplied below, the jmlunit tool made a
// guess as to a minimal strategy for generating test data for
// each type of object used as a receiver, and each type used as
// an argument. There is a library of strategies for generating
// test data in org.jmlspecs.jmlunit.strategies, which are used in
// the tool's guesses. See the documentation for JML and in
// particular for the org.jmlspecs.jmlunit.strategies package for
// a general discussion of how to do this. (This package's
// documentation is available through the JML.html file in the top
// of the JML release, and also in the package.html file that
// ships with the package.)
//
// In the code below, you can change the strategies from those
// that were guessed by the jmlunit tool, and you can also define
// new ones to suit your needs. You can also delete any useless
// sample test data that has been generated for you to show you
// the pattern of how to add your own test data. The only
// requirement is that you implement the methods below.
//
// If you change the type being tested in a way that introduces
// new types of arguments for some methods, then you will have to
// introduce (by hand) definitions that are similar to the ones
// below, because jmlunit never rewrites this file.

/** Return a new, freshly allocated indefinite iterator that
 * produces test data of type
 * TesteSoma
 * for testing the method named by the String methodName in
 * a loop that encloses loopsThisSurrounds many other loops.

```

```

* @param methodName name of the method for which this
*                   test data will be used.
* @param loopsThisSurrounds number of loops that the test
*                   contains inside this one.
*/
//@ requires methodName != null && loopsThisSurrounds >= 0;
//@ ensures \fresh(\result);
protected org.jmlspecs.jmlunit.strategies.IndefiniteIterator
    vTesteSomaIter
    (java.lang.String methodName, int loopsThisSurrounds)
{
    return vTesteSomaStrategy.iterator();
}

/** The strategy for generating test data of type
 * TesteSoma. */
private org.jmlspecs.jmlunit.strategies.StrategyType
    vTesteSomaStrategy
    = new org.jmlspecs.jmlunit.strategies.NewObjectAbstractStrategy()
    {
        protected Object make(int n) {
            switch (n) {
                // replace this comment with test data if desired
                default:
                    break;
            }
            throw new java.util.NoSuchElementException();
        }
    };

/** Return a new, freshly allocated indefinite iterator that
 * produces test data of type
 * int
 * for testing the method named by the String methodName in
 * a loop that encloses loopsThisSurrounds many other loops.
 * @param methodName name of the method for which this
 *                   test data will be used.
 * @param loopsThisSurrounds number of loops that the test
 *                   contains inside this one.
*/
//@ requires methodName != null && loopsThisSurrounds >= 0;
//@ ensures \fresh(\result);
protected org.jmlspecs.jmlunit.strategies.IntIterator
    vIntIter
    (java.lang.String methodName, int loopsThisSurrounds)
{
    return vIntStrategy.intIterator();
}

/** The strategy for generating test data of type
 * int. */
private org.jmlspecs.jmlunit.strategies.IntStrategyType
    vIntStrategy
    = new org.jmlspecs.jmlunit.strategies.IntStrategy()
    {
        protected int[] addData() {
            return new int[] {
                };
        }
    };

```

} ;

Apêndice C

A Especificação Comportamental de um subconjunto da plataforma J2ME

Esse apêndice tem o objetivo de tornar disponível a nossa especificação comportamental desenvolvida utilizando a linguagem JML. Estarão contidos, nesse Apêndice, as especificações para a maioria das classes do MIDP 2.0.

A ordem de apresentação das especificações, segue a mesma seqüência mostrada na Tabela 1.

Classes do pacote `javax.microedition.lcdui`

Especificação `Alert.jml`

```
package javax.microedition.lcdui;

import java.util.Timer;
import java.util.TimerTask;

import com.sun.midp.lcdui.Resource;
import com.sun.midp.lcdui.Text;

/*@ model import org.jmlspecs.models.*;

public /*@ pure @*/ class Alert extends Screen {

    public final static int FOREVER = -2;

    public final static Command DISMISS_COMMAND =
        new Command("", Command.OK, 0);
```



```
/*@ behaviour
  @ requires title instanceof String;
  @ assignable \nothing;
  @ ensures \result instanceof Alert;
  @*/
public Alert(String title);

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result instanceof Alert;
  @*/
public Alert(String title, String alertText,
             Image alertImage, AlertType alertType);

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result > 0) || (\result == Alert.FOREVER);
  @*/
public int getDefaultTimeout();

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result > 0) || (\result == Alert.FOREVER);
  @*/
public int getTimeout();

/*@ behaviour
  @ requires (time > 0) || (time = Alert.FOREVER);
  @ signals ( IllegalArgumentException e)
  @   e.getMessage() != null && ( time < 0 || time != Alert.FOREVER );
  @*/
public void setTimeout(int time);

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result instanceof AlertType) || (\result == null);
  @*/
public AlertType getType();

/*@ behaviour
  @ requires (type instanceof AlertType) || ( type == null);
  @*/
public void setType(AlertType type);

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result instanceof String) || ( \result == null )
  @*/
public String getString();
```

```
/*@ behaviour
  @ requires (str instanceof String) || ( str == null);
  @*/
public void setString(String str);

/*@ behaviour
  @ assignable \nothing;
  @ ensures ( \result instanceof Image ) || ( \result == null );
  @*/
public Image getImage();

/*@ behaviour
  @ requires ( img instanceof Image ) || ( img == null );
  @*/
public void setImage(Image img);

/*@ behaviour
  @ requires (indicator instanceof Gauge) || ( indicator == null );
  @ signals ( IllegalArgumentException e)
  @ e.getMessage() != null && (!isConformantIndicator(indicator));
  @*/
public void setIndicator(Gauge indicator);

/*@ behaviour
  @ assignable \nothing;
  @ ensures ( \result instanceof Gauge )||( \result == null );
  @*/
public Gauge getIndicator();

/*@ behaviour
  @ requires (cmd instanceof Command ) && (cmd != null);
  @ signals ( NullPointerException e)
  @ e.getMessage() != null && ( cmd == null );
  @*/
public void addCommand(Command cmd);

/*@ behaviour
  @ requires (cmd instanceof Command ) && (cmd != null);
  @ assignable \nothing;
  @*/
public void removeCommand(Command cmd);

/*@ behaviour
  @ requires (l instanceof CommandListener ) || ( l == null);
  @*/
public void setCommandListener(CommandListener l);

// *****
// Package private members
// *****
```

```
// *****
// Private members
// *****

private static final Command OK =
    new Command(Resource.getString("Done"), Command.OK, 0);

private static final int CELL_SPACING = 6;

private static final int SCROLL_AMOUNT = 40;

private static final int DEFAULT_TIMEOUT = 2000;

private AlertType type;

private String text;

private Image image;

private Image mutableImage;

private Gauge indicator;

private int time;

private int height;

private static Timer timeoutTimer;

private TimerTask timerTask;

private Displayable returnScreen;

private CommandListener userCommandListener;

}
```

Especificação `AlertType.jml`

```
package javax.microedition.lcdui;
/*@ model import org.jmlspecs.models.*;

public /*@ pure @*/ class AlertType {

    static final int ALERT_INFO = 1;

    static final int ALERT_WARN = 2;

    static final int ALERT_ERR = 3;

    static final int ALERT_ALRM = 4;

    static final int ALERT_CFM = 5;

    public static final AlertType INFO = new AlertType(ALERT_INFO);
```

```

public static final AlertType WARNING = new AlertType(ALERT_WARN);

public static final AlertType ERROR = new AlertType(ALERT_ERR);

public static final AlertType ALARM = new AlertType(ALERT_ALRM);

public static final AlertType CONFIRMATION = new AlertType(ALERT_CFM);

protected AlertType();

/*@ behaviour
  @ requires (display instanceof Display) && ( display != null );
  @ assignable \nothing;
  @ ensures \result instanceof Boolean;
  @ signals ( NullPointerException e )
  @   e.getMessage() != null && ( display == null );
  @*/
public boolean playSound(Display display);

private int type;

/*@ behaviour
  @ requires (type > 0 ) &&
  @   ( type == AlertType.ALERT_INFO || type == AlertType.ALERT_WARN
  @     || type == AlertType.ALERT_ERR || type == AlertType.ALERT_ALRM
  @     || type == AlertType.ALERT_CFM);
  @*/
AlertType(int type);

/*@ behaviour
  @ ensures (\result > 0 ) &&
  @   ( \result == AlertType.ALERT_INFO || \result == AlertType.ALERT_WARN
  @     || \result == AlertType.ALERT_ERR ||
  @     \result == AlertType.ALERT_ALRM
  @     || \result == AlertType.ALERT_CFM);
  @*/
int getType();
}

```

Especificação Choice .jml

```

package javax.microedition.lcdui;
/*@ model import org.jmlspecs.models.*;

public /*@ pure @*/ interface Choice {

    public static final int EXCLUSIVE = 1;

    public static final int MULTIPLE = 2;

    public static final int IMPLICIT = 3;

    public static final int POPUP = 4;

```

```

public static final int TEXT_WRAP_DEFAULT = 0;

public static final int TEXT_WRAP_ON = 1;

public static final int TEXT_WRAP_OFF = 2;

/*@ behaviour
  @ ensures \result >= 0;
  @*/
public int size();

/*@ behaviour
  @ requires (elementNum >= 0) && ( elementNum <= size() - 1);
  @ ensures (\result instanceof String );
  @ signals ( IndexOutOfBoundsException e )
  @   e.getMessage() != null && (elementNum < 0  || elementNum >= size());
  @*/
public String getString(int elementNum);

/*@ behaviour
  @ requires (elementNum >= 0) && ( elementNum <= size() - 1);
  @ ensures (\result instanceof Image );
  @ signals ( IndexOutOfBoundsException e )
  @   e.getMessage() != null && (elementNum < 0  || elementNum >= size());
  @*/
public Image getImage(int elementNum);

/*@ behaviour
  @ requires (stringPart != null) && (stringPart instanceof String);
  @ ensures (\result >= 0);
  @ signals ( NullPointerException e )
  @   e.getMessage() != null && ( stringPart == null);
  @*/
public int append(String stringPart, Image imagePart);

/*@ behaviour
  @ requires (elementNum >= 0) && ( elementNum <= size() - 1) &&
    (stringPart != null);
  @ signals ( IndexOutOfBoundsException e )
  @   e.getMessage() != null && (elementNum < 0  || elementNum >= size());
  @ signals ( NullPointerException e )
  @   e.getMessage() != null && ( stringPart == null );
  @*/
public void insert(int elementNum, String stringPart, Image imagePart);

/*@ behaviour
  @ requires (elementNum >= 0) && ( elementNum <= size() - 1);
  @ signals ( IndexOutOfBoundsException e )
  @   e.getMessage() != null && (elementNum < 0  || elementNum >= size());
  @*/
public void delete(int elementNum);

public void deleteAll();

/*@ behaviour
  @ requires (elementNum >= 0) && ( elementNum <= size() - 1) &&

```

```

    @ ( stringPart != null );
    @ signals ( IndexOutOfBoundsException e )
    @ e.getMessage() != null && ( elementNum < 0 || elementNum >= size() );
    @ signals ( NullPointerException e )
    @ e.getMessage() != null && ( stringPart == null );
    @*/
public void set(int elementNum, String stringPart, Image imagePart);

/*@ behaviour
    @ requires ( elementNum >= 0 ) && ( elementNum <= size() - 1 );
    @ ensures ( \result == true || \result == false );
    @ signals ( IndexOutOfBoundsException e )
    @ e.getMessage() != null && ( elementNum < 0 || elementNum >= size() );
    @*/
public boolean isSelected(int elementNum);

/*@ behaviour
    @ ensures ( \result >= 0 || ( \result == -1 ) );
    @*/
public int getSelectedIndex();

/*@ behaviour
    @ requires ( selectedArray_return.length >= size() );
    @ ensures ( \result >= 0 );
    @ signals ( IndexOutOfBoundsException e )
    @ e.getMessage() != null && ( elementNum < 0 || elementNum >= size() );
    @ signals ( NullPointerException e )
    @ e.getMessage() != null && ( selectedArray_return == null );
    @*/
public int getSelectedFlags(boolean[] selectedArray_return);

/*@ behaviour
    @ requires ( elementNum >= 0 ) && ( elementNum <= size() - 1 ) &&
    @ ( selected == true || selected == false );
    @ signals ( IndexOutOfBoundsException e )
    @ e.getMessage() != null && ( elementNum < 0 || elementNum >= size() );
    @*/
public void setSelectedIndex(int elementNum, boolean selected);

/*@ behaviour
    @ requires ( selectedArray.length >= 0 ) &&
    @ ( selectedArray.length <= size() - 1 );
    @ signals ( IndexOutOfBoundsException e )
    @ e.getMessage() != null && ( selectedArray.length < size() );
    @ signals ( NullPointerException e )
    @ e.getMessage() != null && ( selectedArray == null ) /
    @*/
public void setSelectedFlags(boolean[] selectedArray);

/*@ behaviour
    @ requires ( fitPolicy == Alert.TEXT_WRAP_DEFAULT ) ||
    @ ( fitPolicy == Alert.TEXT_WRAP_ON ) ||
    @ ( fitPolicy == Alert.TEXT_WRAP_OFF );
    @ signals ( IllegalArgumentException e )
    @ e.getMessage() != null &&
    @ (( fitPolicy != Alert.TEXT_WRAP_DEFAULT ) &&

```

```

        @ ( fitPolicy != Alert.TEXT_WRAP_ON ) &&
        @ ( fitPolicy != Alert.TEXT_WRAP_OFF );
    @*/
public void setFitPolicy(int fitPolicy);

    /*@ behaviour
    @ ensures ( \result == Alert.TEXT_WRAP_DEFAULT ) ||
    @ ( \result == Alert.TEXT_WRAP_ON ) ||
    @ ( \result == Alert.TEXT_WRAP_OFF );
    @*/
public int getFitPolicy();

    /*@ behaviour
    @ requires (elementNum >= 0) && ( elementNum <= size() - 1) &&
    @ (( font instanceof Font ) || ( font == null));
    @ signals ( IndexOutOfBoundsException e )
    @ e.getMessage() != null &&
    @ (elementNum < 0 || elementNum >= size());
    @*/

public void setFont(int elementNum, Font font);

    /*@ behaviour
    @ requires (elementNum >= 0) && ( elementNum <= size() - 1);
    @ ensures (\result instanceof Font);
    @ signals ( IndexOutOfBoundsException e )
    @ e.getMessage() != null &&
    @ (elementNum < 0 || elementNum >= size());
    @*/
public Font getFont(int elementNum);
}

```

Especificação ChoiceGroup.jml

```

package javax.microedition.lcdui;

/*@ model import org.jmlspecs.models.*;

import com.sun.midp.lcd.ui.Text;
import com.sun.midp.lcd.ui.*;

public /*@ pure @*/ class ChoiceGroup extends Item implements Choice {

    boolean isList;

    private int choiceType;

    private int fitPolicy;

    private int numOfEls;

    private int selectedIndex = -1;

    private int highlightedIndex = -1;

```



```

private boolean selEls[];

private String[] stringEls;

private Image[] imageEls;

private Image[] mutableImageEls;

private Font[] fontEls;

private int[] elHeights;

private int cachedWidth;

private static final int DEFAULT_WIDTH = 80;

private boolean popUpOpen;

private boolean traversedIn;

private DisplayManager displayManager;

private int maxPopupWidth = DEFAULT_WIDTH;

/*@ behaviour
  @ requires (choiceType == EXCLUSIVE)|| ( choiceType == MULTIPLE )
  @   ||( choiceType == IMPLICIT )||( choiceType == POPUP);
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && ((choiceType != EXCLUSIVE)
  @   &&( choiceType != MULTIPLE )&&( choiceType != IMPLICIT )
  @   &&( choiceType != POPUP));
  @*/
public ChoiceGroup(String label, int choiceType);

/*@ behaviour
  @ requires (stringElements != null)&&
  @   (choiceType==EXCLUSIVE||choiceType==MULTIPLE||choiceType==POPUP)
  @   &&( \forall int i; 0 <= i && i < stringElements.length;
  @ (stringElements[i] != null));
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null &&
  @   ( (imageElements != null &&
  @   (imageElements.length != stringElements.length))||
  @   ( (choiceType != EXCLUSIVE)&&( choiceType != MULTIPLE)&&
  @   (choiceType != POPUP) ));
  @ signals ( NullPointerException ex )
  @   ex.getMessage() != null && ( stringElements == null||
  @   ( \forall int i; 0 <= i && i < stringElements.length;
  @   (stringElements[i] != null)) );
  @*/
public ChoiceGroup(String label, int choiceType,
                  String[] stringElements, Image[] imageElements);

/*@ behaviour
  @ requires (stringElements != null)&&
  @   (choiceType==EXCLUSIVE||choiceType==MULTIPLE)
  @   &&( \forall int i; 0 <= i &&

```

```

    @ i < stringElements.length;
    @ (stringElements[i] != null));
    @ signals ( IllegalArgumentException ex )
    @ ex.getMessage() != null &&
    @ ( imageElements != null &&
    @ (imageElements.length != stringElements.length)
    @ ||( (choiceType != EXCLUSIVE) && ( choiceType != MULTIPLE) ));
    @ signals ( NullPointerException ex )
    @ ex.getMessage() != null && ( stringElements == null ||
    @ ( \forall int i; 0 <= i &&
    @ i < stringElements.length;
    @ (stringElements[i] != null) ) );
    @*/
ChoiceGroup(String label, int choiceType, String[] stringElements,
            Image[] imageElements, boolean implicitAllowed);

/*@ behaviour
    @ assignable \nothing;
    @ ensures \result >= 0;
    @*/
public int size();

/*@ behaviour
    @ requires elementNum >= 0 && elementNum < size();
    @ assignable \nothing;
    @ ensures (\result instanceof String);
    @ signals ( IndexOutOfBoundsException ex )
    @ ex.getMessage() != null && (elementNum < 0 && elementNum > size());
    @*/
public String getString(int elementNum);

/*@ behaviour
    @ requires elementNum >= 0 && elementNum < size();
    @ assignable \nothing;
    @ ensures (\result instanceof Image);
    @ signals ( IndexOutOfBoundsException ex )
    @ ex.getMessage() != null && (elementNum < 0 && elementNum > size());
    @*/
public Image getImage(int elementNum);

/*@ behaviour
    @ requires ( stringPart != null) && (stringPart instanceof String);
    @ ensures (\result >= 0);
    @ signals ( NullPointerException ex )
    @ ex.getMessage() != null && ( stringPart == null );
    @*/
public int append(String stringPart, Image imagePart);

/*@ behaviour
    @ requires ( elementNum < cg.size()) && ( stringPart != null);
    @ signals ( IndexOutOfBoundsException ex )
    @ ex.getMessage() != null && ( elementNum >= size() );
    @ signals ( NullPointerException ex )
    @ ex.getMessage() != null && ( stringPart == null );
    @*/
public void insert(int elementNum, String stringPart,
                 Image imagePart);

```

```

/*@ behaviour
  @ requires ( elementNum < size());
  @ signals ( IndexOutOfBoundsException ex )
  @ ex.getMessage() != null && ( elementNum >= size() );
  @*/
public void delete(int elementNum);

/*@ behaviour
  @ ensures size() == 0;
  @*/
public void deleteAll();

/*@ behaviour
  @ requires ( elementNum >= 0 ) && ( stringPart != null);
  @ signals ( IndexOutOfBoundsException ex )
  @ ex.getMessage() != null && ( elementNum < 0 );
  @ signals ( NullPointerException ex )
  @ ex.getMessage() != null && ( stringPart == null );
  @*/
public void set(int elementNum, String stringPart, Image imagePart);

/*@ behaviour
  @ requires ( elementNum < size() && elementNum > 0);
  @ ensures (\result == true) || ( \result == false );
  @ signals ( IndexOutOfBoundsException ex )
  @ ex.getMessage() != null && ( elementNum >= size() );
  @*/
public boolean isSelected(int elementNum);

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result >= -1;
  @*/
public int getSelectedIndex();

/*@ behaviour
  @ requires ( \forall int i;
  @ 0 <= i && i < selectedArray_return.length;
  @ (selectedArray_return[i] == true ||
  @ selectedArray_return[i] == false ));
  @ signals ( IllegalArgumentException ex )
  @ ex.getMessage() != null && ( selectedArray_return.length < size() );
  @ signals ( NullPointerException ex )
  @ ex.getMessage() != null && ( selectedArray_return == null );
  @*/
public int getSelectedFlags(boolean[] selectedArray_return);

/*@ behaviour
  @ requires ( elementNum < size() && elementNum >= 0 )
  @ && ( selected == true || selected == false);
  @ signals ( IndexOutOfBoundsException ex )
  @ ex.getMessage() != null && ( elementNum >= size() );
  @*/
public void setSelectedIndex(int elementNum, boolean selected);

```

```

/*@ behaviour
  @ requires ( \forall int i; 0 <= i &&
  @   i < selectedArray.length;
  @   (selectedArray[i] == true ||
  @   selectedArray[i] == false ));
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && ( selectedArray.length < size() );
  @ signals ( NullPointerException ex )
  @   ex.getMessage() != null && ( selectedArray == null );
  @*/
public void setSelectedFlags(boolean[] selectedArray);

/*@ behaviour
  @ requires ( fitPolicy == TEXT_WRAP_DEFAULT )
  @   ||( fitPolicy == TEXT_WRAP_ON) ||( fitPolicy == TEXT_WRAP_OFF);
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null &&
  @   (( fitPolicy != TEXT_WRAP_DEFAULT )
  @   &&( fitPolicy != TEXT_WRAP_ON) &&( fitPolicy != TEXT_WRAP_OFF));
  @*/
public void setFitPolicy(int fitPolicy);

/*@ behaviour
  @ assignable \nothing;
  @ ensures ( \result == TEXT_WRAP_DEFAULT )
  @   ||( \result == TEXT_WRAP_ON) ||( \result == TEXT_WRAP_OFF);
  @*/
public int getFitPolicy();

/*@ behaviour
  @ requires (font instanceof Font) &&
  @   ( elementNum >= 0 && elementNum <= size()-1 );
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && (elementNum < 0 || elementNum > size()-1);
  @*/
public void setFont(int elementNum, Font font);

/*@ behaviour
  @ requires ( elementNum >= 0 && elementNum <= size()-1 );
  @ ensures (\result instanceof Font);
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && (elementNum < 0 || elementNum > size()-1);
  @*/
public Font getFont(int elementNum);
}

```

Especificação Command. jml

```

package javax.microedition.lcdui;

/*@ model import org.jmlspecs.models.*;

import com.sun.midp.lcdui.CommandAccess;

```

```

public /*@ pure @*/ class Command {

    public static final int SCREEN = 1;

    public static final int BACK = 2;

    public static final int CANCEL = 3;

    public static final int OK = 4;

    public static final int HELP = 5;

    public static final int STOP = 6;

    public static final int EXIT = 7;

    public static final int ITEM = 8;

    // protected members //

    String      shortLabel;

    String      longLabel;

    int         commandType;

    int         priority;

    // private members //

    private int   id;

    // Constructors //

    /*@ behaviour
       @ requires ( (commandType == Command.SCREEN)||
       @   (commandType == Command.BACK)||
       @   (commandType == Command.CANCEL)||
       @   (commandType == Command.OK)||
       @   (commandType == Command.HELP)||
       @   (commandType == Command.STOP)||
       @   (commandType == Command.EXIT)||
       @   (commandType == Command.ITEM)) && ( label != null )
       @   && (priority > 0);
       @ assignable \nothing;
       @ signals ( NullPointerException e )
       @   e.getMessage() != null && ( label == null );
       @ signals ( IllegalArgumentException e)
       @   e.getMessage() != null &&
       @   ( (commandType != Command.SCREEN)&&
       @   (commandType != Command.BACK)&&
       @   (commandType != Command.CANCEL)&&
       @   (commandType != Command.OK)&&
       @   (commandType != Command.HELP)&&
       @   (commandType != Command.STOP)&&
       @   (commandType != Command.EXIT)&&
       @   (commandType != Command.ITEM));
       @*/
    public Command(String label, int commandType, int priority);

```

```

/*@ behaviour
  @ requires ( (commandType == Command.SCREEN) ||
  @   (commandType == Command.BACK) ||
  @   (commandType == Command.CANCEL) ||
  @   (commandType == Command.OK) ||
  @   (commandType == Command.HELP) ||
  @   (commandType == Command.STOP) ||
  @   (commandType == Command.EXIT) ||
  @   (commandType == Command.ITEM) ) &
  @   ( shortLabel != null ) & (priority > 0);
  @ assignable \nothing;
  @ signals ( NullPointerException e )
  @   e.getMessage() != null && ( shortLabel == null );
  @ signals ( IllegalArgumentException e )
  @   e.getMessage() != null &&
  @   ( (commandType != Command.SCREEN) &&
  @   (commandType != Command.BACK) &&
  @   (commandType != Command.CANCEL) &&
  @   (commandType != Command.OK) &&
  @   (commandType != Command.HELP) &&
  @   (commandType != Command.STOP) &&
  @   (commandType != Command.EXIT) &&
  @   (commandType != Command.ITEM) );
  @*/
public Command(String shortLabel, String longLabel, int commandType,
               int priority);

// public method implementations //

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result == shortLabel;
  @*/
public String getLabel();

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result == longLabel) || ( \result == null);
  @*/
public String getLongLabel();

/*@ behaviour
  @ assignable \nothing;
  @ ensures ( \result == Command.SCREEN) ||
  @   (\result == Command.BACK) ||
  @   (\result == Command.CANCEL) ||
  @   (\result == Command.OK) ||
  @   (\result == Command.HELP) ||
  @   (\result == Command.STOP) ||
  @   (\result == Command.EXIT) ||
  @   (\result == Command.ITEM) );
  @*/
public int getCommandType();

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result > 0;
  @*/
public int getPriority();

```

```
} // class Command
```

Especificação CommandListener.jml

```
package javax.microedition.lcdui;

/*@ model import org.jmlspecs.models.*;
public /*@ pure @*/ interface CommandListener {

    /*@ behaviour
       @ requires ( c instanceof Command ) && ( d instanceof Displayable );
       @*/
    void commandAction(Command c, Displayable d);
}

```

Especificação DateField.jml

```
package javax.microedition.lcdui;

/*@ model import org.jmlspecs.*;

import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;
import com.sun.midp.lcdui.Resource;
import com.sun.midp.lcdui.Text;

public /*@ pure @*/ class DateField extends Item {

    public static final int DATE = 1;

    public static final int TIME = 2;

    public static final int DATE_TIME = 3;

    /*@ behaviour
       @ requires (label instanceof String)&&
       @   ( mode == DATE || mode == TIME || mode == DATE_TIME);
       @ ensures \result instanceof DateField;
       @ signals( IllegalArgumentException ex)
       @   ex.getMessage() != null &&
       @   ( mode != DATE && mode != TIME && mode != DATE_TIME);
       @*/
    public DateField(String label, int mode);

    /*@ behaviour
       @ requires (label instanceof String)&&
       @   ( mode == DATE || mode == TIME || mode == DATE_TIME)&&
       @   ((timeZone instanceof java.util.TimeZone) || (timeZone == null));
       @ ensures \result instanceof DateField;
       @ signals( IllegalArgumentException ex)
       @   ex.getMessage() != null &&

```



```
    @ ( mode != DATE && mode != TIME && mode != DATE_TIME);
    */
public DateField(String label, int mode, java.util.TimeZone timeZone);

/*@ behaviour
   @ assignable \nothing;
   @ ensures \result instanceof java.util.Date;
   */
public java.util.Date getDate();

/*@ behaviour
   @ requires ( date instanceof java.util.Date);
   */
public void setDate(java.util.Date date);

/*@ behaviour
   @ assignable \nothing;
   @ ensures ( mode == DATE || mode == TIME || mode == DATE_TIME);
   */
public int getInputMode();

/*@ behaviour
   @ requires ( mode == DATE || mode == TIME || mode == DATE_TIME);
   */
public void setInputMode(int mode);

}
```

Especificação Display.jml

```
package javax.microedition.lcdui;

/*@ model import org.jmlspecs.models.*;

import java.util.*;
import java.io.*;

import javax.microedition.midlet.MIDlet;

import javax.microedition.lcdui.game.GameCanvas;
import javax.microedition.io.Connector;

import com.sun.midp.midlet.MIDletState;
import com.sun.midp.midlet.MIDletStateMap;
import com.sun.midp.midlet.MIDletSuite;
import com.sun.midp.midlet.Scheduler;

import com.sun.midp.io.j2me.storage.File;
import com.sun.midp.io.j2me.storage.RandomAccessStream;
import com.sun.midp.main.Configuration;
import com.sun.midp.lcdui.*;

import com.sun.midp.security.*;
```

```
public /*@ pure @*/ class Display {

    public static final int LIST_ELEMENT = 1;

    public static final int CHOICE_GROUP_ELEMENT = 2;

    public static final int ALERT = 3;

    public static final int COLOR_BACKGROUND = 0;

    public static final int COLOR_FOREGROUND = 1;

    public static final int COLOR_HIGHLIGHTED_BACKGROUND = 2;

    public static final int COLOR_HIGHLIGHTED_FOREGROUND = 3;

    public static final int COLOR_BORDER = 4;

    public static final int COLOR_HIGHLIGHTED_BORDER = 5;

    /*
     * ***** protected member variables
     */

    /*
     * ***** package private member variables
     */

    static final Object LCDUILock = new Object();

    static final Object calloutLock = new Object();

    static final int WIDTH;

    static final int HEIGHT;

    static final int ADORNEDHEIGHT;

    static final int ERASE_COLOR;

    static final int BORDER_COLOR = 0x00AFAFAF; // light gray

    static final int BORDER_H_COLOR = 0x00606060; // dark gray

    static final int DISPLAY_DEPTH;

    static final boolean DISPLAY_IS_COLOR;

    static final boolean POINTER_SUPPORTED;

    static final boolean MOTION_SUPPORTED;

    static final boolean REPEAT_SUPPORTED;

    static final boolean IS_DOUBLE_BUFFERED;

    static final int FG_COLOR;

    static final int BG_H_COLOR;
```

```
static final int FG_H_COLOR;

static final int ALPHA_LEVELS;

static final int KEYCODE_UP;

static final int KEYCODE_DOWN;

static final int KEYCODE_LEFT;

static final int KEYCODE_RIGHT;

static final int KEYCODE_SELECT;

/*
 * ***** private member variables
 */

private static DisplayManagerImpl displayManagerImpl;

private static DisplayDeviceAccess deviceAccess;

private static EventHandler eventHandler;

private static final Graphics screenGraphics;

private DisplayAccessor accessor;

private MIDlet midlet;

private Displayable current;

private boolean wantsForeground;

private int stickyKeyMask;

private int currentKeyMask;

private MIDletEventListener midletEventListener;

private boolean paintSuspended; // = false;

private boolean hasForeground; // = false;

private static java.util.Vector queue1 = new java.util.Vector();

private static java.util.Vector queue2 = new java.util.Vector();

private static java.util.Vector currentQueue = queue1;

private static SecurityToken classSecurityToken;

/*@ behaviour
 * @ requires (m instanceof MIDlet) || ( m == null );
 * @ assignable \nothing;
 */
Display(MIDlet m);
```

```

/*@ behaviour
  @ requires (m instanceof MIDlet) || ( m != null );
  @ assignable \nothing;
  @ ensures (\result instanceof Display );
  @ signals ( NullPointerException e)
  @ e.getMessage() != null && ( m == null );
  @*/
public static Display getDisplay(MIDlet m);

/*@ behaviour
  @ requires ( colorSpecifier == COLOR_BACKGROUND) ||
  @ ( colorSpecifier == COLOR_FOREGROUND) ||
  @ ( colorSpecifier == COLOR_HIGHLIGHTED_BACKGROUND) ||
  @ ( colorSpecifier == COLOR_HIGHLIGHTED_FOREGROUND) ||
  @ ( colorSpecifier == COLOR_BORDER) ||
  @ ( colorSpecifier == COLOR_HIGHLIGHTED_BORDER);
  @ assignable \nothing;
  @ ensures (\result >= 0);
  @ signals (IllegalArgumentException e)
  @ e.getMessage() != null &&
  @ ( colorSpecifier != COLOR_BACKGROUND) &&
  @ ( colorSpecifier != COLOR_FOREGROUND) &&
  @ ( colorSpecifier != COLOR_HIGHLIGHTED_BACKGROUND) &&
  @ ( colorSpecifier != COLOR_HIGHLIGHTED_FOREGROUND) &&
  @ ( colorSpecifier != COLOR_BORDER) &&
  @ ( colorSpecifier != COLOR_HIGHLIGHTED_BORDER);
  @*/
public int getColor(int colorSpecifier);

/*@ behaviour
  @ requires highlighted == true || highlighted == false;
  @ assignable \nothing;
  @ ensures \result >= 0;
  @*/
public int getBorderStyle(boolean highlighted);

/*@ behaviour
  @ assignable \nothing;
  @ ensures ( \result == true ) || ( \result == false );
  @*/
public boolean isColor();

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result > 0;
  @*/
public int numColors();

/*@ ensures \result >= 0;
public int numAlphaLevels();

/*@ behaviour
  @ assignable \nothing;
  @ ensures ( \result instanceof Displayable ) || ( \result == null);
  @*/
public Displayable getCurrent();

/*@ behaviour
  @ requires ( nextDisplayable instanceof Displayable ) ||

```

```

        @ ( nextDisplayable == null);
        @*/
public void setCurrent(Displayable nextDisplayable);

/*@ behaviour
  @ requires ( alert instanceof Alert && alert != null ) &&
  @   ( nextDisplayable instanceof Displayable
  @   && nextDisplayable != null);
  @ signals ( NullPointerException e )
  @   e.getMessage() != null &&
  @   ( alert == null || nextDisplayable == null );
  @ signals ( IllegalArgumentException e)
  @   e.getMessage() != null && ( nextDisplayable instanceof Alert );
  @*/
public void setCurrent(Alert alert, Displayable nextDisplayable);

/*@ behaviour
  @ requires ( item instanceof Item ) && ( item != null );
  @ signals ( NullPointerException e)
  @   e.getMessage() != null && ( item == null);
  @ signals ( IllegalStateException e)
  @   e.getMessage() != null && ( item instanceof Alert );
  @*/
public void setCurrentItem(Item item);
}

```

Especificação Displayable .jml

```

package javax.microedition.lcdui;

/*@ model import org.jmlspecs.models.*;

import java.util.TimerTask;
import java.util.Timer;

import com.sun.midp.lcdui.Text;

abstract public /*@ pure @*/ class Displayable {

    Display currentDisplay;

    Command commands[];

    int numCommands;

    CommandListener listener;

    final static int X      = 0;

    final static int Y      = 1;

    final static int WIDTH  = 2;

    final static int HEIGHT = 3;

```

```

int[] viewport;

boolean fullScreenMode;

boolean sizeChangeOccurred;

Displayable paintDelegate;

// *****
// private member variables
// *****

private final static Font TITLE_FONT =
    Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, Font.SIZE_MEDIUM);

private final static int TITLE_HEIGHT = TITLE_FONT.getHeight() + 1;

private String title;

private Ticker ticker;

private final static Timer tickerTimer;

private TickerPainter tickerPainter;

private int tickerHeight;

private int totalHeight;

private int vScrollPosition = 0;

private int vScrollProportion = 100;

Displayable();

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result == null) || (\result instanceof String);
  */
public String getTitle();

/*@ behaviour
  @ requires (s instanceof String) || ( s == null );
  */
public void setTitle(String s);

/*@ behaviour
  @ ensures (\result instanceof Ticker) || ( \result == null );
  */
public Ticker getTicker();

/*@ behaviour
  @ requires (ticker instanceof Ticker);
  */
public void setTicker(Ticker ticker);

/*@ behaviour
  @ assignable \nothing;
  @ ensures ( \result == true ) || ( \result == false );
  */

```

```
public boolean isShown();

/*@ behaviour
  @ requires ( cmd != null ) && ( cmd instanceof Command );
  @ signals ( NullPointerException e)
  @ e.getMessage() != null && ( cmd == null);
  @*/
public void addCommand(Command cmd);

/*@ behaviour
  @ requires ( cmd instanceof Command ) || ( cmd == null );
  @ assignable \nothing;
  @*/
public void removeCommand(Command cmd);

/*@ behaviour
  @ requires (l instanceof CommandListener) || ( l == null );
  @ ensures ( listener == l ) || ( listener == null);
  @*/
public void setCommandListener(CommandListener l);

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result >= 0;
  @*/
public int getWidth();

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result >= 0;
  @*/
public int getHeight();

} // Displayable
```

Especificação Form. jml

```
package javax.microedition.lcdui;

/*@ model import org.jmlspecs.models.*;

public /*@ pure @*/ class Form extends Screen {

    final boolean TRAVERSE_HORIZONTAL = true;

    final boolean TRAVERSE_VERTICAL = true;

    static final int CELL_SPACING = 4;

    static final int ONE_PIXEL_BOX = 0;

    static final int TRIANGLE_CORNERS = 1;

    static final int TRAVERSE_INDICATOR = ONE_PIXEL_BOX;
```



```

static final int TRAVERSE_INDICATOR_COLOR = Item.DARK_GRAY_COLOR;

static final int LAYOUT_HMASK = 0x03;

static final int LAYOUT_VMASK = 0x30;

// *****
// private member variables
// *****
private int formMode;

private static final int FORM_TRAVERSE = 0;

private static final int ITEM_TRAVERSE = 2;

private int traverseIndex = -1;

private boolean indicateTraverse = true;

private boolean validateVisibility = true;

private int[] viewable;

private int[] visRect;

private static final int GROW_SIZE = 4;

private Item items[];

private int numOfItems; // = 0;

private ItemStateListener itemStateListener;

private boolean pointerPressed;

/*@ public normal_behaviour
   @ assignable \nothing;
   @*/
public Form(String title);

/*@ behaviour
   @ requires (\forall int i; 0 < i && i < items.length;
   @   items[i] instanceof Item);
   @ assignable \nothing;
   @ signals ( NullPointerException e)
   @   e.getMessage() != null &&
   @   (\forall int j; 0 < j &&
   @   j < items.length; items[j] == null );
   @*/
public Form(String title, Item[] items);

/*@ behaviour
   @ requires item != null;
   @ assignable \nothing;
   @ signals ( NullPointerException e)
   @   e.getMessage() != null && item == null;
   @ ensures \result >= 0;
   @*/

```

```

public int append(Item item);

/*@ behaviour
  @ requires str != null;
  @ assignable \nothing;
  @ signals ( NullPointerException e)
  @   e.getMessage() != null && str == null;
  @ ensures \result >= 0;
  @*/
public int append(String str);

/*@ behaviour
  @ requires img != null;
  @ assignable \nothing;
  @ signals ( NullPointerException e)
  @   e.getMessage() != null && img == null;
  @ ensures \result >= 0;
  @*/
public int append(Image img);

/*@ behaviour
  @ requires item != null && itemNum >= 0;
  @ assignable \nothing;
  @ signals ( NullPointerException e)
  @   e.getMessage() != null && item == null;
  @ signals ( IndexOutOfBoundsException e)
  @   e.getMessage() != null && ( itemNum < 0 || itemNum > items.length );
  @*/
public void insert(int itemNum, Item item);

/*@ behaviour
  @ requires item != null && itemNum >= 0;
  @ assignable \nothing;
  @ signals ( NullPointerException e)
  @   e.getMessage() != null && item == null;
  @ signals ( IndexOutOfBoundsException e)
  @   e.getMessage() != null && ( itemNum < 0 || itemNum > items.length );
  @*/
public void delete(int itemNum);

/*@ behaviour
  @ requires items.length != 0;
  @ assignable \nothing;
  @ ensures items.length == 0;
  @*/
public void deleteAll();

/*@ behaviour
  @ requires (item != null && item instanceof Item )&& itemNum >= 0;
  @ assignable \nothing;
  @ ensures items[itemNum] = item;
  @ signals ( NullPointerException e)
  @   e.getMessage() != null && item == null;
  @ signals ( IndexOutOfBoundsException e)
  @   e.getMessage() != null && ( itemNum < 0 || itemNum > items.length );
  @*/
public void set(int itemNum, Item item);

```

```

/*@ behaviour
  @ requires itemNum >= 0 && itemNum < items.length;
  @ assignable \nothing;
  @ ensures \result = items[itemNum] && (\result instanceof Item);
  @ signals ( IndexOutOfBoundsException e)
  @ e.getMessage() != null && ( itemNum < 0 || itemNum > items.length );
  @*/
public Item get(int itemNum);

/*@ behaviour
  @ requires (iListener instanceof ItemStateListener) ||
  @ iListener == null;
  @ assignable \nothing;
  @ ensures ( if( iListener == null) \old( iListener ) == null );
  @*/
public void setItemStateListener(ItemStateListener iListener);

/*@ behaviour
  @ requires items != null;
  @ assignable \nothing;
  @ ensures \result > 0 && (\result == items.length);
  @*/
public int size();

/*@ behaviour
  @ ensures \result >= 0;
  @*/
public int getWidth();

/*@ behaviour
  @ ensures \result >= 0;
  @*/
public int getHeight();
}

```

Especificação Item. jml

```

package javax.microedition.lcdui;

/*@ model import org.jmlspecs.models.*;

import com.sun.midp.lcd.ui.Text;

abstract public /*@ pure @*/ class Item {

    public final static int LAYOUT_DEFAULT = 0;

    public final static int LAYOUT_LEFT = 1;

    public final static int LAYOUT_RIGHT = 2;

    public final static int LAYOUT_CENTER = 3;

    public final static int LAYOUT_TOP = 0x10;

    public final static int LAYOUT_BOTTOM = 0x20;

```

```
public final static int LAYOUT_VCENTER = 0x30;

public final static int LAYOUT_NEWLINE_BEFORE = 0x100;

public final static int LAYOUT_NEWLINE_AFTER = 0x200;

public final static int LAYOUT_SHRINK = 0x400;

public final static int LAYOUT_EXPAND = 0x800;

public final static int LAYOUT_VSHRINK = 0x1000;

public final static int LAYOUT_VEXPAND = 0x2000;

public static final int LAYOUT_2 = 0x4000;

public final static int PLAIN = 0;

public final static int HYPERLINK = 1;

public final static int BUTTON = 2;

private Command commands[];

/*@ behaviour
  @ requires (label instanceof String) || ( label == null );
  @*/
Item(String label);

/*@ behaviour
  @ requires (label instanceof String) || ( label == null );
  @ signals (IllegalStateException ex )
  @   ex.getMessage() != null;
  @*/
public void setLabel(String label);

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result instanceof String);
  @*/
public String getLabel();

}
```

Especificação List.jml

```
package javax.microedition.lcdui;

/*@ model import org.jmlspecs.models.*;

public /*@ pure @*/ class List extends Screen implements Choice {

    public final static Command SELECT_COMMAND =
        new Command("", Command.SCREEN, 0);
```

```
// private members //

private Form form;

private ChoiceGroup cg;

private Command selectCommand = SELECT_COMMAND;

/*@ behaviour
  @ requires (listType == cg.IMPLICIT)||
  @   (listType == cg.EXCLUSIVE)||
  @   (listType == cg.MULTIPLE);
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null &&
  @   ( (listType != cg.IMPLICIT)&&(listType != cg.EXCLUSIVE)&&
  @     (listType != cg.MULTIPLE));
  @*/
public List(String title, int listType);

/*@ behaviour
  @ requires (stringElements != null)&&
  @   (imageElements!= null &&
  @   (stringElements.length == imageElements.length))&&
  @   (\forall int i; 0 <= i &&
  @   i < stringElements.length; stringElements[i] != null)&&
  @   (listType == cg.IMPLICIT)|| (listType == cg.EXCLUSIVE)||
  @   (listType == cg.MULTIPLE);
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && ( (listType != cg.IMPLICIT)&&
  @   (listType != cg.EXCLUSIVE)&&(listType != cg.MULTIPLE));
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null &&
  @   (imageElements!= null &&
  @   (stringElements.length != imageElements.length));
  @ signals ( NullPointerException ex )
  @   ex.getMessage() != null &&
  @   ( (stringElements == null) ||
  @   (\for all int i; 0 <= i &&
  @   i < stringElements.length; stringElements[i]==null) ) ;
  @*/
public List(String title, int listType, String[] stringElements,
            Image[] imageElements);

// public methods //

/*@ behaviour
  @ requires (ticker instanceof Ticker);
  @*/
public void setTicker(Ticker ticker);

/*@ behaviour
  @ requires (s instanceof String);
  @*/
public void setTitle(String s);

/*@ behaviour
  @ assignable \nothing;
```

```

    @ ensures \result >= 0;
    */
public int size();

/*@ behaviour
  @ requires ( elementNum < cg.size());
  @ assignable \nothing;
  @ ensures (\result instanceof String);
  @ signals ( IndexOutOfBoundsException ex )
  @   ex.getMessage() != null && ( elementNum >= cg.size() );
  */
public String getString(int elementNum);

/*@ behaviour
  @ requires ( elementNum < cg.size());
  @ assignable \nothing;
  @ ensures (\result instanceof Image);
  @ signals ( IndexOutOfBoundsException ex )
  @   ex.getMessage() != null && ( elementNum >= cg.size() );
  */
public Image getImage(int elementNum);

/*@ behaviour
  @ requires ( stringPart != null);
  @ ensures (\result >= 0);
  @ signals ( NullPointerException ex )
  @   ex.getMessage() != null && ( stringPart == null );
  */
public int append(String stringPart, Image imagePart);

/*@ behaviour
  @ requires ( elementNum < cg.size()) && ( stringPart != null);
  @ signals ( IndexOutOfBoundsException ex )
  @   ex.getMessage() != null && ( elementNum >= cg.size() );
  @ signals ( NullPointerException ex )
  @   ex.getMessage() != null && ( stringPart == null );
  */
public void insert(int elementNum,
                  String stringPart, Image imagePart);

/*@ behaviour
  @ requires ( elementNum < cg.size());
  @ signals ( IndexOutOfBoundsException ex )
  @   ex.getMessage() != null && ( elementNum >= cg.size() );
  */
public void delete(int elementNum);

/*@ behaviour
  @ ensures cg.size() == 0;
  */
public void deleteAll();

/*@ behaviour
  @ requires ( elementNum < cg.size()) && ( stringPart != null);
  @ signals ( IndexOutOfBoundsException ex )
  @   ex.getMessage() != null && ( elementNum >= cg.size() );
  @ signals ( NullPointerException ex )

```

```

    @ ex.getMessage() != null && ( stringPart == null );
    */
public void set(int elementNum, String stringPart, Image imagePart);

/*@ behaviour
  @ requires ( elementNum < cg.size());
  @ ensures (\result == true) || ( \result == false );
  @ signals ( IndexOutOfBoundsException ex )
  @ ex.getMessage() != null && ( elementNum >= cg.size() );
  */
public boolean isSelected(int elementNum);

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result >= -1;
  */
public int getSelectedIndex();

/*@ behaviour
  @ requires ( \forall int i; 0 <= i &&
  @ i < selectedArray_return.length; (selectedArray_return[i] == true ||
  @ selectedArray_return[i] == false ));
  @ signals ( IllegalArgumentException ex )
  @ ex.getMessage() != null &&
  @ ( selectedArray_return.length < cg.size() );
  @ signals ( NullPointerException ex )
  @ ex.getMessage() != null && ( selectedArray_return == null );
  */
public int getSelectedFlags(boolean[] selectedArray_return);

/*@ behaviour
  @ requires ( elementNum < cg.size() && elementNum >= 0)
  @ && ( selected == true || selected == false);
  @ signals ( IndexOutOfBoundsException ex )
  @ ex.getMessage() != null && ( elementNum >= cg.size() );
  */
public void setSelectedIndex(int elementNum, boolean selected);

/*@ behaviour
  @ requires ( \forall int i; 0 <= i &&
  @ i < selectedArray.length;
  @ (selectedArray[i] == true || selectedArray[i] == false ));
  @ signals ( IllegalArgumentException ex )
  @ ex.getMessage() != null && ( selectedArray.length < cg.size() );
  @ signals ( NullPointerException ex )
  @ ex.getMessage() != null && ( selectedArray == null );
  */
public void setSelectedFlags(boolean[] selectedArray);

/*@ behaviour
  @ requires ( cmd instanceof Command );
  @ assignable \nothing;
  */
public void removeCommand(Command cmd);

/*@ behaviour
  @ requires ( command instanceof Command);
  @ assignable \nothing;

```

```

    @*/
    public void setSelectCommand(Command command);

    /*@ behaviour
    @ requires ( fitPolicy == cg.TEXT_WRAP_DEFAULT )||
    @   ( fitPolicy == cg.TEXT_WRAP_ON)||
    @   ( fitPolicy == cg.TEXT_WRAP_OFF);
    @ signals ( IllegalArgumentException ex )
    @   ex.getMessage() != null &&
    @   (( fitPolicy != cg.TEXT_WRAP_DEFAULT )&&
    @   ( fitPolicy != cg.TEXT_WRAP_ON)&&( fitPolicy != cg.TEXT_WRAP_OFF));
    @*/
    public void setFitPolicy(int fitPolicy);

    /*@ behaviour
    @ assignable \nothing;
    @ ensures ( \result == cg.TEXT_WRAP_DEFAULT )||
    @   ( \result == cg.TEXT_WRAP_ON)|| ( \result == cg.TEXT_WRAP_OFF);
    @*/
    public int getFitPolicy();

    /*@ behaviour
    @ requires (font instanceof Font)&&
    @   ( elementNum >= 0 && elementNum <= cg.size()-1 );
    @ signals ( IllegalArgumentException ex )
    @   ex.getMessage() != null &&
    @   (elementNum < 0 || elementNum > cg.size()-1);
    @*/
    public void setFont(int elementNum, Font font);

    /*@ behaviour
    @ requires ( elementNum > 0 && elementNum <= cg.size()-1 );
    @ ensures ( \result instanceof Font);
    @ signals ( IllegalArgumentException ex )
    @   ex.getMessage() != null && (elementNum < 0 ||
    @   elementNum > cg.size()-1);
    @*/
    public Font getFont(int elementNum);

}

```

Especificação Screen . jml

```

package javax.microedition.lcdui;

/*@ models import org.jmlspecs.models.*;

import java.util.TimerTask;
import java.util.Timer;

public /*@ pure @*/ abstract class Screen extends Displayable {

    final static Font CONTENT_FONT = Font.getDefaultFont();

    final static int CONTENT_HEIGHT = CONTENT_FONT.getHeight();

```



```

int paintBorder;          // = BORDER_NONE;

final static int BORDER_NONE = 0;

final static int BORDER_SOLID = 1;

final static int BORDER_GRAY = 2;

final static boolean SCROLLS_HORIZONTAL = false;

final static boolean SCROLLS_VERTICAL = true;

int view[];

boolean resetToTop = true;

    //@ ensures \result instanceof Screen;
Screen();

/*@ behaviour
    @ requires ( title == null ) || ( title instanceof String );
    @*/
Screen(String title);
}

```

Especificação TextBox.jml

```

package javax.microedition.lcdui;

//@ model import org.jmlspecs.models.*;

import com.sun.midp.lcdui.InputMethodClient;
import com.sun.midp.lcdui.InputMethodHandler;
import com.sun.midp.lcdui.EventHandler;

public /*@ pure @*/ class TextBox extends Screen {

    private Form form;

    private TextField textField;

    /*@ behaviour
        @ requires ( (title instanceof String ) ||(title == null)) ||
        @   ( (text instanceof String ) || (text == null) ) ||
        @   ( maxSize <= getMaxSize() ) ||
        @   ( ( constraints == TextField.CONSTRAINT_MASK) ||
        @   ( constraints == TextField.ANY) ||
        @   ( constraints == TextField.EMAILADDR) ||
        @   ( constraints == TextField.NUMERIC) ||
        @   ( constraints == TextField.PASSWORD) ||
        @   ( constraints == TextField.PHONENUMBER) ||
        @   ( constraints == TextField.URL) );
    */
}

```

```

    @ signals ( IllegalArgumentException e )
    @   e.getMessage() != null && ( maxSize <= 0 |
    @   ( constraints != TextField.CONSTRAINT_MASK)&&
    @   ( constraints != TextField.ANY)&&
    @   ( constraints != TextField.EMAILADDR)&&
    @   ( constraints != TextField.NUMERIC)&&
    @   ( constraints != TextField.PASSWORD)&&
    @   ( constraints != TextField.PHONENUMBER)&&
    @   ( constraints != TextField.URL)) || text.length > getMaxSize();
    @*/
public TextBox(String title, String text, int maxSize, int constraints);

/*@ behaviour
    @ assignable \nothing;
    @ ensures \result instanceof String;
    @*/
public String getString();

/*@ behaviour
    @ requires ( text instanceof String ) || ( text == null );
    @ signals ( IllegalArgumentException e )
    @   e.getMessage() != null && ( text.length > getMaxSize());
    @*/
public void setString(String text);

/*@ behaviour
    @ requires data != null;
    @ assignable \nothing;
    @ ensures \result == data.length();
    @ signals ( ArrayIndexOutOfBoundsException e )
    @   e.getMessage() != null && ( data.length() < getString().length );
    @ signals ( NullPointerException e )
    @   e.getMessage() != null && ( data == null);
    @*/
public int getChars(char[] data);

/*@ behaviour
    @ requires ( offset >= 0 && offset <= data.length ) &&
    @   ( offset + length <= data.length );
    @ signals( ArrayIndexOutOfBoundsException e )
    @   e.getMessage() != null && ( offset + length > data.length );
    @ signals ( IllegalArgumentException e )
    @   e.getMessage() != null && ( getString().length > getMaxSize());
    @*/
public void setChars(char[] data, int offset, int length);

/*@ behaviour
    @ requires ((src instanceof String) && (src != null))&&
    @   ( src.length + position <= getMaxSize());
    @ signals ( NullPointerException e )
    @   e.getMessage() != null && ( src == null);
    @ signals ( IllegalArgumentException e )
    @   e.getMessage() != null && ( src.length + position > getMaxSize());
    @*/
public void insert(String src, int position);

```

```

/*@ behaviour
  @ requires (data != null) && ( offset < data.length && offset >= 0 ) &&
  @   ( length < data.length && length >= 0 ) &&
  @   (position < data.length && position >= 0);
  @ signals ( ArrayIndexOutOfBoundsException ex )
  @   ex.getMessage() != null && ( offset < data.length && offset >= 0 ) &&
  @   ( length < data.length && length >= 0 );
  @ signals ( NullPointerException ex )
  @   ex.getMessage() != null && data == null;
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && ( offset + length > data.length );
  @*/
public void insert(char[] data, int offset, int length, int position);

/*@ behaviour
  @ requires (offset >= 0 && offset <= size())&&
  @   ( offset + length <= size());
  @ assignable \nothing;
  @ signals ( StringIndexOutOfBoundsException ex )
  @   ex.getMessage() != null && ( offset + length > size() );
  @*/
public void delete(int offset, int length);

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result >= 0;
  @*/
public int getMaxSize();

/*@ behaviour
  @ requires maxSize >= 0;
  @ ensures \result >= 0;
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && ( maxSize <= 0 );
  @*/
public int setMaxSize(int maxSize);

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result >= 0;
  @*/
public int size();

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result >= 0;
  @*/
public int getCaretPosition();

/*@ behaviour
  @ requires constraints >= 0;
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && constraints < 0;
  @*/
public void setConstraints(int constraints);

```

```
/*@ behaviour
  @ assignable \nothing;
  @ ensures \result >= 0;
  @*/
public int getConstraints();

/*@ behaviour
  @ requires characterSubset instanceof String;
  @*/
public void setInitialInputMode(String characterSubset);

/*@ behaviour
  @ requires s instanceof String;
  @*/
public void setTitle(String s);

/*@ behaviour
  @ requires ticker instanceof Ticker;
  @*/
public void setTicker(Ticker ticker);
} // class TextBox
```

Especificação Ticker.jml

```
package javax.microedition.lcdui;

/*@ model import org.jmlspecs.models.*;

public /*@ pure @*/ class Ticker {

  /*@ behaviour
    @ requires ( str != null )&&( str instanceof String);
    @ signals (NullPointerException e)
    @   e.getMessage() != null && ( str == null);
    @*/
  public Ticker(String str);

  /*@ behaviour
    @ requires ( str != null )&&( str instanceof String );
    @ signals ( NullPointerException e )
    @   e.getMessage() != null && ( str == null );
    @*/
  public void setString(String str);

  /*@ behaviour
    @ assignable \nothing;
    @ ensures \result != null;
    @*/
  public String getString();

  /*@ behaviour
    @ assignable \nothing;
```

```

    @ requires (g instanceof Graphics);
    */
void paintContent(Graphics g);

private String          message;

private String          displayedMessage;

private int             messageLoc;

private int             messageWidth;

private int             tickSpeed;

private static final Image TICKER_IMG;

static final long       TICK_RATE = 250;

static final int        PREFERRED_HEIGHT;

static final int        DECORATION_HEIGHT = 2;
}

```

Classes do Pacote `javax.microedition.rms`

Especificação `DataConverter.jml`

```

package javax.microedition.rms;

/*@ model import org.jmlspecs.*;

import java.lang.String;

class /*@ pure @*/ DataConverter
{

    /*@ behaviour
       @ ensures \result instanceof DataConverter;
       */
    private DataConverter();

    /*@ behaviour
       @ requires (data.length >= 0) && (offset >= 0);
       @ ensures \result >= 0;
       */
    public static int getInt(byte[] data, int offset);

    /*@ behaviour
       @ requires (data.length >= 0) && (offset >= 0) && ( i >= 0);
       @ ensures \result >= 0;

```

```

    @*/
    public static int putInt(int i, byte[] data, int offset);

    /*@ behaviour
       @ requires (data.length >= 0)&&(offset >= 0);
       @ ensures \result >= 0;
    @*/
    public static long getLong(byte[] data, int offset);

    /*@ behaviour
       @ assignable \nothing;
       @ requires (data.length >= 0)&&(offset >= 0)&(l >=0);
       @ ensures \result >= 0;
    @*/
    public static int putLong(long l, byte[] data, int offset);

    /*@ behaviour
       @ assignable \nothing;
       @ requires (data.length >= 0)&&(offset >= 0);
    @*/
    public static char getChar(byte[] data, int offset);

    /*@ behaviour
       @ assignable \nothing;
       @ requires (data.length >= 0)&&(offset >= 0);
       @ ensures \result >= 0;
    @*/
    public static int putChar(char c, byte[] data, int offset);

    /*@ behaviour
       @ assignable \nothing;
       @ requires (data.length >= 0)&&(offset >= 0)&&(numBytes >= 0);
       @ ensures (\result instanceof String) || (\result == null);
    @*/
    public static String getString(byte[] data, int offset, int numBytes);

    /*@ behaviour
       @ requires (data.length >= 0)&&(offset >= 0)&&(s instanceof String);
       @ ensures \result >= 0;
    @*/
    public static int putString(String s, byte[] data, int offset);
}

```

Especificação InvalidRecordIDException.jml

```

package javax.microedition.rms;

public /*@ pure @*/ class InvalidRecordIDException
    extends RecordStoreException
{
    /*@ behaviour
       @ requires (message instanceof String)|| (message == null);
       @ ensures \result instanceof InvalidRecordIDException;
    @*/
}

```

```
public InvalidRecordIDException(String message);

/*@ behaviour
  @ ensures \result instanceof InvalidRecordIDException;
  @*/
public InvalidRecordIDException();
}

```

Especificação RecordComparator.jml

```
package javax.microedition.rms;

/*@ model import org.jmlspecs.*;

public /*@ pure @*/ interface RecordComparator
{
    public static final int EQUIVALENT = 0;
    public static final int FOLLOWS = 1;
    public static final int PRECEDES = -1;

    /*@ behaviour
      @ assignable \nothing;
      @ requires (rec1.length >= 0) && (rec2.length >= 0);
      @ ensures (\result == PRECEDES) || (\result == FOLLOWS) ||
      @ (\result == EQUIVALENT);
      @*/
    public abstract int compare(byte[] rec1, byte[] rec2);
}

```

Especificação RecordEnumeration.jml

```
package javax.microedition.rms;

/*@ model import org.jmlspecs.*;

public /*@ pure @*/ interface RecordEnumeration
{
    /*@ behaviour
      @ ensures \result >= 0;
      @*/
    public int numRecords();

    /*@ behaviour
      @ ensures (\result).length >= 0;
      @ signals ( InvalidRecordIDException ex)
      @ ex.getMessage() != null;
      @ signals (RecordStoreNotOpenException ex)
      @ ex.getMessage() != null;

```

```
@ signals ( RecordStoreException ex)
@ ex.getMessage() != null;
@*/
public byte[] nextRecord()
    throws InvalidRecordIDException, RecordStoreNotOpenException,
        RecordStoreException;

/*@ behaviour
@ ensures (\result) >= 0;
@ signals ( InvalidRecordIDException ex)
@ ex.getMessage() != null;
@*/
public int nextRecordId()
    throws InvalidRecordIDException;

/*@ behaviour
@ ensures (\result).length >= 0;
@ signals ( InvalidRecordIDException ex)
@ ex.getMessage() != null;
@ signals (RecordStoreNotOpenException ex)
@ ex.getMessage() != null;
@ signals ( RecordStoreException ex)
@ ex.getMessage() != null;
@*/
public byte[] previousRecord()
    throws InvalidRecordIDException, RecordStoreNotOpenException,
        RecordStoreException;

/*@ behaviour
@ ensures (\result) >= 0;
@ signals ( InvalidRecordIDException ex)
@ ex.getMessage() != null && (numRecords() == 0);
@*/
public int previousRecordId()
    throws InvalidRecordIDException;

/*@ behaviour
@ assignable \nothing;
@ ensures (\result == true || \result == false);
@ signals ( InvalidRecordIDException ex)
@*/
public boolean hasNextElement();

/*@ behaviour
@ assignable \nothing;
@ ensures (\result == true || \result == false);
@*/
public boolean hasPreviousElement();

/*@ behaviour
@ assignable \nothing;
@*/
public void reset();

/*@ behaviour
@ assignable \nothing;
@*/
public void rebuild();
```



```

/*@ behaviour
  @ requires (keepUpdated == true || keepUpdate == false );
  @*/
public void keepUpdated(boolean keepUpdated);

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result == true || \result == false);
  @*/
public boolean isKeptUpdated();

/*@ behaviour
  @ requires \nothing;
  @*/
public void destroy();
}

```

Especificação RecordEnumerationImpl.jml

```

package javax.microedition.rms;

/*@ model import org.jmlspecs.*;

class /*@ pure @*/ RecordEnumerationImpl implements RecordEnumeration,
RecordListener
{
    private RecordStore recordStore;

    private RecordFilter filter;

    private RecordComparator comparator;

    private boolean beObserver; // false by default

    private int index; // 0 by default

    private int[] records;

    private static final int NO_SUCH_RECORD = -1;

    /*@ behaviour
      @ ensures \result instanceof RecordEnumerationImpl;
      @*/
    private RecordEnumerationImpl();

    /*@ behaviour
      @ requires (recordStore != null &&
      @   recordStore instanceof RecordStore)&&
      @   ( filter instanceof RecordFilter || filter == null)&&
      @   (comparator instanceof RecordComparator || comparator == null)&&
      @   ( keepUpdated == true || keepUpdated == false);
      @ ensures \result instanceof RecordEnumerationImpl;

```

```
@*/
RecordEnumerationImpl(RecordStore recordStore, RecordFilter filter,
    RecordComparator comparator, boolean keepUpdated);

/*@ behaviour
    @ ensures \result >= 0;
    @*/
public synchronized int numRecords();

/*@ behaviour
    @ ensures (\result).length >= 0;
    @ signals ( InvalidRecordIDException ex)
    @   ex.getMessage() != null;
    @ signals (RecordStoreNotOpenException ex)
    @   ex.getMessage() != null;
    @ signals ( RecordStoreException ex)
    @   ex.getMessage() != null;
    @*/
public synchronized byte[] nextRecord() throws InvalidRecordIDException,
    RecordStoreNotOpenException, RecordStoreException;

/*@ behaviour
    @ ensures (\result) >= 0;
    @ signals ( InvalidRecordIDException ex)
    @   ex.getMessage() != null;
    @*/
public synchronized int nextRecordId()
    throws InvalidRecordIDException;

/*@ behaviour
    @ ensures (\result).length >= 0;
    @ signals ( InvalidRecordIDException ex)
    @   ex.getMessage() != null;
    @ signals (RecordStoreNotOpenException ex)
    @   ex.getMessage() != null;
    @ signals ( RecordStoreException ex)
    @   ex.getMessage() != null;
    @*/
public synchronized byte[] previousRecord() throws
    InvalidRecordIDException, RecordStoreNotOpenException,
    RecordStoreException;

/*@ behaviour
    @ ensures (\result) >= 0;
    @ signals ( InvalidRecordIDException ex)
    @   ex.getMessage() != null && (numRecords() == 0);
    @*/
public synchronized int previousRecordId()
    throws InvalidRecordIDException;

/*@ behaviour
    @ assignable \nothing;
    @ ensures (\result == true || \result == false);
    @ signals ( InvalidRecordIDException ex)
```

```
@*/
public boolean hasNextElement();

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result == true || \result == false);
  @*/
public boolean hasPreviousElement();

/*@ behaviour
  @ assignable \nothing;
  @*/
public void reset();

/*@ behaviour
  @ assignable \nothing;
  @*/
public void rebuild();

/*@ behaviour
  @ requires (keepUpdated == true || keepUpdate == false );
  @*/
public void keepUpdated(boolean keepUpdated)

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result == true || \result == false);
  @*/
public boolean isKeptUpdated()

/*@ behaviour
  @ requires (recordStore instanceof RecordStore) && ( recordId >= 0);
  @*/
public synchronized void recordAdded(RecordStore recordStore,
                                      int recordId);

/*@ behaviour
  @ requires (recordStore instanceof RecordStore) && ( recordId >= 0);
  @*/
public synchronized void recordChanged(RecordStore recordStore,
                                       int recordId);

/*@ behaviour
  @ assignable \nothing;
  @ requires (recordStore instanceof RecordStore) && ( recordId >= 0);
  @*/
public synchronized void recordDeleted(RecordStore recordStore,
                                       int recordId);

/*@ behaviour
  @ requires \nothing;
```

```

    @*/
    public synchronized void destroy();

    /*@ behaviour
       @ assignable \nothing;
    @*/
    private void checkDestroyed();

    /*@ behaviour
       @ requires ( recordId >= 0);
    @*/
    private void filterAdd(int recordId);

    /*@ behaviour
       @ ensures \result >= 0;
       @ signals ( RecordStoreException ex)
       @   ex.getMessage() != null;
    @*/
    private int sortInsert() throws RecordStoreException;

    /*@ behaviour
       @ requires (recordId >= 0);
       @ ensures \result >= 0;
    @*/
    private int findIndexOfRecord(int recordId);

    /*@ behaviour
       @ assignable \nothing;
       @ requires (recIndex >= 0);
    @*/
    private void removeRecordAtIndex(int recIndex);

    /*@ behaviour
       @ requires (filtered.length >= 0);
    @*/
    private void reFilterSort(int[] filtered);

    /*@ behaviour
       @ requires (a.length >= 0)&&( lowIndex >= 0 )&&
       @ (highIndex >= 0)&&( comparator instanceof RecordComparator);
       @ ensures (\forall int i,j;
       @   0 <= i && i < j && j < a.length; a[i] < a[j]);
       @ signals (RecordStoreException ex)
       @   ex.getMessage() != null;
    @*/
    private void QuickSort(int a[], int lowIndex, int highIndex,
        RecordComparator comparator)
        throws RecordStoreException;
}

```

Especificação RecordFilter.jml

```
package javax.microedition.rms;

/*@ model import org.jmlspecs.*;

public /*@ pure @*/ interface RecordFilter
{
    /*@ behaviour
        @ assignable \nothing;
        @ requires candidate.length >= 0;
        @ ensures (\result == true) || (\result==false);
        @*/
    public abstract boolean matches(byte[] candidate);
}
```

Especificação RecordListener.jml

```
package javax.microedition.rms;

/*@ model import org.jmlspecs.*;

public /*@ pure @*/ interface RecordListener
{
    /*@ behaviour
        @ requires (recordStore instanceof RecordStore) && ( recordId >= 0);
        @*/
    public abstract void recordAdded(RecordStore recordStore, int recordId);

    /*@ behaviour
        @ requires (recordStore instanceof RecordStore) && ( recordId >= 0);
        @*/
    public abstract void recordChanged(RecordStore recordStore, int recordId);

    /*@ behaviour
        @ requires (recordStore instanceof RecordStore) && ( recordId >= 0);
        @*/
    public abstract void recordDeleted(RecordStore recordStore, int recordId);
}
```

Especificação RecordStore.jml

```
package javax.microedition.rms;

/*@ model import org.jmlspecs.*;

import com.sun.midp.rms.RecordStoreFile;
```

```

public /*@ pure @*/ class RecordStore
{
    /*@ behaviour
    @ ensures \result instanceof RecordStore;
    @*/
    private RecordStore();

    /*@ behaviour
    @ requires (recordStoreName instanceof String)&&
    @   ( recordStoreName != null);
    @ signals(RecordStoreException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreNotFoundException ex)
    @   ex.getMessage() != null;
    @*/
    public static void deleteRecordStore(String recordStoreName)
        throws RecordStoreException, RecordStoreNotFoundException;

    /*@ behaviour
    @ requires (recordStoreName instanceof String)&&
    @   ( recordStoreName != null)&&
    @   ( createIfNecessary == true || createIfNecessary == false);
    @ signals(RecordStoreException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreNotFoundException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreFullException ex)
    @   ex.getMessage() != null;
    @ signals(IllegalArgumentException ex)
    @   ex.getMessage() != null;
    @*/
    public static RecordStore openRecordStore(String recordStoreName,
                                             boolean createIfNecessary)
        throws RecordStoreException, RecordStoreFullException,
        RecordStoreNotFoundException;

    /*@ behaviour
    @ requires (recordStoreName instanceof String)&&
    @   ( recordStoreName != null)&&
    @   ( createIfNecessary == true || createIfNecessary == false)&&
    @   ( authmode == AUTHMODE_PRIVATE ||
    @     authmode == AUTHMODE_ANY)&(writable == true ||
    @     writable == false );
    @ signals(RecordStoreException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreNotFoundException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreFullException ex)
    @   ex.getMessage() != null;
    @ signals(IllegalArgumentException ex)
    @   ex.getMessage() != null;
    @*/
    public static RecordStore openRecordStore(String recordStoreName,
                                             boolean createIfNecessary,
                                             int authmode,
                                             boolean writable)
        throws RecordStoreException, RecordStoreFullException,

```

```

RecordStoreNotFoundException;

/*@ behaviour
  @ requires (recordStoreName instanceof String)&&
  @   ( recordStoreName != null)&(vendorName instanceof String)&&
  @   ( vendorName != null)&&(suiteName instanceof String)&&
  @   ( suiteName != null);
  @ signals(RecordStoreException ex)
  @   ex.getMessage() != null;
  @ signals(RecordStoreNotFoundException ex)
  @   ex.getMessage() != null;
  @ signals(SecurityException ex)
  @   ex.getMessage() != null;
  @ signals(IllegalArgumentException ex)
  @   ex.getMessage() != null;
  @*/
public static RecordStore openRecordStore(String recordStoreName,
                                         String vendorName,
                                         String suiteName)
    throws RecordStoreException, RecordStoreNotFoundException;

public final static int AUTHMODE_PRIVATE = 0;

public final static int AUTHMODE_ANY = 1;

private final static int AUTHMODE_ANY_RO = 2;

/*@ behaviour
  @ requires (authmode == AUTHMODE_PRIVATE || authmode == AUTHMODE+ANY)&&
  @   ( writable == true || writable == false);
  @ signals(RecordStoreException ex)
  @   ex.getMessage() != null;
  @ signals(SecurityException ex)
  @   ex.getMessage() != null;
  @ signals(IllegalArgumentException ex)
  @   ex.getMessage() != null;
  @*/
public void setMode(int authmode,
                   boolean writable)
    throws RecordStoreException;

/*@ behaviour
  @ assignable \nothing;
  @ signals(RecordStoreException ex)
  @   ex.getMessage() != null;
  @ signals(RecordStoreNotOpenException ex)
  @   ex.getMessage() != null;
  @*/
public void closeRecordStore()
    throws RecordStoreNotOpenException, RecordStoreException;

/*@ behaviour
  @ ensures (\forall int i; i >= 0 &&
  @   i < (\result).length; (\result)[i] instanceof String);
  @*/
public static String[] listRecordStores();

```

```
/*@ behaviour
  @ ensures \result instanceof String;
  @ signals (RecordStoreNotOpenException ex)
  @   ex.getMessage() != null;
  @*/
public String getName()
  throws RecordStoreNotOpenException;

/*@ behaviour
  @ ensures \result >= 0;
  @ signals (RecordStoreNotOpenException ex)
  @   ex.getMessage() != null;
  @*/
public int getVersion()
  throws RecordStoreNotOpenException;

/*@ behaviour
  @ ensures \result >= 0;
  @ signals (RecordStoreNotOpenException ex)
  @   ex.getMessage() != null;
  @*/
public int getNumRecords()
  throws RecordStoreNotOpenException;

/*@ behaviour
  @ ensures \result >= 0;
  @ signals (RecordStoreNotOpenException ex)
  @   ex.getMessage() != null;
  @*/
public int getSize()
  throws RecordStoreNotOpenException;

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result >= 0;
  @ signals (RecordStoreNotOpenException ex)
  @   ex.getMessage() != null;
  @*/
public int getSizeAvailable()
  throws RecordStoreNotOpenException;

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result >= 0;
  @ signals (RecordStoreNotOpenException ex)
  @   ex.getMessage() != null;
  @*/
public long getLastModified()
  throws RecordStoreNotOpenException;

/*@ behaviour
  @ requires listener instanceof RecordListener;
  @*/
public void addRecordListener(RecordListener listener);

/*@ behaviour
```



```

    @ assignable \nothing;
    @ requires listener instanceof RecordListener;
    @*/
public void removeRecordListener(RecordListener listener);

/*@ behaviour
    @ assignable \nothing;
    @ ensures \result >= 0;
    @ signals(RecordStoreNotOpenException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreException ex)
    @   ex.getMessage() != null;
    @*/
public int getNextRecordID()
    throws RecordStoreNotOpenException, RecordStoreException;

/*@ behaviour
    @ assignable \nothing;
    @ requires ( data.length > 0 || data == null)&&( offset >= 0)&&
    @   (numBytes >= 0);
    @ ensures \result >= 0;
    @ signals(RecordStoreNotOpenException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreFullException ex)
    @   ex.getMessage() != null;
    @ signals(SecurityException ex)
    @   ex.getMessage() != null;
    @*/
public int addRecord(byte[] data, int offset, int numBytes)
    throws RecordStoreNotOpenException, RecordStoreException,
    RecordStoreFullException;

/*@ behaviour
    @ assignable \nothing;
    @ requires recordId >= 0;
    @ signals(RecordStoreNotOpenException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreException ex)
    @   ex.getMessage() != null;
    @ signals(InvalidRecordIDException ex)
    @   ex.getMessage() != null;
    @ signals(SecurityException ex)
    @   ex.getMessage() != null;
    @*/
public void deleteRecord(int recordId)
    throws RecordStoreNotOpenException, InvalidRecordIDException,
    RecordStoreException;

/*@ behaviour
    @ assignable \nothing;
    @ requires recordId >= 0;
    @ ensures \result >= 0;
    @ signals(RecordStoreNotOpenException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreException ex)
    @   ex.getMessage() != null;
    @

```

```

    @ signals(InvalidRecordIDException ex)
    @   ex.getMessage() != null;
    @ signals(SecurityException ex)
    @   ex.getMessage() != null;
    @*/
public int getRecordSize(int recordId)
    throws RecordStoreNotOpenException, InvalidRecordIDException,
           RecordStoreException;

/*@ behaviour
    @ assignable \nothing;
    @ requires (recordId >= 0)&&( buffer.length > 0)&&(offset >= 0);
    @ ensures \result >= 0;
    @ signals(RecordStoreNotOpenException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreException ex)
    @   ex.getMessage() != null;
    @ signals(InvalidRecordIDException ex)
    @   ex.getMessage() != null;
    @ signals(ArrayIndexOutOfBoundsException ex)
    @   ex.getMessage() != null;
    @*/
public int getRecord(int recordId, byte[] buffer, int offset)
    throws RecordStoreNotOpenException, InvalidRecordIDException,
           RecordStoreException;

/*@ behaviour
    @ assignable \nothing;
    @ requires recordId >= 0;
    @ ensures (\result).length >= 0;
    @ signals(RecordStoreNotOpenException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreException ex)
    @   ex.getMessage() != null;
    @ signals(InvalidRecordIDException ex)
    @   ex.getMessage() != null;
    @*/
public byte[] getRecord(int recordId)
    throws RecordStoreNotOpenException, InvalidRecordIDException,
           RecordStoreException;

/*@ behaviour
    @ assignable \nothing;
    @ requires (recordId >= 0) && ( newData.length > 0)&&
    @   (offset >= 0)&&( numBytes >= 0);
    @ signals(RecordStoreNotOpenException ex)
    @   ex.getMessage() != null;
    @ signals(RecordStoreException ex)
    @   ex.getMessage() != null;
    @ signals(InvalidRecordIDException ex)
    @   ex.getMessage() != null;
    @ signals(SecurityException ex)
    @   ex.getMessage() != null;
    @ signals( RecordStoreFullException ex)
    @   ex.getMessage() != null;
    @*/
public void setRecord(int recordId, byte[] newData,
                     int offset, int numBytes)
    throws RecordStoreNotOpenException, InvalidRecordIDException,
           RecordStoreException, RecordStoreFullException;

```

```
/*@ behaviour
  @ requires ((filter instanceof RecordFilter) &&
  @   (filter != null)) && ( (comparator instanceof RecordComparator)&&
  @   ( comparator != null) )&&
  @   ( keepUpdate == true || keepUpdate == false);
  @ signals(RecordStoreNotOpenException ex)
  @   ex.getMessage() != null;
  @*/
public RecordEnumeration enumerateRecords(RecordFilter filter,
                                         RecordComparator comparator,
                                         boolean keepUpdated)
    throws RecordStoreNotOpenException;

private static final int SIGNATURE_LENGTH = 8;

private static final int DB_RECORD_HEADER_LENGTH = 16;

private static final int DB_BLOCK_SIZE = 16;

private static final int DB_COMPACTBUFFER_SIZE = 64;

private static java.util.Vector dbCache = new java.util.Vector(3);

private static final Object dbCacheLock = new Object();

private String recordStoreName;

private String uniqueIdPath;

private int opencount;

private RecordStoreFile dbrf;

Object rsLock;

private java.util.Vector recordListener;

private RecordHeaderCache recHeadCache;

private static int CACHE_SIZE = 8;

private static byte[] recHeadBuf = new byte[DB_RECORD_HEADER_LENGTH];

private int dbNextRecordID = 1;

private int dbVersion;

private int dbAuthMode;

private int dbNumLiveRecords;

private long dbLastModified;

private int dbFirstRecordOffset;

private int dbFirstFreeBlockOffset;

private int dbDataStart = 48;
```

```
private int dbDataEnd = 48;

private static byte[] dbState = new byte[DB_INIT.length];

private static final int RS_SIGNATURE = 0;

private static final int RS_NUM_LIVE = 8;

private static final int RS_AUTHMODE = 12;

private static final int RS_VERSION = 16;

private static final int RS_NEXT_ID = 20;

private static final int RS_REC_START = 24;

private static final int RS_FREE_START = 28;

private static final int RS_LAST_MODIFIED = 32;

private static final int RS_DATA_START = 40;

private static final int RS_DATA_END = 44;

}
```

Especificação RecordStoreException.jml

```
package javax.microedition.rms;

/*@ model import org.jmlspecs.*;

public /*@ pure @*/ class RecordStoreException
    extends java.lang.Exception
{
    /*@ behaviour
        @ requires (message instanceof String) || (message == null);
        @ ensures \result instanceof RecordStoreException;
        @*/
    public RecordStoreException(String message);

    /*@ behaviour
        @ ensures \result instanceof RecordStoreException;
        @*/
    public RecordStoreException();
}
```

Especificação RecordStoreFullException.jml

```
package javax.microedition.rms;

/*@ model import org.jmlspecs.*;
```

```
public /*@ pure @*/ class RecordStoreFullException
    extends RecordStoreException
{
    /*@ behaviour
    @ requires (message instanceof String)|| ( message == null);
    @ ensures \result instanceof RecordStoreFullException;
    @*/
    public RecordStoreFullException(String message);

    /*@ behaviour
    @ ensures \result instanceof RecordStoreFullException;
    @*/
    public RecordStoreFullException();
}
```

Especificação RecordStoreNotFoundException.jml

```
package javax.microedition.rms;

/*@ model import org.jmlspecs.*;

public /*@ pure @*/ class RecordStoreNotFoundException
    extends RecordStoreException
{
    /*@ behaviour
    @ requires (message instanceof String)|| ( message == null);
    @ ensures \result instanceof RecordStoreNotFoundException;
    @*/
    public RecordStoreNotFoundException(String message);

    /*@ behaviour
    @ ensures \result instanceof RecordStoreNotFoundException;
    @*/
    public RecordStoreNotFoundException();
}
```

Especificação RecordStoreNotOpenException.jml

```
package javax.microedition.rms;

/*@ model import org.jmlspecs.*;

public /*@ pure @*/ class RecordStoreNotOpenException
    extends RecordStoreException
{
    /*@ behaviour
    @ requires (message instanceof String)|| ( message == null);
    @ ensures \result instanceof RecordStoreNotOpenException;
    @*/
```

```
public RecordStoreNotOpenException(String message);

    /*@ behaviour
       @ ensures \result instanceof RecordStoreNotOpenException;
    @*/
public RecordStoreNotOpenException();
}
```

Classes do Pacote `javax.microedition.io`

Especificação `CommConnection.jml`

```
package javax.microedition.io;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public interface CommConnection
    extends StreamConnection {

    /*@ behaviour
       @ assignable \nothing;
       @ ensures \result >= 0;
    @*/
    public int getBaudRate();

    /*@ behaviour
       @ requires baudrate >= 0;
       @ ensures \result >= 0;
    @*/
    public int setBaudRate(int baudrate);

}
```

Especificação `HttpConnection.jml`

```
package javax.microedition.io;

/*@ model import org.jmlspecs.*;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.String;
import javax.microedition.io.ContentConnection;
```

```
public /*@ pure @*/ interface HttpConnection extends ContentConnection {

    public final static String HEAD = "HEAD";

    public final static String GET = "GET";

    public final static String POST = "POST";

    public static final int HTTP_OK = 200;

    //Existem outras constantes que foram omitidas

    /*@ behaviour
    @ assignable \nothing;
    @ ensures (\result instanceof String);
    @*/
    public String getURL();

    /*@ behaviour
    @ assignable \nothing;
    @ ensures (\result instanceof String);
    @*/
    public String getProtocol();

    /*@ behaviour
    @ assignable \nothing;
    @ ensures (\result instanceof String);
    @*/
    public String getHost();

    /*@ behaviour
    @ assignable \nothing;
    @ ensures (\result == null) || ( \result instanceof String );
    @*/
    public String getFile();

    /*@ behaviour
    @ assignable \nothing;
    @ ensures (\result == null) || ( \result instanceof String );
    @*/
    public String getRef();

    /*@ behaviour
    @ assignable \nothing;
    @ ensures (\result == null) || ( \result instanceof String );
    @*/
    public String getQuery();

    /*@ behaviour
    @ assignable \nothing;
    @ ensures (\result == 80) || ( \result >= 0 );
    @*/
    public int getPort();

    /*@ behaviour
    @ assignable \nothing;
    @ ensures (\result.equals(HEAD)) || (\result.equals(GET)) ||
```

```
@ ( \result.equals(POST));
@*/
public String getRequestMethod();

/*@ behaviour
@ requires (method.equals(HEAD)) || (method.equals(GET)) ||
@ ( method.equals(POST));
@ signals ( IOException ex )
@ ex.getMessage() != null &&
@ ((!method.equals(HEAD)) && (!method.equals(GET)) &&
@ ( !method.equals(POST)));
@*/
public void setRequestMethod(String method) throws IOException;

/*@ behaviour
@ assignable \nothing;
@ requires (key instanceof String);
@ ensures (\result instanceof String) || ( \result == null );
@*/
public String getRequestProperty(String key);

/*@ behaviour
@ requires (key instanceof String) && ( value instanceof String );
@*/
public void setRequestProperty(String key, String value) throws
IOException;

/*@ behaviour
@ assignable \nothing;
@ ensures (\result == -1) || ( \result >= 200);
@*/
public int getResponseCode() throws IOException;

/*@ behaviour
@ assignable \nothing;
@ ensures (\result instanceof String);
@*/
public String getResponseMessage() throws IOException;

/*@ behaviour
@ assignable \nothing;
@ ensures (\result >= 0 );
@*/
public long getExpiration() throws IOException;

/*@ behaviour
@ assignable \nothing;
@ ensures (\result >= 0 );
@*/
public long getDate() throws IOException;

/*@ behaviour
@ assignable \nothing;
@ ensures (\result >= 0 );
@*/
public long getLastModified() throws IOException;

/*@ behaviour
@ assignable \nothing;
```



```

    @ requires (name instanceof String && (name != null));
    @ ensures (\result instanceof String) || ( \result == null);
    */
public String getHeaderField(String name) throws IOException;

/*@ behaviour
    @ assignable \nothing;
    @ requires (name instanceof String && (name != null)) && def >= 0;
    @ ensures (\result >= 0 );
    */
public int getHeaderFieldInt(String name, int def) throws IOException;

/*@ behaviour
    @ assignable \nothing;
    @ requires ((name instanceof String) && (name != null)) && def >= 0;
    @ ensures (\result >= 0 );
    */
public long getHeaderFieldDate(String name, long def) throws IOException;

/*@ behaviour
    @ assignable \nothing;
    @ requires (n >= 0);
    @ ensures (\result instanceof String)||(\result == null);
    */
public String getHeaderField(int n) throws IOException;

/*@ behaviour
    @ assignable \nothing;
    @ requires (n >= 0);
    @ ensures (\result instanceof String)||(\result == null);
    */
public String getHeaderFieldKey(int n) throws IOException;
}

```

Especificação `HttpsConnection.jml`

```

package javax.microedition.io;

/*@ model import org.jmlspecs.*;

import java.lang.String;
import java.io.IOException;
import javax.microedition.pki.Certificate;
import javax.microedition.pki.CertificateException;

public /*@ pure @*/ interface HttpsConnection extends HttpURLConnection {

    /*@ behaviour
    @ assignable \nothing;
    @ ensures \result instanceof SecurityInfo;
    */

```

```

public SecurityInfo getSecurityInfo()
    throws IOException;

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result == 443) || ( \result > 0 );
  @*/
public int getPort();
}

```

Especificação PushRegistry.jml

```

package javax.microedition.io;

/*@ model import org.jmlspecs.*;

import java.lang.ClassNotFoundException;
import java.lang.IllegalStateException;
import java.lang.IllegalArgumentException;
import java.lang.String;
import java.io.IOException;
import java.util.Date;
import java.util.Timer;
import java.util.TimerTask;

import com.sun.midp.io.Util;
import com.sun.midp.io.j2me.push.PushRegistryImpl;

public /*@ pure @*/ class PushRegistry {

    /*@ ensures \result instanceof PushRegistry;
    private PushRegistry() { };

    /*@ behaviour
      @ requires (connection instanceof String && connection != null)&&
      @   (midlet instanceof String && midlet != null)&&
      @   (filter instanceof String && filter != null);
      @ signals (IllegalArgumentException ex)
      @   ex.getMessage() != null && ( connection == null || filter == null );
      @ signals ( ClassNotFoundException ex)
      @   ex.getMessage() != null && ( midlet == null);
      @*/
    public static void registerConnection(String connection, String midlet,
                                         String filter)
        throws ClassNotFoundException,
            IOException;

    /*@ behaviour
      @ requires (connection instanceof String || connection == null);
      @ ensures (\result == true)||(\result == false);
      @*/
    public static boolean unregisterConnection(String connection);
}

```

```

/*@ behaviour
  @ requires (available == true)|| (available == false) ;
  @ ensures (\forall int i; i >= 0 &&
    @   i < \result.length; (\result[i] instanceof String));
  @*/
public static String[] listConnections(boolean available);

/*@ behaviour
  @ requires (connection instanceof String);
  @ ensures (\result instanceof String)||(\result == null);
  @*/
public static String getMIDlet(String connection);

/*@ behaviour
  @ requires (connection instanceof String);
  @ ensures (\result instanceof String)||(\result == null);
  @*/
public static String getFilter(String connection);

/*@ behaviour
  @ requires (midlet instanceof String) && ( time >= 0);
  @ ensures (\result >= 0);
  @*/
public static long registerAlarm(String midlet, long time)
    throws ClassNotFoundException, ConnectionNotFoundException;
}

```

Especificação SecureConnection.jml

```

package javax.microedition.io;

/*@ model import org.jmlspecs.*;

import java.lang.String;
import java.lang.IllegalArgumentException;
import java.io.IOException;
import javax.microedition.pki.CertificateException;

public /*@ pure @*/ interface SecureConnection extends SocketConnection {

  /*@ normal_behaviour
    @ assignable \nothing;
    @ ensures \result instanceof SecurityInfo;
    @*/
  public SecurityInfo getSecurityInfo() throws IOException;
}

```

Especificação SecurityInfo.jml

```
package javax.microedition.io;

/*@ model import org.jmlspecs.*;

import java.lang.String;
import java.io.IOException;
import javax.microedition.pki.Certificate;
import javax.microedition.pki.CertificateException;

public /*@ pure @*/ interface SecurityInfo {

    /*@ behaviour
       @ assignable \nothing;
       @ ensures \result instanceof Certificate;
    @*/
    public Certificate getServerCertificate();

    /*@ behaviour
       @ assignable \nothing;
       @ ensures
       @ ( (\result instanceof String) &&
         @ (\result != null) ) &&
       @ ((\result.equals("3.0")) || (\result.equals("3.1")) || (\result.equals("1"))
         @ );
    @*/
    public String getProtocolVersion();

    /*@ behaviour
       @ assignable \nothing;
       @ ensures ((\result instanceof String) && (\result != null));
    @*/
    public String getProtocolName();

    /*@ behaviour
       @ assignable \nothing;
       @ ensures ((\result instanceof String) && (\result != null));
    @*/
    public String getCipherSuite();
}
```

Especificação `ServerSocketConnection.jml`

```
package javax.microedition.io;

/*@ model import org.jmlspecs.*;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;

public /*@ pure @*/ interface ServerSocketConnection
    extends StreamConnectionNotifier {

    /*@ behaviour
       @ assignable \nothing;
```

```

    @ ensures (\result instanceof String) && ( \result != null);
    */
public String getLocalAddress() throws IOException;

/*@ behaviour
    @ assignable \nothing;
    @ ensures (\result >= 0);
    */
public int getLocalPort() throws IOException;

}

```

Especificação SocketConnection.jml

```

package javax.microedition.io;

/*@ model import org.jmlspecs.*;

import java.lang.IllegalArgumentException;
import java.io.IOException;
import java.lang.String;

public /*@ pure @*/ interface SocketConnection
    extends StreamConnection {

    public final byte DELAY = 0;

    public final byte LINGER = 1;

    public final byte KEEPALIVE = 2;

    public final byte RCVBUF = 3;

    public final byte SNDBUF = 4;

    /*@ behaviour
        @ requires (option == KEEPALIVE)|| (option == LINGER )||
        @   ( option == SNDBUF )||(option == RCVBUF)|| (option == DELAY);
        @ signals ( IllegalArgumentException ex )
        @   ex.getMessage() != null && ( (option < 0)||
        @   ( ( option != LINGER )||(option != SNDBUF )||
        @   (option != RCVBUF)|| (option != DELAY) ))
        */
    public void setSocketOption(byte option, int value)
        throws IllegalArgumentException, IOException;

    /*@ behaviour
        @ requires (option == KEEPALIVE)|| (option == LINGER )||
        @   ( option == SNDBUF )||(option == RCVBUF)||
        @   (option == DELAY);
        @ ensures (\result >= 0)|| ( \result == -1);
        @ signals ( IllegalArgumentException ex )
        @   ex.getMessage() != null && ( (option < 0)||
        @   ( ( option != LINGER )||(option != SNDBUF )||

```

```
    @ (option != RCVBUF)|| (option != DELAY) )
    */
public int getSocketOption(byte option)
    throws IllegalArgumentException, IOException;

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result instanceof String)&&( \result != null);
  */
public String getLocalAddress() throws IOException;

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result > 0);
  */
public int getLocalPort() throws IOException;

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result instanceof String)&&( \result != null);
  */
public String getAddress() throws IOException;

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result > 0);
  */
public int getPort() throws IOException;
}
```

Especificação UDPDatagramConnection.jml

```
package javax.microedition.io;

/*@ model import org.jmlspecs.*;

import java.io.IOException;

public /*@ pure @*/ interface UDPDatagramConnection extends DatagramConnection
{

    /*@ behaviour
      @ assignable \nothing;
      @ ensures (\result instanceof String)&&( \result != null);
      */
    public String getLocalAddress() throws IOException;

    /*@ behaviour
      @ assignable \nothing;
      @ ensures (\result >= 0);
      */
    public int getLocalPort() throws IOException;

}
```

Classes do Pacote `javax.microedition.pki`

Especificação `Certificate.jml`

```
package javax.microedition.pki;

import java.lang.String;

public /*@ pure @*/ interface Certificate {

    /*@ behaviour
       @ assignable \nothing;
       @ ensures (\result instanceof String )&&(\result != null);
    @*/
    public String getSubject();

    /*@ behaviour
       @ assignable \nothing;
       @ ensures (\result instanceof String )&&(\result != null);
    @*/
    public String getIssuer();

    /*@ behaviour
       @ assignable \nothing;
       @ ensures \result != null;
    @*/
    public String getType();

    /*@ behaviour
       @ assignable \nothing;
       @ ensures (\result instanceof String )&&(\result != null);
    @*/
    public String getVersion();

    /*@ behaviour
       @ assignable \nothing;
       @ ensures (\result instanceof String )&&(\result != null);
    @*/
    public String getSigAlgName();

    /*@ behaviour
       @ assignable \nothing;
       @ ensures \result >= 0;
    @*/
    public long getNotBefore();

    /*@ behaviour
       @ assignable \nothing;
       @ ensures (\result > 0 || \result == Long.MAX_VALUE);
    @*/
    public long getNotAfter();
```

```
/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result instanceof String) || (\result == null);
  @*/
public String getSerialNumber();
}

```

Especificação CertificateException.jml

```
package javax.microedition.pki;

import javax.microedition.pki.Certificate;
import java.lang.String;

public /*@ pure @*/ class CertificateException extends java.io.IOException {
    private byte reason;

    private Certificate cert;

    public static final byte BAD_EXTENSIONS = 1;

    public static final byte CERTIFICATE_CHAIN_TOO_LONG = 2;

    public static final byte EXPIRED = 3;

    public static final byte UNAUTHORIZED_INTERMEDIATE_CA = 4;

    public static final byte MISSING_SIGNATURE = 5;

    public static final byte NOT_YET_VALID = 6;

    public static final byte SITENAME_MISMATCH = 7;

    public static final byte UNRECOGNIZED_ISSUER = 8;

    public static final byte UNSUPPORTED_SIGALG = 9;

    public static final byte INAPPROPRIATE_KEY_USAGE = 10;

    public static final byte BROKEN_CHAIN = 11;

    public static final byte ROOT_CA_EXPIRED = 12;

    public static final byte UNSUPPORTED_PUBLIC_KEY_TYPE = 13;

    public static final byte VERIFICATION_FAILED = 14;

    /*@ behaviour
      @ requires (certificate instanceof Certificate) &&
      @   ( status >=1 && status <= 14);
      @*/
    public CertificateException(Certificate certificate, byte status);
}

```



```

/*@ behaviour
  @ requires (certificate instanceof Certificate) &
  @   ( status >=1 && status <= 14 ) & (message instanceof String);
  @*/
public CertificateException(String message, Certificate certificate,
                           byte status);

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result instanceof Certificate;
  @*/
public Certificate getCertificate();

/*@ behaviour
  @ assignable \nothing;
  @ ensures (\result >= 1)&&(\result <= 14);
  @*/
public byte getReason();
}

```

Classes do Pacote `javax.microedition.midlet`

Especificação `MIDlet.jml`

```

package javax.microedition.midlet;

/*@ model import org.jmlspecs.*;

import com.sun.midp.midlet.MIDletState;

public /*@ pure @*/ abstract class MIDlet {

    private MIDletProxy state;

    /*@ behaviour
      @ assignable \nothing;
      @ ensures \result instanceof MIDletProxy;
      @*/
    MIDletProxy getProxy();

    /*@ behaviour
      @ assignable \nothing;
      @ ensures \result instanceof MIDlet;
      @*/
    protected MIDlet();

    /*@ behaviour
      @ assignable \nothing;
      @ signals( MIDletStateException ex)

```

```

    @ ex.getMessage() != null;
    @*/
protected abstract void startApp() throws MIDletStateChangeException;

/*@ behaviour
    @ assignable \nothing;
    @*/
protected abstract void pauseApp();

/*@ behaviour
    @ assignable \nothing;
    @ requires unconditional == true || unconditional == false;
    @ signals (MIDletStateChangeException ex)
    @ ex.getMessage() != null && (unconditional != true);
    @*/
protected abstract void destroyApp(boolean unconditional)
    throws MIDletStateChangeException;

/*@ behaviour
    @ assignable \nothing;
    @*/
public final void notifyDestroyed();

/*@ behaviour
    @ assignable \nothing;
    @*/
public final void notifyPaused();

/*@ behaviour
    @ assignable \nothing;
    @ requires (key != null)&&( key instanceof String );
    @ ensures (\result instanceof String)||(\result == null);
    @ signals (NullPointerException ex)
    @ ex.getMessage() != null && ( key == null);
    @*/
public final String getAppProperty(String key);

/*@ behaviour
    @ assignable \nothing;
    @*/
public final void resumeRequest();

/*@ behaviour
    @ assignable \nothing;
    @ requires URL instanceof String;
    @ ensures (\result == true) || (\result == false);
    @ signals ( javax.microedition.io.ConnectionNotFoundException ex )
    @ ex.getMessage() != null;
    @*/
public final boolean platformRequest(String URL)
    throws javax.microedition.io.ConnectionNotFoundException;

/*@ behaviour
    @ assignable \nothing;
    @ requires permission instanceof String;
    @ ensures (\result == 0)|(\result == 1)|(\result == -1);
    @*/
public final int checkPermission(String permission);

```

```
}
```

Especificação MIDletProxy .jml

```
package javax.microedition.midlet;

/*@ model import org.jmlspecs.*;

import com.sun.midp.midlet.MIDletState;
import com.sun.midp.midlet.MIDletStateMap;

class /*@ pure @*/ MIDletProxy extends MIDletState {

    /*@ behaviour
       @ requires m instanceof MIDlet;
       @*/
    MIDletProxy(MIDlet m);

    /*@ behaviour
       @ signals ( MIDletStateChangeException ex )
       @   ex.getMessage() != null;
       @*/
    protected void startApp() throws MIDletStateChangeException;

    /*@ behaviour
       @ ensures false;
       @*/
    protected void pauseApp();

    /*@ behaviour
       @ also
       @ requires (unconditional == true) || ( unconditional == false ) ;
       @ signals ( MIDletStateChangeException ex )
       @   ex.getMessage() != null;
       @*/
    protected void destroyApp(boolean unconditional)
        throws MIDletStateChangeException;

}
```

Especificação MIDletStateChangeException .jml

```
package javax.microedition.midlet;

/*@ model import org.jmlspecs.*;

import java.lang.String;

public /*@ pure @*/ class MIDletStateChangeException extends Exception {

    /*@ behaviour
```

```
    @ ensures \result instanceof MIDletStateChangeException;
    @*/
    public MIDletStateChangeException();

    /*@ behaviour
    @ requires s instanceof String;
    @ ensures \result instanceof MIDletStateChangeException;
    @*/
    public MIDletStateChangeException(String s);
}

```

Especificação MIDletStateMapImpl . jml

```
package javax.microedition.midlet;

/*@ model import org.jmlspecs.*;

import com.sun.midp.midlet.MIDletState;
import com.sun.midp.midlet.MIDletStateMap;

class /*@ pure @*/ MIDletStateMapImpl extends MIDletStateMap {

    /*@ behaviour
    @ ensures \result instanceof MIDletStateMapImpl;
    @*/
    MIDletStateMapImpl();

    /*@ behaviour
    @ requires (m instanceof MIDlet)&&( m != null);
    @ ensures \result instanceof MIDletState;
    @ signals ( NullPointerException ex )
    @ ex.getMessage() != null && ( m == null);
    @*/
    protected MIDletState getStateImpl(MIDlet m);
}

```

Classes do Pacote java . lang

Especificação Class . jml

```
package java.lang;

/*@ model import org.jmlspecs.*;

import com.sun.midp.io.ResourceInputStream;

```

```

public /*@ pure @*/ final
class Class {

    /*@ ensures \result instanceof Class;
private Class();

    /*@ also
    @ behaviour
    @ assignable \nothing;
    @ ensures \result instanceof String;
    @*/
public String toString();

    /*@ behaviour
    @ requires (className instanceof String);
    @ assignable \nothing;
    @ ensures \result instanceof Class;
    @ signals ( ClassNotFoundException ex )
    @   ex.getMessage() != null;
    @*/
public static native Class forName(String className)
    throws ClassNotFoundException;

    /*@ behaviour
    @ assignable \nothing;
    @ ensures \result instanceof Object;
    @ signals ( InstantiationException ex )
    @   ex.getMessage() != null;
    @ signals ( IllegalAccessException ex )
    @   ex.getMessage() != null;
    @*/
public native Object newInstance()
    throws InstantiationException, IllegalAccessException;

    /*@ behaviour
    @ requires (obj instanceof Object);
    @ assignable \nothing;
    @ ensures (\result == true) || (\result == false);
    @*/
public native boolean isInstance(Object obj);

    /*@ behaviour
    @ requires (cls instanceof Class) && ( cls != null);
    @ assignable \nothing;
    @ ensures (\result == true)&&(\result == false) ;
    @ signals ( NullPointerException ex )
    @   ex.getMessage() != null;
    @*/
public native boolean isAssignableFrom(Class cls);

    /*@ behaviour
    @ assignable \nothing;
    @ ensures (\result == true)&&(\result == false) ;
    @*/
public native boolean isInterface();

    /*@ behaviour

```

```

    @ assignable \nothing;
    @ ensures (\result == true)&&(\result == false) ;
    */
public native boolean isArray();

/*@ behaviour
   @ assignable \nothing;
   @ ensures (\result instanceof String )&&(\result != null) ;
   */
public native String getName();

/*@ behaviour
   @ requires (name instanceof String) && ( name != null);
   @ assignable \nothing;
   @ ensures (\result instanceof java.io.InputStream)||(\result == null) ;
   */
public java.io.InputStream getResourceAsStream(String name);
}

```

Especificação Runtime .jml

```

package java.lang;

/*@ model import org.jmlspecs.*;

public /*@ pure */ class Runtime {

    private static Runtime currentRuntime = new Runtime();

    /*@ behaviour
       @ assignable \nothing;
       @ ensures \result instanceof Runtime;
       */
    public static Runtime getRuntime();

    /*@ behaviour
       @ assignable \nothing;
       @ ensures \result instanceof Runtime;
       */
    private Runtime();

    /*@ behaviour
       @ requires status >= 0;
       @ assignable \nothing;
       */
    public void exit(int status);

    /*@ behaviour
       @ assignable \nothing;
       @ ensures \result >= 0;
       */

```

```

public native long freeMemory();

/*@ behaviour
  @ assignable \nothing;
  @ ensures \result >= 0;
  @*/
public native long totalMemory();

/*@ behaviour
  @ assignable \nothing;
  @*/
public native void gc();
}

```

Especificação System.jml

```

package java.lang;

/*@ model import org.jmlspecs.*;

import java.io.*;

import com.sun.midp.io.SystemOutputStream;

public /*@ pure @*/ final class System {

    /*@ behaviour
      @ ensures \result instanceof System;
      @*/
    private System();

    public final static PrintStream out = getConsoleOutput();

    /*@ behaviour
      @ assignable \nothing;
      @ ensures \result instanceof PrintStream;
      @*/
    private static PrintStream getConsoleOutput();

    public final static PrintStream err = out;

    /*@ behaviour
      @ assignable \nothing;
      @ ensures \result >= 0;
      @*/
    public static native long currentTimeMillis();

    /*@ behaviour
      @ requires ( src instanceof Object && src != null)&
      @   ( dst instanceof Object && dst != null)&
      @   ( src_position >= 0 & dst_position >= 0 & length >= 0 );
      @ signals ( NullPointerException ex )
      @   ex.getMessage() != null && ( src == null || dst == null);
      @ signals ( IndexOutOfBoundsException ex )

```

```

    @   ex.getMessage() != null;
    @ signals ( ArrayStoreException ex )
    @   ex.getMessage() != null;
    @*/
public static native void arraycopy(Object src, int src_position,
                                   Object dst, int dst_position,
                                   int length);

/*@ behaviour
   @ requires x instanceof Object;
   @ assignable \nothing;
   @ ensures \result >= 0;
   @*/
public static native int identityHashCode(Object x);

/*@ behaviour
   @ requires ( key instanceof String && key != null);
   @ assignable \nothing;
   @ ensures \result instanceof String;
   @ signals ( NullPointerException ex )
   @   ex.getMessage() != null && ( key == null );
   @ signals ( IllegalArgumentException ex )
   @   ex.getMessage() != null && ( key.equals(" "));
   @*/
public static String getProperty(String key);

/*@ behaviour
   @ requires ( key instanceof String && key != null);
   @ assignable \nothing;
   @ ensures \result instanceof String;
   @ signals ( NullPointerException ex )
   @   ex.getMessage() != null && ( key == null );
   @ signals ( IllegalArgumentException ex )
   @   ex.getMessage() != null && ( key.equals(" "));
   @*/
private native static String getProperty0(String key);

/*@ behaviour
   @ requires ( status >= 0);
   @ assignable \nothing;
   @*/
public static void exit(int status);

/*@ behaviour
   @ assignable \nothing;
   @*/
public static void gc();
}

```

Especificação `IllegalStateException.jml`

```

package java.lang;

/*@ model import org.jmlspecs.*;

```



```
public
/*@ pure @*/ class IllegalStateException extends RuntimeException {

    /*@ behaviour
       @ ensures \result instanceof IllegalStateException;
    @*/
    public IllegalStateException();

    /*@ behaviour
       @ requires s instanceof String;
       @ ensures \result instanceof IllegalStateException;
    @*/
    public IllegalStateException(String s);
}
```

Classes do Pacote `java.util`

Especificação `Timer.jml`

```
package java.util;

/*@ model import org.jmlspecs.*;

import java.util.Date;

public /*@ pure @*/ class Timer {

    private TaskQueue queue = new TaskQueue();

    private TimerThread thread = new TimerThread(queue);

    /*@ behaviour
       @ assignable \nothing;
       @ ensures \result instanceof Timer;
    @*/
    public Timer();

    /*@ behaviour
       @ requires ( task instanceof TimerTask && ( task != null)) &&
       @   ( ( delay >= 0 && ( delay + System.currentTimeMillis() >= 0)));
       @ signals ( IllegalArgumentException ex )
       @   ex.getMessage() != null &&
       @   ( ( delay < 0 || ( delay + System.currentTimeMillis() < 0)));
    @*/
```

```

public void schedule(TimerTask task, long delay);

/*@ behaviour
  @ requires ( task instanceof TimerTask && ( task != null)) &&
  @   ( ( time.getTime() >= 0 && ( time instanceof Date)));
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && ( time.getTime() < 0 );
  @*/
public void schedule(TimerTask task, Date time);

/*@ behaviour
  @ requires ( period > 0)&&( task instanceof TimerTask &&
  @   ( task != null)) && ( ( delay >= 0 &&
  @   ( delay + System.currentTimeMillis() >= 0)));
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && ( ( period <= 0)|| ( delay < 0 ||
  @   ( delay + System.currentTimeMillis() < 0)));
  @*/
public void schedule(TimerTask task, long delay, long period);

/*@ behaviour
  @ requires (period > 0)&&( task instanceof TimerTask &&
  @   ( task != null)) && ( ( time.getTime() >= 0 &&
  @   ( time instanceof Date)));
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && ( (period <= 0)||time.getTime() < 0 );
  @*/
public void schedule(TimerTask task, Date firstTime, long period);

/*@ behaviour
  @ requires ( period > 0)&&( task instanceof TimerTask &&
  @   ( task != null)) && ( ( delay >= 0 &&
  @   ( delay + System.currentTimeMillis() >= 0)));
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && ( (period <= 0)|| ( delay < 0 ||
  @   ( delay + System.currentTimeMillis() < 0)));
  @*/
public void scheduleAtFixedRate(TimerTask task, long delay, long period);

/*@ behaviour
  @ requires (period > 0)&&( task instanceof TimerTask &&
  @   ( task != null)) && ( ( firstTime.getTime() >= 0 &&
  @   ( time instanceof Date)));
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null &&
  @   ( firstTime.getTime() < 0 || period <= 0 );
  @*/
public void scheduleAtFixedRate(TimerTask task, Date firstTime,
                                long period);

/*@ behaviour
  @ requires (period > 0)&&( task instanceof TimerTask &&
  @   ( task != null)) && ( ( time.getTime() >= 0 &&
  @   ( time instanceof Date)));
  @ signals ( IllegalArgumentException ex )
  @   ex.getMessage() != null && ( (period <= 0) ||(time.getTime() < 0) );
  @*/

```

```
private void sched(TimerTask task, long time, long period);

/*@ behaviour
  @ assignable \nothing;
  @*/
public void cancel();

}
```

Especificação TimerTask.jml

```
package java.util;

/*@ model org.jmlspecs.*;

public /*@ pure @*/ abstract class TimerTask implements Runnable {

    final Object lock = new Object();

    int state = VIRGIN;

    static final int VIRGIN = 0;

    static final int SCHEDULED = 1;

    static final int EXECUTED = 2;

    static final int CANCELLED = 3;

    long nextExecutionTime;

    long period = 0;

    /*@ behaviour
      @ ensures (\result instanceof TimerTask);
      @*/
    protected TimerTask();

    public abstract void run();

    /*@ behaviour
      @ assignable \nothing;
      @ ensures (\result == true || \result == false);
      @*/
    public boolean cancel();

    /*@ behaviour
      @ assignable \nothing;
      @ ensures (\result >= 0);
      @*/
    public long scheduledExecutionTime();

}
```

Bibliografia

- [1] MUCHOW, John W. Core J2ME – Tecnologia e MIDP. John W. Muchow. São Paulo: Pearson Makron Books, 2004.
- [2] KEOGHS, JAMES - J2ME: The Complete Reference. Osborne - McGraw-Hil, 2003.
- [3] Sun Inc. <<http://java.sun.com/j2me/index.jsp>>. Visitado pela última vez em 22/10/2005.
- [4] GOLD Russel, HAMMEL Thomas e SNYDER, Tom. Test-Driven Development: A J2EE Example. Apress Books. Novembro, 2004.
- [5] CLDC. <http://java.sun.com/products/cldc/>. Visitado pela última vez em 22/10/2005.
- [6] CDC.< <http://java.sun.com/products/cdc/index.jsp>>. Visitado pela última vez em 23/10/2005.
- [7] LEAVENS G., BAKER Albert L., e RUBY Clyde. JML: A notation for Detailed Design. In Haim Kilov, Bernahard Rumpe, e Ian Simmonds(editors), Behavioral Specifications of Businesses and Systems, capítulo 12, páginas 175-158. Copyright Kluwer,1999.
- [8] LEAVENS G., BAKER Albert L., e RUBY Clyde. Preliminary Design of JML: Behavioral Interface Specification language for Java. Department of Computer Science, Iowa State University, TR #98-06-rev27, Junho, 1998, revisado em Abril 2005.
- [9] VERZULLI, JOE. Getting started with JML: Improve your Java programs with JML annotation. IBM DeveloperWorks article, Março 2003.
- [10] LEAVENS G., e CLIFTON Curtis. Lessons from the JML Project. To appear in Verified Software: Theories, Tools, Experiments.Department of Computer Science, Iowa State University, TR #05-12a, Abril 2005, revisado em Julho de 2005.
- [11] BURDY, Lilian. et al. An Overview of JML Tools and Applications. In Thomas Arts and Wan Fokkink(editors), Eighth International Workshop on Formal Methods for Industrial Critical Systems(FMICS '03), páginas 73—89.Volume 80 of Electronic Notes in Theoretical Computer Science(ENTCS), Elsevier, Junho, 2005.
- [12] ESC/Java 2.<<http://secure.ucd.ie/products/opensource/ESCJava2/>>.Visitado pela última vez em 23/10/2005.
- [13] GOSLING, James et al.The Java Language Specification Second Edition.The Java Series. Addison-Wesley, Boston, MA, 2000.
- [14] LEAVENS, Gary and CHEON, Yoonsik. Project By Contract with JML.Novembro, 2004.Disponível em <<http://www.cs.iastate.edu/~leavens/JML/teaching.shtml>>.
- [15] B. MEYER et al. Design by Contract: The Lessons of Ariane. IEEE Computer, 30(2):129–130, January 1997.
- [16] MEYER, Bertrand. Object-Oriented Software Construction, Second Edition, Santa Barbara, Prentice Hall Professional Technical Reference, 1997.
- [17] E. M. Clarke and J. M. Wing. Formal Methods: State of the Art and Future Directions. ACM Computing Surveys, 1996.
- [18] Sun Inc.MIDP 2.0. <<http://java.sun.com/products/midp/>>. Visitado pela última vez em 22/10/2005.

- [19] Sun Inc. JSR 118.<<http://www.jcp.org/aboutJava/communityprocess/final/jsr118/>>. Visitado pela última vez em 22/10/2005.
- [20] WAP. <<http://www.faqs.org/rfcs/rfc2757.html>>. Visitado pela última vez em 22/10/2005.
- [21] Windows CE. <<http://download.microsoft.com/download/7/2/f/72fef3b0-9545-46a4-8886-a94f265df9c4/EVA-2.9-OS-CE-01-I01.pdf>>. Visitado pela última vez em 19/10/2005.
- [22] Symbian.< <http://www.symbian.com/>>. Visitado pela última vez em 19/10/2005.
- [23] KINIRY, J., CHAPLIN, P., e HURLIN, C. "Integrating Static Checking and Interactive Verification: Supporting Multiple Theories and Provers in Verification." VSTTE 2005.
- [24] LEAVENS, Gary T. e CHEON, Yoonsik., "The JML and JUnit Way of Unit Testing and its Implementation", Abril, 2004, Disponível em <<ftp://ftp.cs.iastate.edu/pub/techreports/TR04-02/TR.pdf>>.
- [25] Windows CE. <<http://download.microsoft.com/download/7/2/f/72fef3b0-9545-46a4-8886-a94f265df9c4/EVA-2.9-OS-CE-01-I01.pdf>>. Visitado pela última vez em 19/10/2005.
- [26] Sun Inc.J2ME Building Blocks for Mobile Devices:White Paper on KVM and the Connected, Limited Device Configuration(CLDC).Palo Alto, Maio de 2000.
- [27] P. KRUCHTEN. An Introduction to the Rational Unified Process.Addison-Wesley,2000.
- [28] RUMBAUGH, JAMES.UML Guia do Usuario, Editora Campus.
- [29] OMG. Unified modeling language. Specification v1.5, Object Management Group, March 2003. <http://www.omg.org/cgi-bin/doc?formal/03-03-01>.
- [30] SPIVEY, M. The Z Notation. Prentice-Hall, 1992.
- [31] WOODCOCK, J and DAVIES, J. Using Z: Specification, Refinement, and Proof. Prentice Hall, 1996.
- [32] WOODCOCK, J. C. P. and CAVALCANTI, A. L. C.. The Semantics of Circus. In D. Bert, J. P. Bowen, M. C. Henson, and K. Robinson, editors, ZB 2002: Formal Specification and Development in Z and B, volume 2272 of Lecture Notes in Computer Science, páginas . Springer-Verlag, 2002.
- [33] MORGAN, C.C.. Programming of Specifications, 2 nd Edition. Prentice Hall.1994.
- [34] LEINO, K. RUSTAN M., et al."ESC/Java User's Manual". Technical Note 2000-002, Compaq Systems Research Center, Outubro, 2000.
- [35] GAMA, Erich e BECK, Kent. JUnit <<http://www.junit.org>>. Visitado pela última vez em 09/10/2005.
- [36] LEAVENS, G. et al. JML Reference Manual. Disponível em: http://www.cs.iastate.edu/~leavens/JML/jmlrefman/jmlrefman_toc.html. Visitado pela última vez em 09/10/2005.
- [37] SUCCI, Giancarlo, MARCHESI, Michele. Extreme Programming Examined. Addison Wesley, 2001.
- [38] POLL, Erick , BERG, Joachim v. d. , e JACOBS, Bart. Specification of the JavaCard API in JML. In J. Domingo-Ferrer, D. Chan, and A. Watson, editors, Smart Card Research and Advanced Application, páginas 135-154. Kluwer Academic Publishers, 2000.
- [39] Sun Inc. API JavaCard Sun Inc. <<http://java.sun.com/products/javacard/index.jsp>>. Visitado pela última vez em 09/10/2005.
- [40] Protocolo HTTP.< <http://www.faqs.org/rfcs/rfc2616.html>>. Visitado pela última vez em 22/10/2005.
- [41] ESC/Java 2(Extended Static Checking for Java 2) Versão 2.0a8. <http://secure.ucd.ie/products/opensource/ESCJava2/download.html>. Visitado pela última vez em 24/09/2005.
- [42] e-CPF e e-CNPJ. <http://www.receita.fazenda.gov.br>. Visitado pela última vez em 22/10/2005.

- [43] ALSTERS, Michiel et al. <<http://www.verificard.org/>>. Visitado pela última vez em 09/10/2005.
- [44] Isabelle.< <http://isabelle.in.tum.de/>>. Visitado pela última vez em 24/09/2005.
- [45] JAAS . <<http://java.sun.com/products/jaas/>>. Visitado pela última vez em 22/10/2005.
- [46] BOND, Martin, et al. Aprenda J2EE em 21 dias. Pearson Education. São Paulo. 2003.