“SL Series”

Java™ Runtime Environment

（Java™実行環境）

プログラミングガイド

(第 2.10 版　2003 年 5 月 24 日)

シャープ株式会社
情報システム事業本部

改訂履歴

2003 年 5 月 24 日　**バージョン** 2.10 **リリース**

2003 年 1 月 24 日　**バージョン** 2.00 **リリース**

2002 年 7 月 25 日　**バージョン** 1.00 **リリース**

0. はじめに

0.1. SL Series の概要

0.1.1. 製品の概要

SL Series は Linux®/Java™環境のシャープ製 PDA です。SL Series では, OS に Linux® 2.4.x (Metrowerks 社の OpenPDA™、あるいは、Lineo 社の Embedix™)を採用し、Linux 上の標準的なアプリケーションの実行環境として、Trolltech 社の Qt/Embedded を採用しています。C 言語や C++言語を使って開発したアプリケーションプログラムは、SL Series のハードウェア環境にあわせてクロスコンパイルすることで利用可能となります。
SL Series はまた、ハードウェアに非依存なアプリケーションの実行環境として、Java™実行環境を搭載しています。Java™のアプリケーションソフト実行環境には Insignia Solutions 社の Jeode™あるいは Sun Microsystems 社の Java™ 2 Platform Micro Edition Connected Device Configuration, Foundation Profile and Personal Profile を標準採用しています。Java™というオープンな標準プラットフォームを採用することで、誰でも既存の容易な Java™ツールを使って SL Series 向けアプリケーションソフトウェア開発が可能となります。

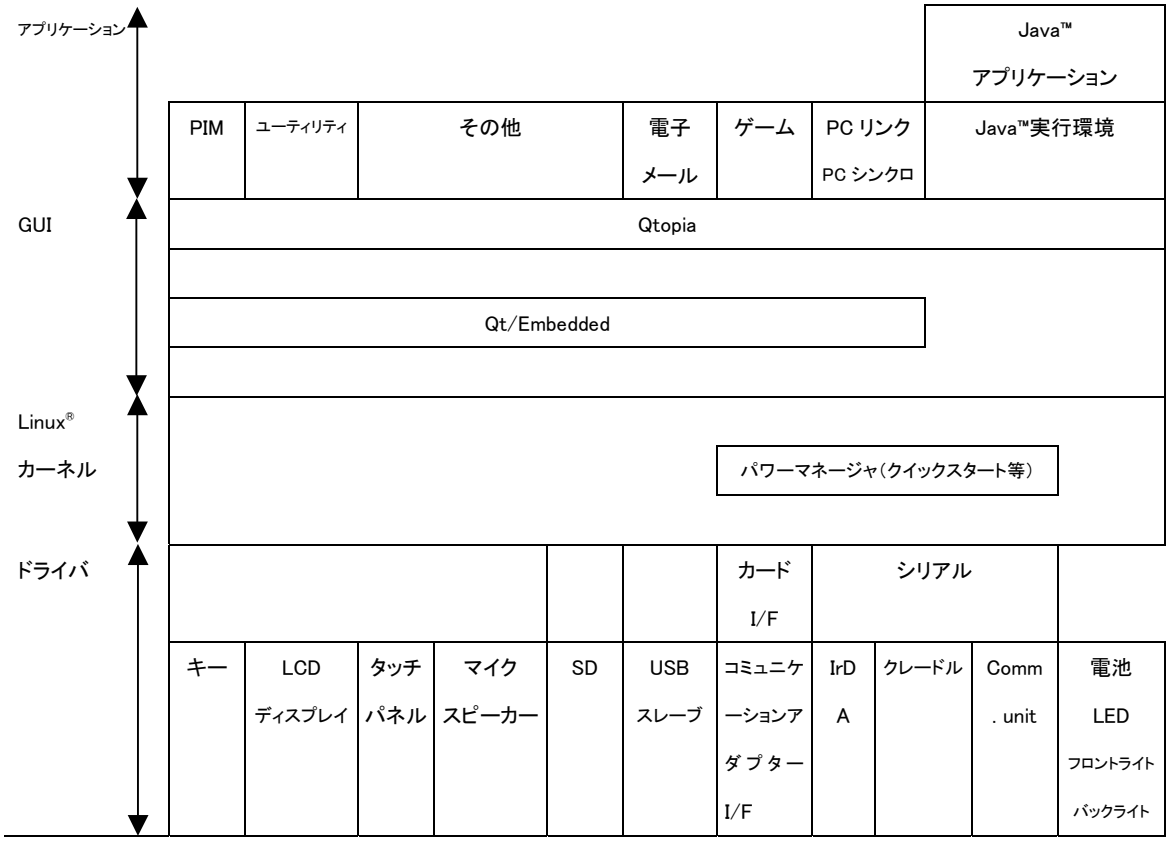
ハードウェアのシステム要件によってそれぞれ異なる Java™規格が存在することは周知のとおりです。また、Java™規格も日々進化しています。それら Java™規格の中で、SL-C700/SL-B500/SL-A300 では、高性能内蔵型モバイル機器向けの PersonalJava™バージョン 1.2 を採用しています。PersonalJava™は、PC 向けの Java 環境を組込機器向けに再設計したものであり、数多くの PC 向け資産を活用することが可能です。SL-C750/SL-C760 では、Java™ 2 Platform Micro Edition Connected Device Configuration, Foundation Profile and Personal Profile(以下、J2ME™ Personal Profile と呼ぶ)を採用しています。J2ME™ Personal Profile は、PersonalJava™の後継にあたる最新モバイル機器向けの Java™規格です。J2ME™ Personal Profile では、決められたルールに従ってアプリケーション開発を行うことで、PersonalJava™との互換性を確保することが可能です。また、SL-C700/SL-B500 シリーズでは、Sun Microsystems 社のサイトで公開されている SL series 向け J2ME™ Personal Profile の early access 版を動作させることも可能です。
SL Series では、今後は最新の J2ME™ Personal Profile を標準としてサポートしていきます。

0.1.2. アーキテクチャの概要

SL Series アーキテクチャの概要を以下の図に示します。SL Series は Linux® 2.4.x 上の Java™実行環境を採用しています。

Linux®では、SL Series は Busy Box（http://www.busybox.net/）のコマンドをサポートします。さらに（一部機種ではオプションを購入することにより）ネットワークまたは Compact Flash カード、nfs 等の Linux®ツールの使用も可能です。

PersonalJava™ 1.2 および J2ME™ Personal Profile の実行環境は、Qt/Embedded および Qtopia 上のひとつのアプリケーションとして動作します。PersonalJava™ 1.2 および J2ME™ Personal Profile 上で開発されたアプリケーションは、SL Series 上では「そのまま」実行されますが、開発者は SL Series のスクリーンサイズにフィットするようにユーザーインターフェースの調整を必要とする場合があります。



0.2. 本プログラミングガイドの対象読者

本書は Java™プログラミング言語を使い、PersonalJava™開発環境もしくは J2ME™ Personal Profile 開発環境下で SL Series 向けアプリケーションとアプレットの開発を希望するユーザー向けに発行するものです。本書の読者は以下の要件を満たすものと仮定します。

- Java™プログラミング技術
- PersonalJava™, J2ME™ Platform, Connected Device Configuration (CDC), Foundation Profile, Personal Profile の仕様
- SL Series PDA に関する基本的知識

必要に応じて市販の Java™プログラマ向けガイドブックを参照することを推奨します。

0.3. ガイドブックの概要

1 は PersonalJava™と J2ME™ Personal Profile を簡単に紹介し、その互換性を説明します。

2 は Java™プログラムの開発を支援する PersonalJava™向け各種ツールを紹介し、これらのツールを使用して、SL Series 用 Java™プログラム開発の基本的開発ワークフローを説明します。

3 では、SL Series PersonalJava™ Runtime Environment 上でのアプリケーション作成方法の例を紹介します。簡単なアプリケーションとアプレットの作成方法を紹介し、GUI 作成を説明するソースコードや外部ファイル入出力等についても説明します。

0.4. Java™アプリケーション開発に必要な環境

SL Series で動作する PersonalJava™および J2ME™ Personal Profile アプリケーションとアプレットを開発するためには、Java™の開発ツールを入手する必要があります。PersonalJava™環境は JDK 1.1 準拠ですので、JDK 1.1 またはその上位を使用する必要があります。同様に J2ME™ Personal Profile 環境は Java™ 2 Platform, Standard Edition version 1.3.1 準拠ですので、J2SDK™ 1.3.1 またはその上位を使用する必要があります。
SL Series 向け Java™アプリケーション開発では最新の開発環境を使用することを推奨します。本書記載時点での最新版は J2SDK™ 1.4.1_01 です。システム要件は以下の通りです。

Microsoft Windows 版は、Intel ハードウェア上で動作する、Microsoft Windows 98 (Second Edition を含む)、Windows NT Workstation 4.0 (Service Pack 5 以降を適用)、Windows Me、Windows XP、Windows 2000 (Service Pack 2 以降を適用) オペレーティングシステムで使用できます。Pentium 166MHz 以上のプロセッサが必要であり、GUI アプリケーションの実行には、32MB 以上の RAM が必要です。J2SDK™ソフトウェアをインストールするには、120MB の空きディスク容量が必要です。

また、Linux 版は、Linux カーネル v 2.2.12 および glibc v2.1.2-11 以降が動作する Intel Pentium プラットフォームでサポートされます。32MB 以上の RAM が必要です。また、48MB の RAM、16 ビットカラーモード、ローカルホストに設定したディスプレイとともに使用する KDE または Gnome デスクトップが推奨されます。この J2SDK™ソフトウェアをインストールするには、75M バイトの空きディスク容量が必要です。

0.5. 参照 URL

0.5.1. PersonalJava™関連

- **PersonalJava™公式ウェブサイト URL:**
  http://java.sun.com/products/personaljava/

- **PersonalJava™ API 仕様:**
  PersonalJava™ API 仕様は以下のサイトからダウンロードできます。
  http://java.sun.com/products/personaljava/

SL Series PersonalJava™ Runtime Environment は PersonalJava™バージョン 1.2 に準拠します。

- **PersonalJava™技術白書:**
  PersonalJava™技術白書は PDF フォーマットで以下のサイトからダウンロードできます。
  http://java.sun.com/products/personaljava/pj_white.pdf

- **PersonalJava™エミュレーション環境:**
  Solaris®と Windows®のエミュレーション環境は以下のサイトからダウンロードできます。
  http://java.sun.com/products/personaljava/pj-emulation.html

PersonalJava™アプリケーション環境をエミュレートするには、プラットフォーム(Windows®/x86 または Solaris/Sparc)を選択し、Build Configuration で"max"を選びます("Look & Feel"が Win32 または Motif のどちらかに自動的に選択される)。

- **JavaCheck™:**
  JavaCheck™は文字通り Java™コードをチェック分析してコードが仕様(例えば PersonalJava™)に準拠しているかどうかを確認します。本ソフトウェアは以下のサイトからダウンロードできます。
  http://java.sun.com/products/personaljava/javacheck.html

バイナリコードライセンスをチェックし、合意事項に合意した後に JavaCheck™3.0 をダウンロードします。

- **PersonalJava™ FAQ：**

  PersonalJava™ FAQ は以下の URL で閲覧することができます。

  http://java.sun.com/products/personaljava/faq.html

0.5.2. J2ME™ Personal Profile 関連

- **J2ME™ Personal Profile 公式ウェブサイト URL：**

  http://java.sun.com/products/personalprofile/

- **J2ME™ Personal Profile Programmer's Guide：**

  J2ME™ Personal Profile Programmer's Guide は以下のサイトからダウンロードできます。

  http://java.sun.com/j2me/docs/pdf/PP_Programmer_Guide.pdf

- **SL Series 向け J2ME™ Personal Profile アプリケーション開発チュートリアル：**

  http://wireless.java.sun.com/personal/articles/ztutorial/

- **J2ME™ Connected Device Configuration and Foundation Profile 技術白書：**

  J2ME™ Connected Device Configuration and Foundation Profile 技術白書は PDF フォーマットで以下のサイトからダウンロードできます。

  http://java.sun.com/products/cdc/wp/CDCwp.pdf

0.5.3. その他

- **Sun Microsystems 社 SL Series 向け J2ME™ Personal Profile の early access 版ダウンロードサイト URL:**

  Java Developer Connection (JDC)に User ID と Password を登録した後に、以下のサイトに User ID と Password を入力することで、SL Series 向け J2ME™ Personal Profile の early access 版がダウンロードできます。

  http://developer.java.sun.com/developer/earlyAccess/pp4zaurus

- **Java™ 2 Platform, Standard Edition v1.4:**

  SL Series 向け Java™アプリケーションを開発するための推奨環境です。

  http://java.sun.com/j2se/1.4.1/ja/download.html

- **ザウルス宝箱 Pro SL シリーズ技術サポートページ:**

  国内向け SL Series ザウルスのソフト開発者のためのページです。開発ツールの情報やザウルスのハードウェア/ソフトウェア技術情報、その他開発のための参考情報を提供しています。

  http://more.sbc.co.jp/sl_j/sl_top.asp

目次

1. SL Series Java™ Runtime Environment の概要

SL series ではこれまで, PersonalJava™ Application Environment(以下 PJAE と呼ぶ)を採用してきました. PJAE は、PDA を始めとする組込機器で Java™ベースのアプリケーションを実行できる Java™アプリケーション実行環境です。J2SDK™や JRE™がパーソナルコンピュータ(PC)やワークステーション(WS)向けに Java™ベースのソフトウェア開発環境と実行環境を提供するのに対し、PJAE はセットトップボックスや WebPhone、SmartPhone、高性能 PDA、その他ネットワーク機能内蔵機器に対して、Java™ベースのアプリケーション開発環境と実行環境を提供します。

PDA を始めとするこれら小型機器は、PC や WS と比較するとメモリ容量が少なく、CPU 処理サイクルも遅いために、ハードウェアに限界があります。さらにこれら機器のディスプレイも小型であるために、グラフィカルユーザーインターフェース(GUI) の開発にあっても制限を与えているかもしれません。つまり、これらポータブル機器に J2SDK™や JRE™を使うのは適切とは言えず、小型機器に最適な Java™の開発環境と実行環境を提供する必要があります。
SL Series では, 上記に加え, プログラマに Java™開発環境を提供する一方で、上記ポータブルネットワーク機器の機能向上を図ることを理由として, PersonalJava™を採用してきました.

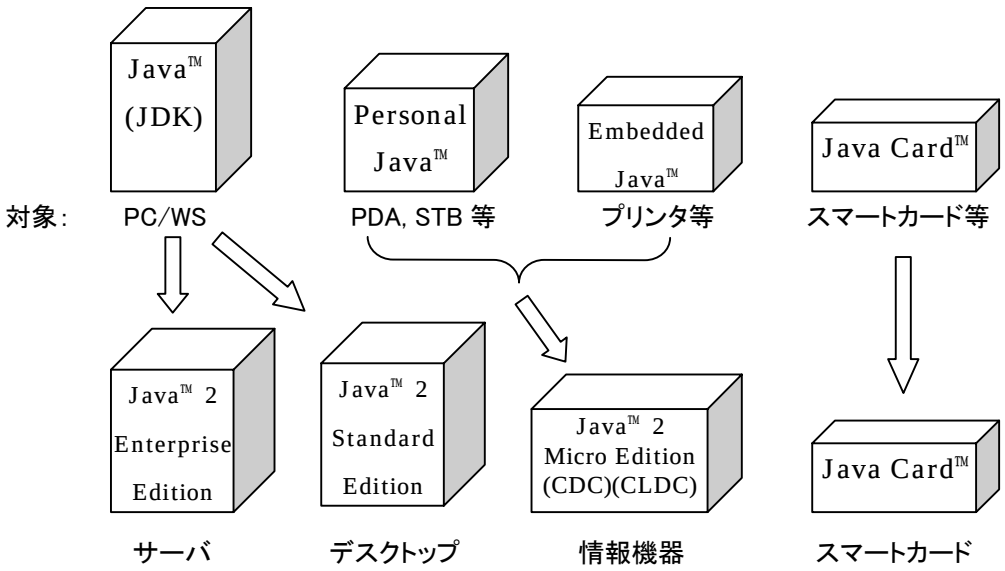


図 1-1:Java™アプリケーション環境内の様々な対象デバイス

Java™フレームワークは Java™ 2 の登場以来変貌を遂げています。それは、サーバ向けの Java™ 2 Platform, Enterprise Edition (J2EE™)、デスクトップ/クライアント PC 向けの Java™ 2 Platform,

Standard Edition (J2SE™)、内蔵型機器向けの Java™ 2 Platform, Micro Edition (J2ME™)、そして Java Card™といったように、Java™実行環境の名前が種々あることからも明らかです。J2EE™と J2SE™はすでに多くのサーバや PC に採用されています。

PersonalJava™も J2ME™の一部として統合され、Connected Device Configuration (CDC)と Foundation Profile、Personal Basis Profile、Personal Profile、RMI Optional Package 等として再構成されています。SL series では, この PersonalJava™の J2ME™における再構成に伴い, 最新の Java 環境を提供する意味も含め, J2ME™ Personal Profile を採用しました.

なお, これら新しい構成やプロファイルとは別に、J2ME™は Connected Limited Device Configuration (CLDC)、Mobile Information Device (MID) Profile、PDA Profile があります. これらのテクノロジにより, J2ME™では, 低価格機器やローエンドの携帯情報端末、携帯電話などをサポートしています。

1.1. PersonalJava™の簡単な説明

これまでの SL Series（SL-A300, SL-B500, SL-C700）では Java™のアプリケーション実行環境として、PersonalJava™実行環境である Insignia Solutions 社の Jeode™を採用しています。ここではPersonalJava™実行環境について簡単に説明します。

PJAE 仕様は PersonalJava™ API に準拠しており、仕様面では JDK™や JRE™とは異なるものです。PersonalJava™ API は Java™ API をベースに作成されていますが、PJAE は対象機器グループごとに分類されています。PersonalJava™ Virtual Machine は、標準 Java™ Virtual Machine (JVM)のコンパクト版であり、メモリ容量の小さい携帯機器向けに使用されます(JVM と PersonalJava™ Virtual Machine は共に同じ Java™言語と Virtual Machine 仕様を採用しますが、CLDC のサブセットは大きく改造されています)。PJAE は、PersonalJava™ Virtual Machine と PersonalJava™ API 両方をサポートするクラスライブラリを持っています。各 PJAE はそれぞれの対象機器ごとに専用のインストール方法を採用しますが、同じ API コードが使用されている限り、アプレットとアプリケーションの PJAE 機器間互換性は保証されています。しかし、オプション API や機器特有のライブラリと API、または独自の Java™アプレットとアプリケーションを使う場合には、互換性は保証されません。

1.1.1. PersonalJava™パッケージリスト

以下のパッケージが PersonalJava™には含まれます。

PersonalJava™		
java.applet	java.io	java.security.spec
java.awt	java.lang	java.text
java.awt.datatransfer	java.lang.reflect	java.text.resources
java.awt.event	java.net	java.util
java.awt.image	java.security	java.util.jar
java.peer	java.security.cert	java.util.zip
java.beans	java.security.interfaces	

1.1.2. Jeode™がサポートするオプショナルパッケージリスト

以下のパッケージは PersonalJava™に含まれるオプショナルパッケージです。Jeode™にはすべてのオプショナルパッケージが含まれます。

PersonalJava™ Optional Package		
java.math	java.rmi.dgc	java.rmi.server
java.rmi	java.rmi.registry	java.sql

1.2. J2ME™ Personal Profile の簡単な説明

SL-C750/SL-C760 では，PersonalJava™の再構成の流れをいち早く取り入れ最新の Java™環境を提供するために，J2ME™ Personal Profile を採用しています． また，これまでの SL series 向けには，Sun Microsystems 社のサイトで J2ME™ Personal Profile の early access 版が公開されています。ここでは J2ME™ Personal Profile について簡単に説明します。

J2ME™ Personal Profile は、ネットワークに接続する機器向けの J2ME™アプリケーション実行環境です。JDK™ 1.1 の Abstract Window Toolkit (AWT) を完全にサポートしており、さらに J2SE™で導入された多くの特徴的な改善も含んでいます。J2ME™ Personal Profile は一般消費者向け電気機器用に設計されたアプリケーション開発環境です。J2ME™ Personal Profile は PersonalJava™環境の多くの機能を提供し、それらは次世代の J2ME™のプロファイルにも含まれます。

J2ME™ Personal Profile 環境は PersonalJava™環境と主に以下の点が異なります。

- J2ME™ Connected Device Configuration (CDC) と Foundation Profile から deprecated のメソッドが削除されました。
  J2ME™ Personal Profile はこれらのソフトウェアの上で動作するため、Java™のパッケージ、クラス、メソッドは小さくなっています。
- “optional” の機能はありません。
  J2ME™ Personal Profile の簡素化された API は必要のない機能が含まれていないため、互換性のあるアプリケーションを簡単に書くことができます。

PersonalJava™環境と J2ME™ Personal Profile 環境の詳細な比較は **J2ME™ Personal Profile Programmer's Guide** を参考にしてください。

1.2.1. J2ME™ Personal Profile パッケージリスト

以下のパッケージが J2ME™ Personal Profile には含まれます。

J2ME™ Personal Profile		
java.applet	java.lang.ref	java.security.interfaces
java.awt	java.lang.reflect	java.security.spec
java.awt.color	java.math	java.text
java.awt.datatransfer	java.net	java.util
java.awt.event	java.rmi	java.util.jar
java.awt.image	java.rmi.registry	java.util.zip
java.beans	java.security	javax.microedition.io
java.io	java.security.acl	javax.microedition.xlet
java.lang	java.security.cert	javax.microedition.xlet.ixc

1.2.2. J2ME™ RMI オプショナルパッケージリスト

以下のパッケージは J2ME™に含まれる RMI オプショナルパッケージです。

J2ME™ RMI Optional Package		
java.rmi	java.rmi.dgc	java.rmi.server
java.rmi.activation	java.rmi.registry	

1.3. PersonalJava™と J2ME™ Personal Profile の互換性

J2ME™ Personal Profile は PersonalJava™と多くの部分で互換性があります。ただし、1.2 で示したように異なる部分も存在します。

SL Series 向け Java™アプリケーションやアプレットを作成する際に、PersonalJava™実行環境でも J2ME™ Personal Profile 実行環境でも動作するようにするためには、1.3.1 に示すメソッドを使用しないようにする必要があります。

1.3.1. J2ME™ Personal Profile 環境ではサポートされない PersonalJava™環境の API 一覧

以下のメソッドは J2ME™ Personal Profile では存在しないものです。

Classes
java.awt.PrintGraphics
java.awt.PrintJob
java.beans.BeanDescriptor
java.beans.BeanInfo
java.beans.Customizer
java.beans.EventSetDescriptor
java.beans.FeatureDescriptor
java.beans.IndexedPropertyDescriptor
java.beans.IntrospectionException
java.beans.Introspector
java.beans.MethodDescriptor
java.beans.ParameterDescriptor
java.beans.PropertyDescriptor
java.beans.PropertyEditor
java.beans.PropertyEditorManager
java.beans.PropertyEditorSupport
java.beans.SimpleBeanInfo
java.io.LineNumberInputStream
java.io.StringBufferInputStream

Fields
java.awt.GridBagLayoutInfo.layoutInfo
java.net.HttpURLConnection.HTTP_SERVER_ERROR
java.lang.SecurityManager.inCheck

Methods	
java.awt.peer 関連のメソッド	Toolkit.createXXX ( ), Component.getPeer ( ) など
java.awt.GridBagLayout	GetLayoutInfo ( )
java.bean.Beans	getInstanceOf (Object, Class)
	isInstanceOf (Object, Class)
	setDesignTime (boolean)
	setGuiAvailable (boolean)
java.io.ByteArrayOutputStream	toString (int)
java.lang.Character	isJavaLetter (char)
	isJavaLetterOrDigit (char)
	isSpace (char)
java.lang.ClassLoader	defineClass (byte[ ], int, int)
java.lang.Runtime	getLocalizedInputStream (InputStream)
	getLocalizedOutputStream (OutputStream)
	runFinalizersOnExit (boolean)
java.lang.SecurityManager	classDepth (String)
	classLoaderDepth ( )
	currentClassLoader ( )
	currentLoadedClass ( )
	getInCheck ( )
	inClass (String)
	inClassLoader ( )
java.lang.String	getBytes (int, int, byte[ ], int)
java.lang.System	getenv (String)
	runFinalizersOnExit (boolean)
java.lang.Thread	countStackFrames ( )
	resume ( )
	stop ( )
	stop (Throwable)
	suspend ( )

java.lang.ThreadGroup	allowThreadSuspension (boolean)
	resume ( )
	stop ( )
	suspend ( )
java.net.DatagramSocketImpl	getTTL ( )
	setTTL (byte)
java.net.MulticastSocket	getTTL ( )
	setTTL (byte)
java.net.URLConnection	getDefaultRequestProperty (String)
	setDefaultRequestProperty (String, String)
java.net.URLStreamHandler	setURL (URL, String, String, int, String, String)
java.security.Security	getAlgorithmProperty (String, String)
java.security.Signature	getParameter (String)
	setParameter (String, Object)
java.security.SignatureSpi	engineGetParameter (String)
	engineSetParameter (String, Object)
java.util.Date	getDate ( )
	getDay ( )
	getHours ( )
	getMinutes ( )
	getMonth ( )
	getSeconds ( )
	getTimezoneOffset ( )
	getYear ( )
	parse (String)
	setDate (int)
	setHours (int)
	setMinutes (int)
	setMonth (int)
	setSeconds (int)
	setYear (int)
	toGMTString ( )
	toLocaleString ( )
	UTC (int, int, int, int, int, int)

Deprecated Constructors	
java.io.StreamTokenizer	StreamTokenizer (InputStream)
java.lang.String	String (byte[ ], int)
	String (byte[ ], int, int, int)
java.net.Socket	Socket (InetAddress, int, boolean)
	Socket (String, int, boolean)
java.util.Date	Date (int, int, int)
	Date (int, int, int, int, int)
	Date (int, int, int, int, int, int)
	Date (String)

1.4. SL Series Java™ Runtime Environment における制限事項

SL Series ではアプリケーション実行時に使用できるメモリ容量が制限されています。Java™アプリケーション実行時には Java™ VM 自体もメモリ上に存在するため、Java™アプリケーション自体が使用できるメモリはさらに制限されます。

また、画面デザインにおいても注意が必要です。SL Series では画面下部にタスクバーが常駐しており、さらにそれが最前面に表示されるため、フルスクリーンの Java™アプリケーションおよびアプレットを作成したときには下端が表示されないという問題が発生します。この問題を回避するにはあらかじめタスクバーの高さを引いたサイズのフレームを使用する必要があります。このテンプレートサンプルソースは 4.1 を参照してください。

2. SL Series 向け Java™アプリケーション開発の概要

2.1. アプリケーション開発ツール

数多くの Java™の開発ツールは、様々な開発ツールベンダのウェブサイトから有料あるいは無料で入手することができます。本章では、PersonalJava™および J2ME™ Personal Profile アプリケーションとアプレットを開発する上での基本ツールを、以下のセクションで紹介し説明いたします。ここでは、J2SDK™, Standard Edition 1.4.1_01 の例を示しますので、ご自分の開発環境に合わせて環境変数やコマンド名などを読み替えてください。

2.1.1. Java™コンパイラ

Java™コンパイラは Java™ソースファイルをコンパイルして class ファイルを作成するツールです。各 J2SDK™ディストリビューションに含まれています。Java™コンパイラを呼び出すには、コマンドラインの“javac”をご使用ください。PersonalJava™環境は JDK 1.1 準拠ですので、SL Series 向け PersonalJava™アプリケーション開発には、オプションに“-target 1.1”を指定してください。

```
Usage: javac <options> <source files>
where possible options include:
  -g                    Generate all debugging info
  -g:none               Generate no debugging info
  -g:{lines,vars,source}   Generate only some debugging info
  -nowarn               Generate no warnings
  -verbose              Output messages about what the compiler is doing
  -deprecation          Output source locations where deprecated APIs are used
  -classpath <path>       Specify where to find user class files
  -sourcepath <path>      Specify where to find input source files
  -bootclasspath <path>    Override location of bootstrap class files
  -extdirs <dirs>         Override location of installed extensions
  -d <directory>          Specify where to place generated class files
  -encoding <encoding>     Specify character encoding used by source files
  -source <release>       Provide source compatibility with specified release
  -target <release>       Generate class files for specific VM version
  -help                 Print a synopsis of standard options
```

2.1.2. インタプリタ

コンパイルされたクラスオブジェクトは、Java™のインタプリタである Java™ Virtual Machine (JVM) というソフトウェア上で実行されます。インタプリタは J2SDK™、JRE™、PJEE などの Java™開発ソフトウェアディストリビューション内に含まれています。インタプリタは、JRE™内では”Java”、PJEE 内では”pJava”と呼ばれています。これらは実行可能なコマンドラインです。SL Series 向け PersonalJava™アプリケーションの開発には、J2SDK™ 1.2.x (1.2.2 またはその上位)あるいは PJEE 3.1 (最上位バージョン)をご使用ください。

```
Usage: java [-options] class [args...]
         (to execute a class)
   or  java -jar [-options] jarfile [args...]
         (to execute a jar file)

where options include:
    -client    to select the "client" VM
    -server       to select the "server" VM
    -hotspot       is a synonym for the "client" VM  [deprecated]
                The default VM is client.

    -cp -classpath <directories and zip/jar files separated by :>
                set search path for application classes and resources
    -D<name>=<value>
                set a system property
    -verbose[:class|gc|jni]
                enable verbose output
    -version     print product version and exit
    -showversion  print product version and continue
    -? -help     print this help message
    -X          print help on non-standard options
    -ea[:<packagename>...|:<classname>]
    -enableassertions[:<packagename>...|:<classname>]
                enable assertions
    -da[:<packagename>...|:<classname>]
    -disableassertions[:<packagename>...|:<classname>]
                disable assertions
    -esa | -enablesystemassertions
                enable system assertions
    -dsa | -disablesystemassertions
                disable system assertions
```

```
usage: pjava [-options] class

where options include:
    -help           print out this message
    -version         print out the build version
    -v -verbose      turn on verbose mode
    -debug          enable remote JAVA debugging
    -noasyncgc       no effect.  Asynchronous GC support was removed.
    -verbosegc       print a message when garbage collection occurs
    -noclassgc       disable class garbage collection
    -ss<number>      set the maximum native stack size for any thread
    -oss<number>     set the maximum Java stack size for any thread
    -ms<number>      set the initial Java heap size
    -mx<number>      set the maximum Java heap size
    -mr<number>      set the red heap reserve size
    -my<number>      set the yellow heap reserve size
    -classpath <directories separated by semicolons>
                   list directories in which to look for application classes
    -bootclasspath <directories separated by semicolons>
                   list directories in which to look for system classes
    -Xrun<library>[:<option>=<value>, ...]
                    load library on startup
    -verify         verify all classes when read in
    -verifyremote    verify classes read in over the network [default]
    -noverify       do not verify any class
```

2.1.3. ランタイム

JVM の Java™ Runtime Environment（JRE™）は JDK™ 1.2.x 以降インタプリタと統合されました。JDK™ 1.1.x ではコマンドラインから呼び出すことができます。JRE™用には標準クラスライブラリへのパスを指定する必要はありません。

JRE™はデフォルトで JDK™ディストリビューションに含まれていますが、ダウンロードしてインストールすることもできます。JRE™ はコマンドラインの“jre”で呼び出すことができます。

```
Java(tm) Runtime Loader Version 1.1.8_006
Usage: jre [-options] classname [arguments]
Options:
 -?, -help    print out this message
 -v, -verbose  turn on verbose mode
 -verbosegc   print a message when garbage collection occurs
 -noasyncgc   disable asynchronous garbage collection
 -noclassgc   disable class garbage collection
 -ss<number>   set the maximum native stack size for any thread
 -oss<number>  set the maximum Java stack size for any thread
 -ms<number>   set the initial Java heap size
 -mx<number>   set the maximum Java heap size
 -D<name>=<value> set a system property
 -classpath <path>     set class path to <path>
 -cp <path>   prepend <path> to base class path
 -verify     verify all classes when loaded
 -verifyremote  verify classes loaded from the network (default)
 -noverify    do not verify any classes
 -nojit     disable JIT compiler
```

2.1.4. Java™アーカイビングツール

Java™アーカイビングツール（JAR）は複数のファイル(class ファイル、データファイルなど)を整理してアーカイブファイル(jar ファイル)を作成するツールです。各 J2SDK™に含まれています。J2SDK™ バージョン 1.2.2 またはその上位をお使いください。JAR ツール内でディレクトリを指定する場合は、そのディレクトリ内の全てのファイルが 1 つのファイルにアーカイブとしてまとめられます。ディレクトリ階層はアーカイブに反映されます。

```
Usage: jar {ctxu}[vfm0Mi] [jar-file] [manifest-file] [-C dir] files ...
Options:
    -c  create new archive
    -t  list table of contents for archive
    -x  extract named (or all) files from archive
    -u  update existing archive
    -v  generate verbose output on standard output
    -f  specify archive file name
    -m  include manifest information from specified manifest file
    -0  store only; use no ZIP compression
    -M  do not create a manifest file for the entries
    -i  generate index information for the specified jar files
    -C  change to the specified directory and include the following file
If any file is a directory then it is processed recursively.
The manifest file name and the archive file name needs to be specified
in the same order the 'm' and 'f' flags are specified.

Example 1: to archive two class files into an archive called classes.jar:
      jar cvf classes.jar Foo.class Bar.class
Example 2: use an existing manifest file 'mymanifest' and archive all the
          files in the foo/ directory into 'classes.jar':
      jar cvfm classes.jar mymanifest -C foo/ .
```

（注）ZIP ファイルをアーカイブファイルにすることもできます。ただし、圧縮を防ぐためにコマンドラインオプションに”-o”を使用してください。

2.1.5. JavaCheck™

JavaCheck™は PersonalJava™環境でクラスやメソッドが仕様を満たしているか否かを確認するツールです。J2ME™ Personal Profile 環境では必要ありません。PJAE 3.0.2 向けの最適バージョンは JavaCheck™ 3.0 です。PersonalJava™ 1.2 仕様との互換性を確認するには、pJava_1.1.0.spc という指定ファイルをご使用ください。本ファイルは同じ仕様内のウェブサイトからダウンロードすることができます。

JavaCheck™は、コンパイルされたクラスを分析して、変更やオプション設定のなされたクラスを探し出して表示してくれます。本ツールは、インストール環境を比較することで、コンパイルされたプログラムが対象機器で動作するかどうかを判定することができます。JavaCheck™は以下のように呼び出してください。

J2SDK™での実行例

```
$> java -cp D:/JavaCheck3.0/bin/JavaCheck.jar JavaCheckV
```

JRE™での実行例（GUI 付きバージョン）

```
$> jre -cp D:/JavaCheck3.0/bin/JavaCheck.jar JavaCheckV
```

図 2-1:“pjava_1.1.0.spc” 指定ファイルを選択

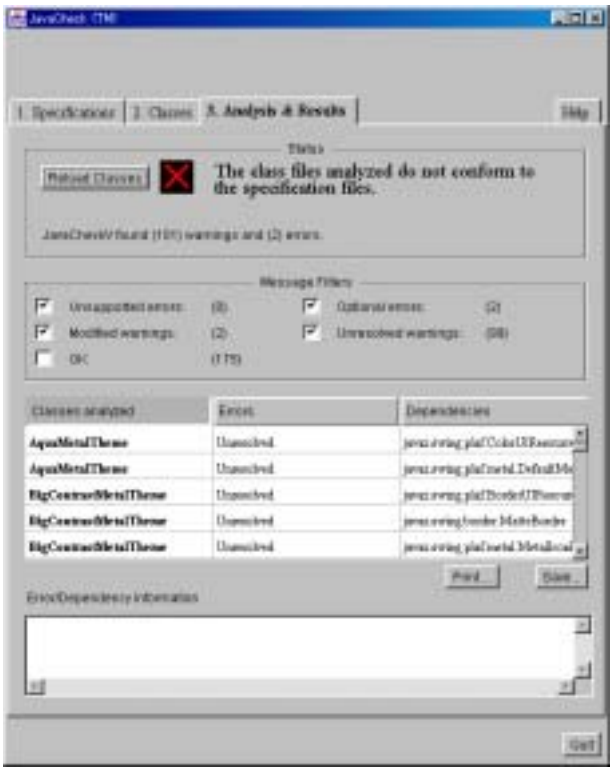


図 2-2：分析結果例

2.2. Java™アプリケーションの開発ワークフロー

下記の図 2-3 に簡単なソフトウェア開発ワークフローを示します。ワークフローの詳細については、第 3 章で説明します。本章では概要を説明します。

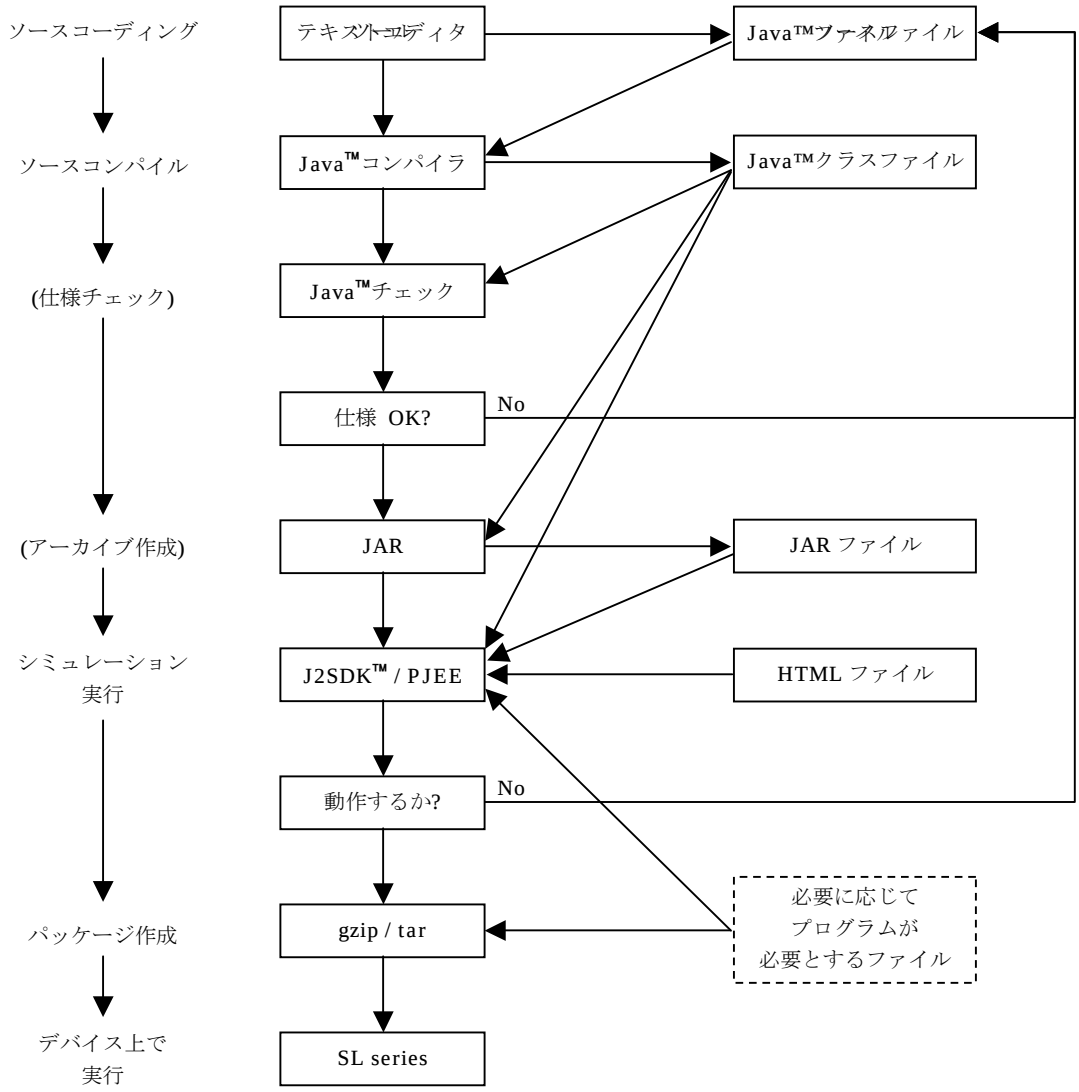


図 2-3:開発フロー例

class ファイルを作成するまでは、市販のいずれの Java™開発ツールを使用してもかまいません（例えば、米国 Sun Microsystems 社の Forte、Metrowerks 社の Codewarrior、Borland 社の JBuilder など）。ただし、これら開発ツールが Java™ 1.2.x またはその上位と互換性のあることを必ず確認してください。ソフトウェアが PersonalJava™もしくは J2ME™ Personal Profile 上で正しく動作するには、ソースファイルに若干の修正を加える必要があるかもしれません。

2.2.1. Java™ プログラムソースコードの書き込み

最初に Java™ソースファイルを作成してください。ただし、ソースファイルを Java™で書く以前に、これから書こうとするソフトウェアの仕様とクラスおよび方法について決める必要がありますが、これらトピックは本章ではカバーされておりません。ソースコードを書く前に、本プログラミングガイドとその他の参考マニュアルを読み、PJAE および J2ME™ Personal Profile の仕様と制限について理解する必要があります。

2.2.2. API 仕様の確認

Java™ API(例えば J2SE™や J2SDK™)と PersonalJava™ API および J2ME™ Personal Profile API はいくつかの部分で相違があります。また、PersonalJava™ API 仕様内でも、パッケージオプションがあります。従って、自分が使用する API が SL Series にも使用することができるか、またこれから書き込むソフトウェアの仕様が API 仕様と合致するかどうかについて調べてください。

2.2.2.1. 2.2.2.1 PC とワークステーション上の PJEE および J2SE™

本ソフトウェアは SL Series 上で動作させるものですが、これからソースコードを書こうとする PC またはワークステーションでもデバッグができるような環境作りをすることが大切です。確かに Java™はクロスプラットフォームが可能ですが、プログラムが思うように動作しないこともあります。さらに、PersonalJava™および J2ME™ Personal Profile を使って作業をする場合でも、ハードウェアのプラットフォームによって使い方が若干異なります。また、SL Series 内であっても、PersonalJava™ Runtime Environment および J2ME™ Personal Profile Runtime Environment にはその使用できるデバイスに制限があります。従って、バージョン別に設定してデバッグ環境を作成することで、PJEE および J2SE™を使ったデバッグを効果的に行うことが可能になります。

2.2.3. ソースコードのコンパイル(class ファイルの作成)

Java™コンパイラを使用して Java™ソースファイルから class ファイルを作成します。PersonalJava™の場合、専用のコンパイラは存在しないので、J2SDK™ (1.2.2 またはその上位)と互換性のあるコンパイラを使用してください。

開発したアプリケーションが PersonalJava™実行環境もしくは J2ME™ Personal Profile 実行環境で動作することを確認してください。仕様に関しての問題がない場合は、バグの原因は SL Series Java™ Runtime Environment にある可能性が高いので、SL Series Java™ Runtime Environment

と制限事項が、使用している Java™ API 仕様と不一致を起こしていないかについて再度確認してください。

2.2.3.1. UnsupportedOperationException

サポートされていないオプション API が PersonalJava™実行環境および J2ME™ Personal Profile 実行環境内に呼ばれると、この例外が 発生します。

2.2.3.2. NoClassDefFoundError

サポートされていないパッケージまたはクラスが呼び出されるとこの例外が発生します。使用している PersonalJava™オプションパッケージあるいはオプションクラスが、SL Series のサポートするものであることを確認してください。

2.2.4. JavaCheck™

ソースファイルを全てコンパイルしたら、全ての class ファイルを動作させることができます。

ただし、PersonalJava™アプリケーションではクラスファイルが PersonalJava™仕様と互換性があるかどうかを確認することをお勧めします。JavaCheck™を使ってクラスファイルを分析チェックしてください。JavaCheck™は下記のウェブサイトからダウンロードすることができます。

http://java.sun.com/products/personaljava/javacheck.html

ダウンロードする際は、PJAE 1.1.x に準拠する SL Series PersonalJava™ Runtime Environment パッケージ用の JavaCheck™ バージョン 3.0 であることを確認してください。
ここで、同じ URL 内にある pjava_1.1.0.spc という名前のファイルもダウンロードすることもできます。JavaCheck™内の本指定ファイルを使って、新しく作成したソフトウェアが PJAE 1.1.x 仕様と互換性があるかどうかを確認してください。

これらのツールを使って class ファイルを確認してください。JavaCheck™からエラーメッセージや警告が戻ってこない場合は、これらファイルが PJAE 1.1.x 仕様に完全に準拠しているので、PersonalJava™ (3.1) 環境で障害なく動作させることができます。もしエラーメッセージや警告が出たら、サポートされていないパッケージやクラス、方法が含まれている可能性があります。PersonalJava™仕様以外のプラットフォーム指定 API が見つかった場合も、JavaCheck™はエラーメッセージを表示します。

JavaCheck™からのエラー情報と依存に関する情報、および本ガイドブックの内容を参照して、APIが SL Series PersonalJava™ Runtime Environment 内で使用できるかどうかについて再度確認してください。もし必要があれば適切なファイルを修正してください。

2.2.5. J2SDK™/PJEE での class ファイルの動作;テストラン

プログラムを動作してみてください。PersonalJava™指定の API が使われていなければ、プログラムは J2SDK™（1.2.2 またはその上位）上で動作するはずです。しかし、PersonalJava™アプリケーションでは SL Series 上での動作を確認する意味でも、現段階では PJEE3.1 環境内で動作させるほうが適切です。

2.2.6. SL Series 上でのアプリケーションのテストラン

最後に、プログラムとデータを SL Series にコピーして機器上でのテストランを行います。

Java™プログラムは基本的にはプラットフォームに依存しないので、開発時に PJEE 3.1 上で動作していたプログラムなら SL Series PersonalJava™実行環境上でも、J2SDK™上で動作していたプログラムなら SL Series J2ME™ Personal Profile 実行環境上でも、それぞれ動作するはずです。もしプログラムが動作しない場合は、その原因を調査してデバッグをしてください。SL Series PersonalJava™ Runtime Environment は PJEE 3.1 と互換性がありますが、それとは別に独自の制限も持ち合わせています。問題の原因はこの辺りの相違点にあるかもしれません。その場合は、SL Series に関する情報を信頼し、パッケージのデバッグおよびチューニングを行ってください。

2.3. Java™プログラムの SL Series への追加

本章では、作成したプログラムを SL Series PersonalJava™実行環境に追加する方法について説明します。

Sun Microsystems 社のサイトで公開されている J2ME™ Personal Profile 実行環境は early access 版となっており、SL-A300, SL-B500, SL-C700 にインストールしても専用のタブは出てきません（2003 年 5 月 24 日現在）。コマンドラインから使用して頂くか，以下の手法を読み，インストールするディレクトリを変更する必要があります．ご注意ください。

2.3.1. 手順と作業環境

SL Series PersonalJava™実行環境および J2ME™ Personal Profile 実行環境にプログラムを追加するには、下記の 2 つの手順を整理する必要があります。

- PC 環境内での作業
- SL Series 内での作業

SD カードにデータ書き込み可能なワークステーションを準備するか、Windows® PC に USB ドライバと（ザウルスドライブを使用するために）“ザウルスショット”をインストールしてください（両者とも SL Series の付属品です）。次に USB ケーブルを PC に接続してください。
また、jar コマンドを使用するために PC もしくはワークステーションに J2SDK™を正しくインストールしてください。米 Sun Microsystems 社のウェブサイト（http://java.sun.com）を参照して、J2SDK™のコピーとインストール指示書を入手してください。

2.3.2. PC 環境での作業

本セクションでは、Windows® PC 環境での作業について説明します。Linux® PC 環境での作業は 2.3.3 章で説明します。以下では、作業ディレクトリを C:￥hello とします。

2.3.2.1. ディレクトリ階層とファイルの作成

C:￥hello 内に適当なディレクトリ階層を作成し、ファイルを適切なディレクトリにコピーしてください。まず、以下に示されているようにディレクトリ階層を作成し、ファイルを指定された場所にコピーしてください。下記の図では、￥マーク（円マーク）のついている部分がディレクトリ名、その他がファイル名を表しています。

ただし、VGA 画面を持つ SL Series（たとえば、SL-C700）では、以下のディレクトリ構成の pics というディレクトリ名は pics144 に変更してください。

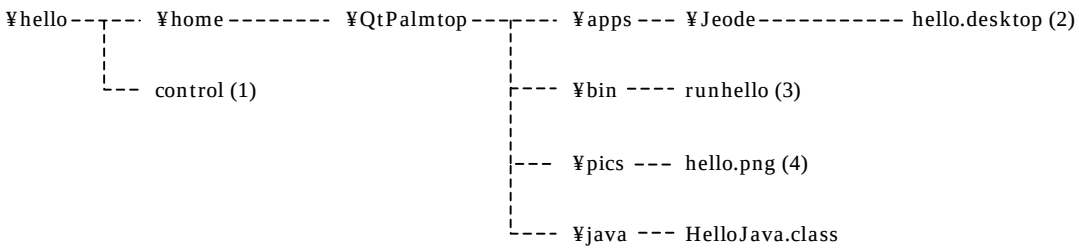


図 2-4(1):ディレクトリ階層とファイル(PersonalJava™)

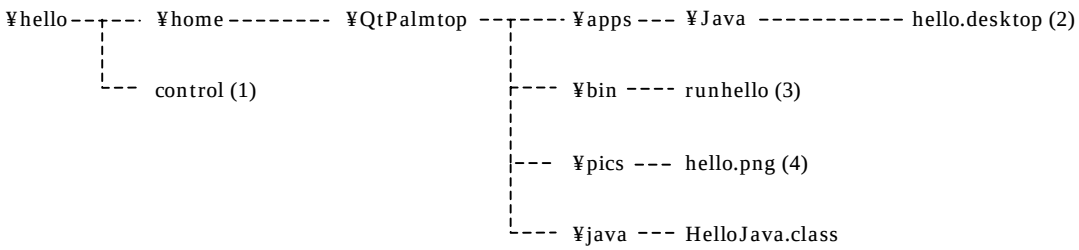


図 2-4(2):ディレクトリ階層とファイル(J2ME™ Personal Profile)

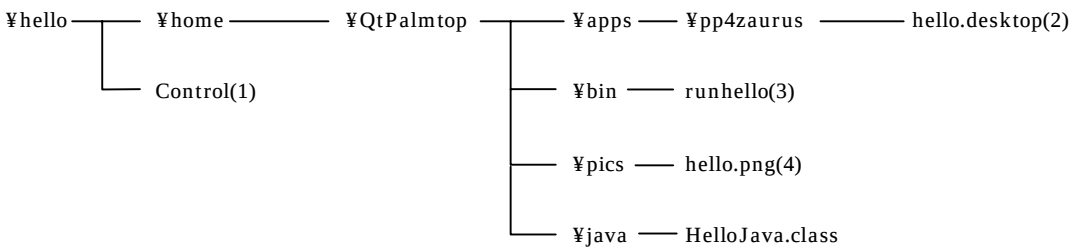


図 2-4(3)：ディレクトリ階層とファイル(J2ME™ Personal Profile early access 版)

データ構成や各ファイルの内容については以下のセクションで説明します。

2.3.2.1.1. コントロール

本セクションでは、図 2-4 内の“control (1)”ファイルについて説明します。このファイルのフィールドを以下に説明します。インストーラは“control (1)”ファイルを使って本パッケージに不可欠な基本的情報を提供します。

FIELD(フィールド)

Package:	パッケージ名(標準の表現制限規定を満たすこと)
Installed-Size:	本アプリケーションのおよそのバイナリファイルサイズを表示(バイト順に)
Filename:	パッケージファイル名 <パッケージ>_<バージョン>_<アーキテクチャ>.ipk
Version:	バージョン番号
Architecture:	Arm
Maintainer:	管理者名と管理者の電子メールアドレス
Description:	本パッケージの説明
Section:	java

EXAMPLE(例)

```
Package: java-hello
Installed-Size: 3k
Filename: ./java-hello_1.0_arm.ipk
Section: java
Maintainer: Hello <hello@sharp.com>
Architecture: arm
Version: 1.0
Description: Test sample
```

(注)本ファイルを Windows®環境で作成した場合は、ファイルを LF コード(16 進法 0x0A)で保存してください。

2.3.2.1.2. デスクトップ(hello.desktop)

本セクションでは、図 2-4 内の“desktop (2)”について説明します。このファイルはデスクトップ上のアイコンや情報を定義するものです。本ファイルには以下の文が含まれなければなりません。

[デスクトップ入力]

Comment= 本アプリケーションの簡単な説明

Exec= 実行スクリプトファイル名

Icon= 図 2-4 の(4)アイコンファイル名

Type= アプリケーション

Name= デスクトップ上で表示されるアプリケーション名

```
[Desktop Entry]
Comment=A Hello Program
Exec=runhello
Icon=hello
Type=Application
```

(注)本ファイルを Windows®環境で作成した場合は、ファイルを LF コード(16 進法 0x0A)で保存してください。

2.3.2.1.3. 実行スクリプトファイル（runhello）

本セクションでは、図 2-4 内の“runhello(3)”について説明します。これは、SL Series 上に JVM を呼び出すための実行スクリプトファイルです。このときのファイル名は、“hello.desktop”ファイルの“Exec=runhello”行にあるファイル名とまったく同一でなければなりません。このファイルに記述する内容は起動する Java™環境によって異なります。

さらに、JavaVM プロセスを認識するために使うアプリケーション名を QPE に設定する必要があります。JavaVM を立ち上げるスクリプトで、スイッチオプションを使うと、すでに起動している Java™アプリケーションに戻ることができます。例えば、以下のオプションを追加することができます。

```
-XappName=<script-name>
```

上記 〈script-name〉 は desktop ファイルの Exec に入力した内容と一致します。

また、SL-A300, SL-B500, SL-C700 では, Jeode™上のアプリケーション起動時にプログレスバーを表示させることが可能です. Jeode™のプログレスバーを表示する場合は、以下のオプションを追加します。 SL-C750/SL-C760 上の J2ME™ Personal Profile でも実行可能な共通パッケージを作成される場合, このオプションは利用しないで下さい.

```
-Xprogress
```

Jeode™を起動するときの記述例

```
. /home/QtPalmtop/bin/installdir.sh      # set INSTALLDIR
$QPEDIR/bin/evm -XappName=runhello -cp $INSTALLDIR/java HelloJava
```

J2ME™ Personal Profile を起動するときの記述例: SL-C750/SL-C760 用

```
. /home/QtPalmtop/bin/installdir.sh      # set INSTALLDIR
$QPEDIR/j2me/bin/cvm -XappName=runhello ¥
   -cp file:$INSTALLDIR/java HelloJava
```

J2ME™ Personal Profile early access 版を起動するときの記述例： SL-C700/SL-B500

```
. /home/QtPalmtop/bin/installdir.sh      # set INSTALLDIR
PATH=$PATH:$QPEDIR/j2me/bin
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$QPEDIR/j2me/lib
cvm -XappName=runhello -cp $INSTALLDIR/java  HelloJava
```

（注）本ファイルを Windows®環境で作成した場合は、ファイルを LF コード(16 進法 0x0A)で保存してください。

2.3.2.1.4. アイコンファイル（hello.png）

本セクションでは、図 2-4 内の “hello.png (4)” について説明します。ファイル名は、“hello.desktop” ファイルの “Icon=hello” 行に記述されているファイル名とまったく同一でなければなりません。

2.3.2.1.5. class ファイルと jar ファイル（HelloJava.class）

図 2-4 のファイル(5)には、必要な class ファイルと jar ファイルを全てコピーする必要があります。セクション 3.1.1 に記述されている HelloJava.java コードを使用する場合は、“HelloJava.class”と“HelloJava$1.class”の両方をコピーしなくてはなりません。

2.3.2.2. ipk ファイルの作成

SL Series 向けアプリケーションは全て ipk ファイル形式で配布されます。
このセクションでは、Windows®環境下で SL Series 用 ipk ファイルを作成する方法について説明します。

STEP1:

最初に Windows®用 tar ツールと gzip ツールを入手します。tar ツールは以下の場所で見つけることができます。

ftp://sdts.er.usgs.gov/pub/sdts/software/tools/winnt/

gzip ツールは以下の場所で見つけることができます。

http://www.gzip.org/

このセクションでは、実行ファイルが C:￥TMP ディレクトリに保存されていると仮定して説明します。

STEP2:

セクション 2.3.2.1 で作成したディレクトリに移動し、以下のコマンドをそのまま入力してください。

```
C:¥hello>¥TMP¥tar -cvf control.tar ./control
./control


C:¥hello>¥TMP¥gzip control.tar


C:¥hello>
```

カレントディレクトリに“control.tar.gz”が作成されます。

STEP3：

STEP2 で“control.tar.gz”を作成したのと同じ要領で、“hello”ディレクトリから以下のコマンドをそのまま入力してください。

```
C:¥hello>¥TMP¥tar -cvf data.tar ./home
./home/
./home/QtPalmtop/
./home/QtPalmtop/apps/
./home/QtPalmtop/apps/Jeode/
./home/QtPalmtop/apps/Jeode/hello.desktop
./home/QtPalmtop/bin/
./home/QtPalmtop/bin/runhello
./home/QtPalmtop/pics/
./home/QtPalmtop/pics/hello.png
./home/QtPalmtop/java/
./home/QtPalmtop/java/HelloJava.class
./home/QtPalmtop/java/HelloJava$1.class

C:¥hello>¥TMP¥gzip data.tar

C:¥hello>
```

カレントディレクトリに“data.tar.gz”が作成されます。

STEP4:

最後に、STEP2 と STEP3 によって作成されたアーカイブから ipk ファイルを作成します。

```
C:¥hello>¥temp¥tar -cvf hello.tar ./control.tar.gz ./data.tar.gz
./control.tar.gz
./data.tar.gz

C:¥hello>¥TMP¥gzip hello.tar

C:¥hello>rename hello.tar.gz java-hello_1.0_arm.ipk

C:¥hello>
```

最後にリネームするファイル名は、セクション 2.3.2.1.1 で定義した“control”ファイルと同じファイル名にしなくてはなりません。これら一連のタスクをより簡単に行うために、BAT ファイルを作ることをお勧めします。

2.3.3. Linux® PC 環境での作業

2.3.3.1. ディレクトリ階層とファイルの作成

$HOME/hello 内に適当なディレクトリ階層を作成し、ファイルを適切なディレクトリにコピーしてください。最初に以下に示されているようにディレクトリ階層を作成し、ファイルを指定された場所にコピーしてください。以下の図では、"/"マークのついている部分がディレクトリ、その他がファイル名を表しています。

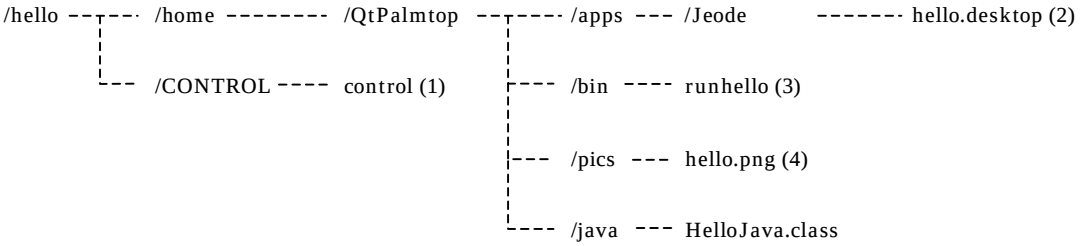


図 2-5(1):ディレクトリ階層とファイル(PersonalJava™)

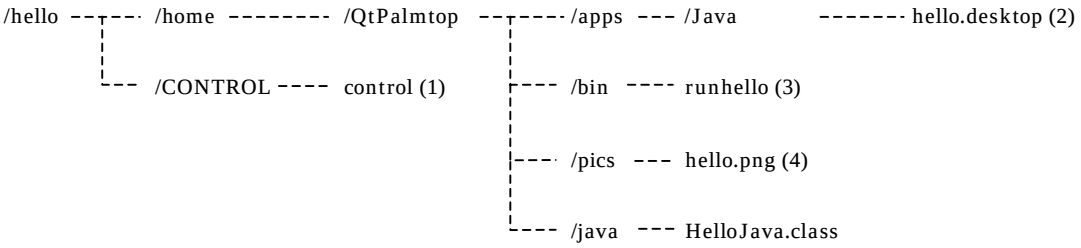


図 2-5(2) : ディレクトリ階層とファイル(J2ME™ Personal Profile)

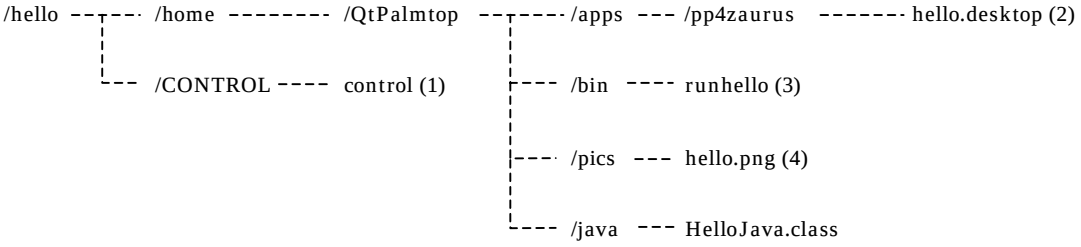


図 2-5(3) : ディレクトリ階層とファイル(J2ME™ Personal Profile early access 版)

図 2-5 にあるファイル(1)から(5)の作成方法は、Windows®環境での通常の作業と同様です。ただし、Linux® PC 環境では改行コードを考慮する必要はありません。

2.3.3.2. ipk ファイルの作成

ipk パッケージを作成するには“ipkg-build”スクリプトを使用します。このスクリプトは以下の URL で見つけることができます。

http://ipkgfind.handhelds.org/details.phtml?package=ipkg-build&official=&format

```
# ./ipkg-build <pkg_directory> [<destination_directory>]
```

このスクリプトの使用方法を以下に示します。

“destination_directory”を定義すると、定義されたディレクトリ内にパッケージファイルが生成されます。本例では、“hello”ディレクトリの親ディレクトリに以下のようにタイプする必要があります。

```
# ./ipkg-build hello
```

“java-hello_1.0_arm.ipk”が作成されます。

3. Java™アプリケーションの開発事例

本章では、以下の例を使用してコーディングからプログラム実行にいたるまでの手順を紹介します。

3.1. Java™アプリケーション開発事例

3.1.1. HelloJava

まず始めに、よく知られている HelloWorld アプリケーションを使用した簡単な例から説明します。本アプリケーションは AWT フレームに“Hello Java for SL Series”を表示します。

3.1.1.1. ソースファイルの作成

以下にソースコードを示します。テキストエディタを使用して以下の例をタイプし、ファイルを HelloJava.java として保存してください。

```
/****** HelloJava.java  ******************************************/

import java.awt.*;
import java.awt.event.*;

class HelloJava implements ActionListener {

  void init() {
    Frame f = new Frame("Hello");
    Label label = new Label("Hello Java for SL series",  Label.CENTER);
    Button b = new Button("exit");

    f.setSize(240,280);
    f.setLayout(new BorderLayout());

    b.addActionListener(this);
```

＜続きは次頁＞

＜以下、続き＞

```
    f.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });

    f.add(label, BorderLayout.CENTER);
    f.add(b, BorderLayout.SOUTH);
    f.show();
  }

  public void actionPerformed(ActionEvent e) {
    System.exit(0);
  }

  public static void main(String[] argv) {
    HelloJava hj = new HelloJava();
    hj.init();
  }
}

/****** HelloJava.java (ENDS HERE) ********************************/
```

（注）どの Java™アプリケーションを使用する場合もメインメソッドが必要となります。加えて、アプリケーションは その終了のために、“Window Closing”用 windowEvent を実装しなければなりません。本例では、このイベントの実行には WindowAdapter クラスを使用しています。

3.1.1.2. ソースファイルのコンパイル

J2SDK™の Java™コンパイラを使用して上記ソースをコンパイルします。以下の例では、J2SDK™ 1.4.1_01 (C:￥j2sdk1.4.1_01)を使用します。Windows®/DOS コマンド内で以下のように実行してください。

```
$>C:¥j2sdk1.4.1_01¥bin¥javac –target 1.1 HelloJava.java
```

HelloJava.class という class ファイルが同じディレクトリ内に作成されます。

3.1.1.1 JavaCheck™を使う

ここでは新たに作成した class ファイルを JavaCheck™にかけます。以下の例では、GUI 機能付きの JavaCheck™バージョンを使用します。GUI がサポートされていない JavaCheck™を使う場合でも同様の結果が得られるはずです。

(1) JavaCheck™の起動

このセクションでは、C:￥JavaCheck3.0 ディレクトリ内の JavaCheck™ 3.0 を使用します。J2SDK™ 内の java を使って JavaCheck™を起動します。

```
$>C:¥j2sdk1.4.1_01¥bin¥java -cp C:¥JavaCheck3.0¥bin¥JavaCheck.jar JavaCheckV
```

ウィンドウが2つ現れるので、“Continue(続行)”ボタンを押して次に進んでください。

(2) 仕様ファイルの設定

JavaCheck™ GUI 内では、以下 4 つのタブを使用してウィンドウ間を移動してください。

- 仕様（Specification）
- クラス（Classes）
- 分析と結果（Analysis & Results）
- ヘルプ（Help）

最初に“1. 仕様（Specification）”タブを選択して仕様ファイルを設定します。JavaCheck™ 3.0 をダウンロードしたのと同じウェブサイトからダウンロード可能な“pjava_1.1.0.spc”ファイルを使ってください。次に、ディレクトリ内の pjava_1.1.0.spc がある箇所に行き、pjava_1.1.0.spc を選択・追加してください。

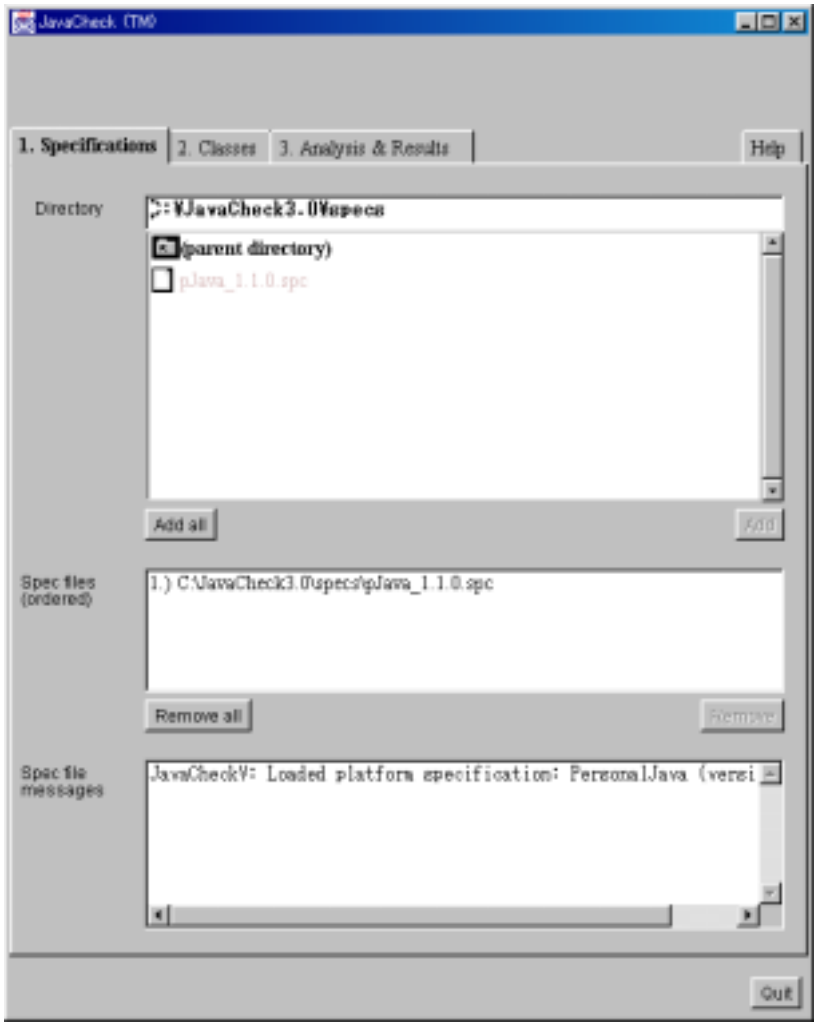


図 3-1：JavaCheck™仕様ファイルの選択

(3) クラスファイルの設定

“2.クラス(Classes)”のタブを選択し、全ての class ファイルに対してクラスファイルを指定してください。アーカイブされたクラスを含む jar ファイルまたは zip ファイルを選択することもできます。一部分析を行う必要がある場合は、分析すべきクラスファイル部分を選択してください。

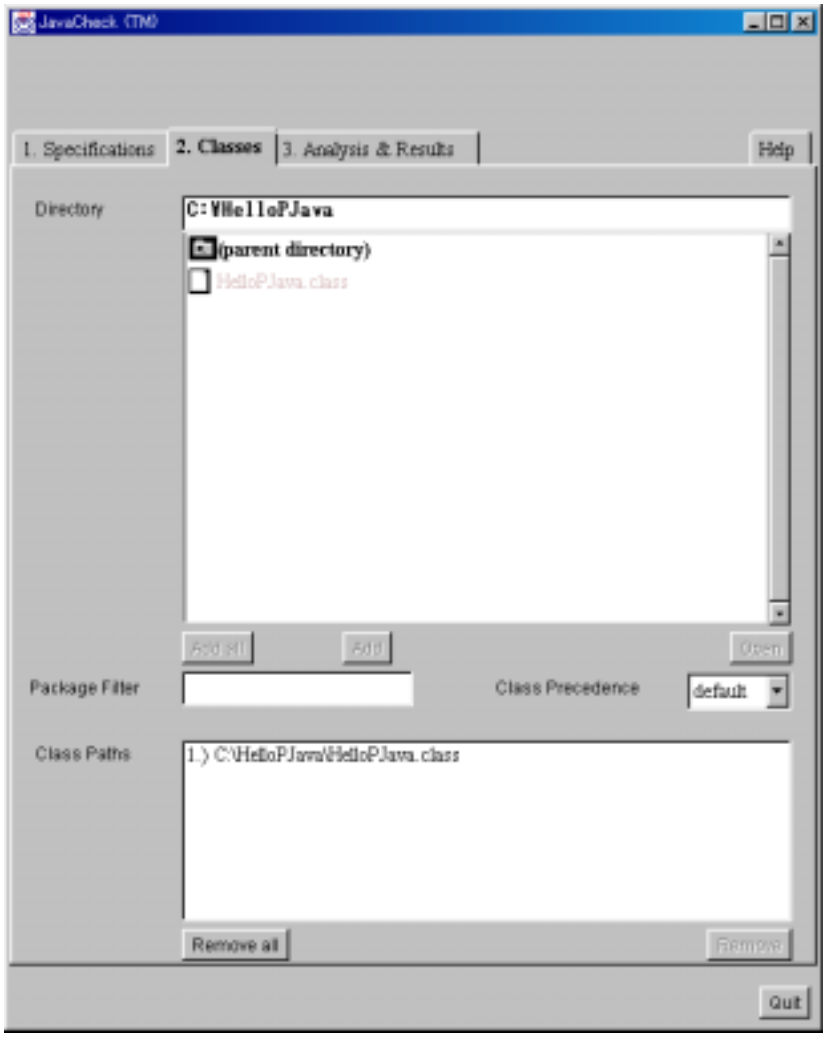


図 3-2：JavaCheck™クラスファイルの指定（jar ファイル）

(4) 分析

分析は“3. 分析と結果(Analysis & Results)”タブを利用して行います。ステータス(status)部分の“Analyze Classes（クラス分析）”ボタンをクリックして分析を開始してください。HelloJava class のようなシンプルなクラスの場合にはエラーは発生しないはずです。
エラーが発生した場合は、クラス名、エラーコード、依存度、オプショナルエラー情報が表示されるのでチェックしてください。

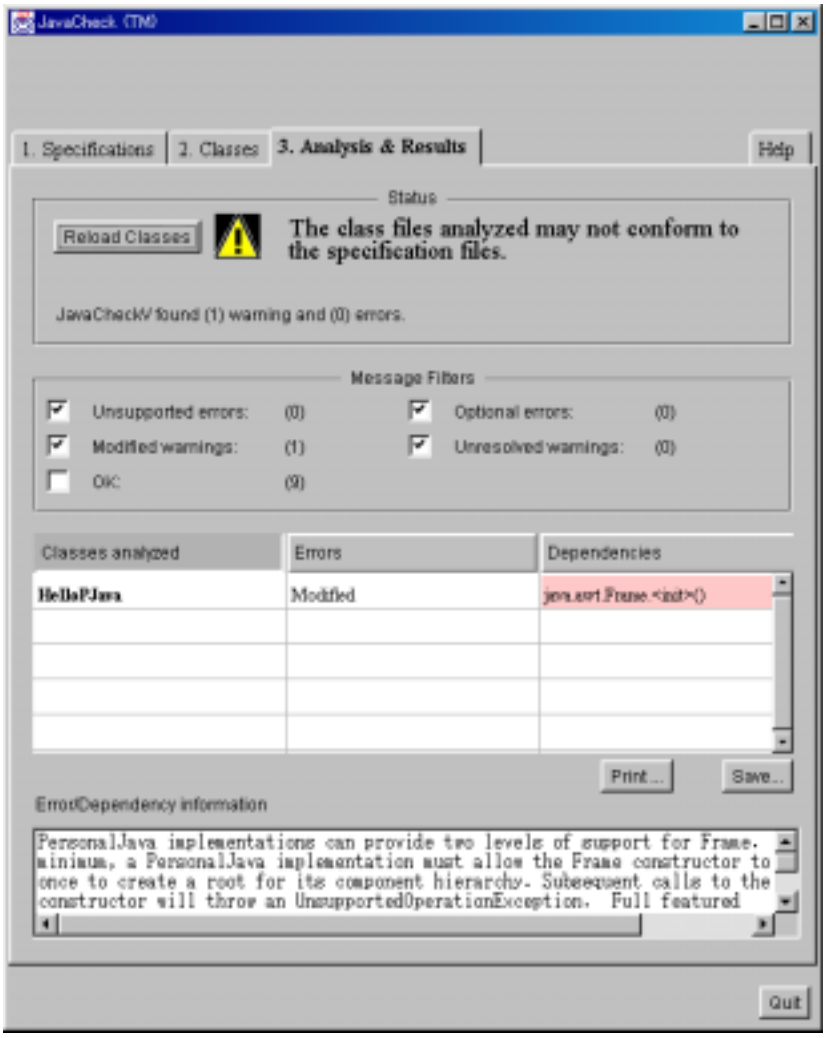


図 3-3：JavaCheck™分析結果

java.awt フレームクラスのメソッド初期化の際に“変更済み警告(Modified Warning)”が表示されます。この警告は、PJAE 仕様内では“変更済み(Modified)”と表示されます。しかし、SL Series を動作している際にこのような警告が出てきても問題とはなりませんので、次のステップに進んでください。

3.1.1.3. SL Series 上での HelloJava 動作

SL Series 上で ipk ファイルを動作させます。ipk ファイルを SD カードにコピーしカードを SL Series に挿入して「ソフトウェアの追加／削除」からインストール、または、ザウルスドライブを使用して ipk ファイルを SL Series へ移行し（Windows® PC のみ）「ソフトウェアの追加／削除」からインストールしてから動作させてください。実行後は“終了(exit)”ボタンを押してアプリケーションを終了してください。

3.1.2. HelloJava アプレット

ここでは SL Series 上で動作するアプレット例を紹介します。このシンプルなプログラムを使うと、アプレットフレームに“Hello Java Applet for SL Series”というテキスト文字列が表示されます。

3.1.2.1. アプレットの作成

この “HelloJavaApplet.java”は以下のように記述されます。

```
/***** HelloJavaApplet.java ****************************/
  import java.awt.Graphics;
  import java.applet.Applet;

  public class HelloJavaApplet extends Applet {
   public void paint(Graphics g) {
   g.drawString("Hello Java Applet for SL Series", 10, 50);
   }
  }
/***** HelloJavaApplet.java (ENDS HERE) ****************/
```

Java™コンパイラを使用して上記ソースコードをコンパイルしてください。JavaCheck™方法については、HelloJava アプリケーション例と同じなので、ここでは説明を省きます。

3.1.2.2. HTML ファイルの作成

```
<--- HJApplet.html ------------------------------- >
<HTML>
<HEAD>
<TITLE>Hello Java Applet</TITLE>
</HEAD>
<BODY>
Output of Hello Java Applet
<HR>
<APPLET CODE="HelloJavaApplet.class" WIDTH=230 HEIGHT=280>
</APPLET>
</BODY>
</HTML>
<--- HJApplet.html (ENDS HERE) ---------------------->
```

上記アプレット用 HTML ファイルを作成します。

最初にこれらファイルを PC 上で稼動します。

J2SDK™ 1.4.1_01 での実行例

```
$> C:¥j2sdk1.4.1_01¥bin¥java sun.applet.AppletViewer HJApplet.html
```

PJEE 3.1 での実行例

```
$> C:¥pjee3.1¥bin¥pjava sun.applet.AppletViewer HJApplet.html
```

“Hello Java Applet for SL series”というウィンドウが表示されます。AppletViewer は、“Applet started”というステータスを下に表示します。

3.1.2.3. ipk ファイルの作成

詳細については 2.3.2.2 を参照してください。2.3.2.2 の例とアプレットを作成する場合との大きな差は、以下に説明する実行スクリプトファイルの違いです。

Jeode™を起動するときの記述例

```
. /home/QtPalmtop/bin/installdir.sh # set INSTALLDIR
$QPEDIR/bin/evm -appletviewer $INSTALLDIR/java/HJApplet.html
```

cvm を起動するときの記述例

```
. /home/QtPalmtop/bin/installdir.sh      # set INSTALLDIR
$QPEDIR/j2me/bin/cvm –appletviewer file:$INSTALLDIR/java/HJApplet.html
```

pp4zaurus を起動するときの記述例

```
. /home/QtPalmtop/bin/installdir.sh      # set INSTALLDIR
PATH=$PATH:$QPEDIR/j2me/bin
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$QPEDIR/j2me/lib
cvm sun.applet.AppletViewer $INSTALLDIR/java/HJApplet.html
```

（注）本ファイルを Windows®環境で作成した場合は、ファイルを LF コード(16 進法 0x0A)で保存してください。

3.1.3. SL Series へのファイル移行、インストール、動作

SL Series へファイルを移行、インストールし呼び出す方法は以下の通りです。

Windows® PC から SL Series への移行

①. SD カードに ipk ファイルをコピーし、カードを SL Series に挿入して「ソフトウェアの追加／削除」からインストールして動作

②.ザウルスドライブを使用して ipk ファイルを SL Series へ移行し、「ソフトウェアの追加／削除」からインストールして動作

Linux® PC から SL series への移行

①. SD カードに ipk ファイルをコピーし、カードを SL Series に挿入してインストールして動作

クラスファイルの実行方法

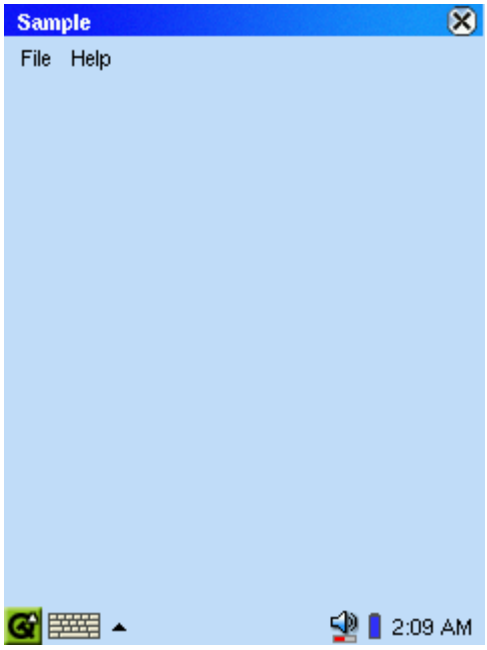
パッケージ化前のクラスファイルを上記方法で SL Series へ移行し動作させるには、ターミナルアプリケーションを起動し、コマンドラインから実行してください。ただし、ターミナルアプリケーションが内蔵されていない機種では、以下の Web サイトの開発サポートツールからターミナルソフトをダウンロードしてください。

http://more.sbc.co.jp/sl_j/tool/tools.htm

4. Appendix

4.1. サンプルソースコード

以下に QVGA 画面の SL Series で動作するテンプレートサンプルコードを示します。コンパイルして実行すると、以下のような画面が SL Series 上に表示されます。




```
/****** Sample.java  ****************************************/

import java.awt.*;
import java.awt.event.*;

final public class Sample extends WindowAdapter
    implements ActionListener {

    // Main frame
    private Frame mainFrame;

    // Menu items -- file --
    private Menu fileMenu;
    private MenuItem exitMI;
```

＜続きは次頁＞

＜以下、続き＞

```
  // Menu items -- help --
  private Menu helpMenu;
  private MenuItem aboutMI;

  // Panel
  private Panel mainPanel;

  // Window size
  private int windowWidth = 240;
  private int windowHeight = 320;
  private int titleHeight = 13;
  private int taskbarHeight = 19;
  private int margin = 2;

public Sample () {
    // Create frame
    mainFrame = new Frame("Sample");

    // Create menu bar
    MenuBar mb = new MenuBar();
    mainFrame.setMenuBar(mb);

    // File menu
    fileMenu = new Menu("File");
    exitMI = new MenuItem("Exit");
    exitMI.addActionListener(this);
    fileMenu.add(exitMI);
    mb.add(fileMenu);
```

＜続きは次頁＞

＜以下、続き＞

```
        // Help menu
        helpMenu = new Menu("Help");
        aboutMI = new MenuItem("About");
        aboutMI.addActionListener(this);
        helpMenu.add(aboutMI);
        mb.add(helpMenu);

        // Setup panel
        mainPanel = new Panel();
        mainFrame.add(mainPanel);

        mainPanel.setVisible(true);

        // Setup frame
        mainFrame.addWindowListener(this);
        mainFrame.setSize((windowWidth - margin * 2),
                          (windowHeight - titleHeight - taskbarHeight
                           - margin * 2));
        mainFrame.setVisible(true);
        mainFrame.setResizable(false);
    }

    // Event handling

    // window event
    public void windowClosing (WindowEvent we) {
        if (we.getWindow() == mainFrame) {
            System.exit(0);
        }
    }
```

＜続きは次頁＞

＜以下、続き＞

```
    // action event
    public void actionPerformed (ActionEvent ae) {
         Object o = ae.getSource();

         if (o instanceof MenuItem) {
             MenuItem mi = (MenuItem)o;
             if (mi == exitMI) {
                  System.exit(0);
             } else if (mi == aboutMI) {
                  new SampleAbout();
             }
         }
    }

    public static void main (String args[]) {
         Sample foo = new Sample();
    }
}

/****** Sample.java (ENDS HERE) ********************************/

/****** SampleAbout.java  ***************************************/

import java.awt.*;
import java.awt.event.*;

final public class SampleAbout extends WindowAdapter
    implements ActionListener {

    // About frame
    private Frame aboutFrame;
```

＜続きは次頁＞

＜以下、続き＞

```
    // Panel
    private Panel aboutPanel;

    // Label
    private Label aboutLabel;

    // Button
    private Button okButton;

    public SampleAbout () {
        // Create frame
        aboutFrame = new Frame("About: Sample");

        // Setup panel
        GridBagLayout gbl = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();

        aboutPanel = new Panel(gbl);
        aboutFrame.add(aboutPanel);

        // Setup label
        aboutLabel = new Label();
        aboutLabel.setText("This application is a prototype for SL Series.");
        gbc.gridwidth = GridBagConstraints.REMAINDER;
        gbl.setConstraints(aboutLabel, gbc);
        aboutPanel.add(aboutLabel);

        // Setup button
        okButton = new Button("OK");
        okButton.addActionListener(this);
        gbc.gridwidth = GridBagConstraints.RELATIVE;
        gbc.fill = GridBagConstraints.NONE;
        gbl.setConstraints(okButton, gbc);
        aboutPanel.add(okButton);
```

＜続きは次頁＞

＜以下、続き＞

```
        aboutPanel.setVisible(true);

        // Setup frame
        aboutFrame.addWindowListener(this);
        aboutFrame.pack();
        aboutFrame.setVisible(true);
        aboutFrame.setResizable(false);
    }

    // Event handling

    // window event
    public void windowClosing (WindowEvent we) {
        aboutFrame.dispose();
    }

    // action event
    public void actionPerformed (ActionEvent ae) {
        if ((Button)ae.getSource() == okButton) {
           aboutFrame.dispose();
        }
    }
}

/****** SampleAbout.java (ENDS HERE) ********************************/
```