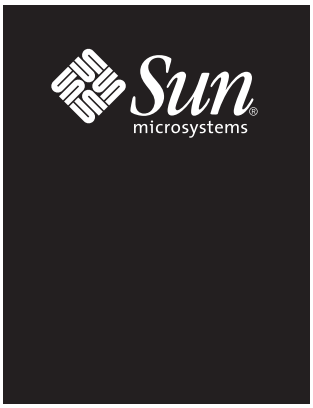
Java™ Platform Migration Guide

Version 1.3 to 5.0



This guide helps developers migrate Java™ applets, standalone applications, Java™ Web Start applications, and development tools from version 1.3 of the Java platform to version 5.0. While many 1.3 applications run without change, compatibility issues do exist, as described in this guide. The first three sections cover issues that are of interest to all application developers. The fourth section covers issues that are primarily of interest to platform implementers and tool developers.

Notes: *In this guide, "1.4" means "version 1.4.x of the J2SE™ Platform", and "5.0" means "version 5.0 of the Java Platform".*

For late-breaking issues and known bugs in the latest release, be sure to consult the Release Notes.

Table of Contents

Chapter 1
Runtime Issues

These issues affect 1.3 classes running on the 5.0 platform.

1.1 AWT

The AWT GUI component library has been modified to improve cross-platform behavior, performance, and interoperation with lightweight GUI components.

1.1.1 MouseEvent.MOUSE_LAST

As of 1.4, the value of static final field `MOUSE_LAST` in class `java.awt.event.MouseEvent` changed to `507`. In previous versions, the value was `506`.

Because compilers hard-code static final values at compile-time, code that refers to `MOUSE_LAST` and that was compiled under 1.3 retains the old value. Such code needs to be recompiled to work properly in 5.0.

1.1.2 AWT Focus Changes

Most developers of AWT 1.3 applications saw incompatibilities when migrating to 1.4, so it is a good idea to verify the focus-behavior of your 1.3 applications in 5.0. The general issues are summarized here; windows-specific issues are in the next section. For details on these issues and the architecture changes that led to them, see The AWT Focus Subsystem.

1. The default focus traversability for all Components in 5.0 is `true`. Previously, some Components (in particular, all lightweights), had a default focus traversability of '`false`'.

   Note: Despite this change, the `DefaultFocusTraversalPolicy` for all AWT Containers preserves the traversal order of previous releases.

2. A request to focus a non-focus traversable (i.e., non-focusable) Component is denied in 5.0. Previously, such requests were granted.

3. In 5.0, `Window.toFront()` and `Window.toBack()` perform no operation if the Window is not visible. Previously, the behavior was platform-dependent.

4. Focus traversal keys (in most cases this means the TAB key) are now consumed, which can cause problems if a program depends on a key listener being notified of these key events. Previously, AWT components saw these events and had an opportunity to consume them before AWT initiated focus traversal. To avoid focus traversal keys being consumed, use the following code:

```
component.setFocusTraversalKeysEnabled(false);
```

where component is the Component that is firing the key events. Focus traversal can then be handled manually. Alternatively, the code can use an `AWTEventListener` or `KeyEventDispatcher` to pre-listen to all key events. For more information, see bug 4650902.

5. As of 1.4, "opposite" fields were added to `java.awt.event.FocusEvent` and `java.awt.event.WindowEvent`. For WindowEvent, for example, the "opposite" is the other Window that participated in the state change. If the source of the event was activated, the opposite would be the Window that was deactivated, and vice versa. For FocusEvent, the opposite is the other component that participated in the focus transfer. If the source of the event has gained focus, then the opposite is the component that lost focus, and vice versa.

   The `source` field in java.util.EventObject, from which other events are inherited, is transient. Since FocusEvent.opposite and WindowEvent.opposite are not transient, serializing and deserializing them doesn't make sense. So, as of 1.4.2, WindowEvent.opposite and FocusEvent.opposite are set to null after deserialization.

   The bug report associated with this change is 4759974.

6. As of 5.0, any container can provide a focus traversal policy; the new `FocusTraversalPolicyProvider` property of `Container` indicates whether it does. Previously, only containers that were focus cycle roots could provide a focus traversal policy.

   The focus traversal policies provided with the Java platform have been changed in 5.0 to accommodate focus traversal policy providers. Specifically, when a policy encounters a focus traversal policy provider during forward (backward) traversal, it should not treat its components as belonging to the provided focus cycle root but should use the focus traversal policy of focus traversal policy provider to get next (previous) component. If the returned component is the same as the first (last) component returned by the focus traversal policy of the focus traversal policy provider, then invoking the policy should get the next (previous) component in the cycle after (before) the focus traversal policy provider. Calculation of "first" and "last" components in focus cycle roots should use the focus traversal policies of focus traversal policy providers when necessary (when a "first" or "last" component is itself a `Container` and a focus traversal policy provider).

   Because this change doesn't require any new methods in focus traversal policies, third-party focus traversal policies will continue to work, although they will not support the notion of providers.
   If you have written a focus traversal policy and wish to support providers, you need to make changes similar to the ones made to the platform-provided policies in 5.0.

   For more information, see the Focus Traversal Policy Providers section of the focus specification, The AWT Focus Subsystem.

1.1.3 AWT Focus Changes on Microsoft Windows

1. `Window.toBack()` changes the focused Window to the top-most Window after the Z-order change.
2. `requestFocus()` now allows cross-Window focus change requests in all cases. Previously, requests were granted for heavyweights, but denied for lightweights.

1.1.4 AWT (XToolkit / XAWT) on Solaris™/Linux

AWT has been re-implemented on the Solaris™ and Linux platforms in 5.0. The new Toolkit implementation provides the following advantages:

- Removes the dependency on Motif and Xt libraries.

- Interoperates better with other GUI Toolkits.
- Provides better performance and quality.

The new Toolkit (XToolkit) is the default on Linux in 5.0. Solaris will continue to use the MToolkit (Motif-based Toolkit) as the default in 5.0, but eventually it will be replaced with XToolkit.

You can explicitly set the toolkit for an applet or application using an environment variable or a system property, or you can use the Java Plug-In Control Panel. (Keep in mind that an environment variable overrides the system property.)

Setting an environment variable

You can set an environment variable before starting the VM:

csh

```
setenv AWT_TOOLKIT XToolkit #selects the XToolkit
setenv AWT_TOOLKIT MToolkit #selects the MToolkit
```

ksh/bash

```
export AWT_TOOLKIT=XToolkit
export AWT_TOOLKIT=MToolkit
```

Setting a system property

Alternatively, you can use a system property on the command line:

```
java -Dawt.toolkit=sun.awt.X11.XToolkit MyApp
java -Dawt.toolkit=sun.awt.motif.MToolkit MyApp
```

Using the Plug-in Control Panel

- Launch the Java Plug-in Control Panel:

  ```
  $java_home/bin/ControlPanel
  ```

- Add the system property to the Java Runtime Parameters field, accessible from the View Java Applet Runtime Settings and View Java Application Runtime Settings buttons under the Java tab.

  ```
  -Dawt.toolkit=sun.awt.X11.XToolkit
  -Dawt.toolkit=sun.awt.motif.MToolkit
  ```

Setting the toolkit for an applet

If the browser is started from a terminal window, you can set the environment variable in the terminal window before launching the browser.

If the browser is launched from a desktop icon or menu, use the Java Plug-in Control Panel, since there is no way to set an environment variable for the browser in this case.

1.1.5 AWT Drag and Drop

1. In 1.4.0, drag and drop behavior changed to fix bug 4395290. In the new behavior, dragExit() is called on the DropTarget and on the DropTargetListener registered with it only when the mouse pointer has exited the operable part of the drop site for this DropTarget during a drag operation. Previously, these methods were also called immediately before `drop()` was called on the DropTarget or the DropTargetListener respectively. The old behavior was documented in the Drag and Drop Specification, but was inconsistent with the Java 2 Platform API Specification. The old behavior was also inconvenient and imposed a significant impact on the usability of Drag and Drop in Swing. Starting with J2SE 1.4.0, the behavior matches the Java 2 platform API

Specification. The Drag and Drop Specification was updated to reflect this change.

2. A change in behavior was introduced in 1.4.0 as a fix for bug 4426794 and bug 4435403. As of 1.4, drag notifications are dispatched to the top-most Component under the mouse cursor with an active drop target. Previously, drag notifications were always dispatched to the top-most Component under the mouse cursor. If this Component did not have an associated drop target, the notifications are discarded. That design had two major flaws:
   - Drag notifications had to ignore the Swing glass pane. Otherwise, it would consume all drag notifications otherwise. But that prevented developers from using a Swing glass pane as a drop target.
   - To implement drop support on a compound Component such as a `JColorChooser`, and accept a drop anywhere inside of it, developers were forced to install drop targets on all descendants of the compound Component, which was unwieldy and inefficient.
3. Previously, the only drag and drop (DnD) protocol supported on X11 was the Motif DnD protocol. In 5.0, the XDND protocol is also supported, and the Motif DnD protocol has been reimplemented so that it no longer depends on the Motif library. It's possible that regressions might be caused by the difference between the new Motif DnD protocol implementation and one provided by the Motif library. However, the Motif library's implementation is buggy, and it's believed that the new implementation is at least as high in quality, as well as better supported.

   For more information on this change, see bug 4638443.

1.2 Java2D™ Graphics

This section lists issues related to Java2D™ graphics, font rendering, and image handling. For detailed descriptions of the architecture that underlies the changes, consult the following resources:

- High Performance Graphics white paper
- New Java2D Features in 1.4
- New Java2D Features in 5.0
- Java2D System Properties

1.2.1 Behavior Changes

1. Font metrics information is different in 5.0 than it was in 1.3. For programs using ANSI-codepage fonts, the differences are small, but have a cumulative effect as the number of components rises. For programs using Asian fonts, the differences can mount quickly.

   As of 1.4, the calculations are more accurate. Previously, the calculations were often rounded up incorrectly. In 1.4 the libraries report sizes that better correspond to the font's true size (but still as integers for the APIs that report only integers).

   For more information, see bugs 4711444., 4455492, and 4467709.
2. As of 1.4, the outline returned by `GlyphVector.getGlyphOutline` and the bounds returned by `GlyphVector.getGlyphVisualBounds` are positioned differently- around the origin of each individual glyph. Previously, the outline and the bounds were positioned around the point (0, 0). This change makes the results consistent with the behavior of `GlyphVector.getGlyphLogicalBounds`.

1.2.2 Image Handling

1. Previously, passing a null `Image` parameter to a `Graphics.drawImage()` method resulted in a `NullPointerException`. As of 5.0, it doesn't. Applications that passed a null Image worked with the Microsoft virtual machine. With the new behavior, those applications work in the 5.0 virtual machine, as well.
2. Hardware acceleration for image scaling can be accessed on Microsoft Windows platforms. That feature is disabled by default, however, to guarantee rendering quality and consistency. Use the following runtime flag to enable hardware-accelerated scaling. Deploy with it only after confirming that your application behaves properly:

   ```
   java -Dsun.java2d.ddscale=true
   ```

3. In 1.4, the `VolatileImage class` was introduced to provide an image that would be hardware accelerated, if possible, on the runtime platform. Previously, images stored in accelerated memory were "lost" on some platforms (Windows, in particular) for reasons outside the control of the application. The new image type was created to help prevent such image loss.

   When `VolatileImage`s can take advantage of hardware acceleration, it accelerates rendering operations both to the image and to copies made from the image. These accelerations may help even simple applications get much greater performance than was previously possible.

   Note: As of 1.4, Swing uses `VolatileImage` for its back buffer, so Swing applications automatically benefit from the accelerated performance.

   As of 5.0, Java2D provides accelerated support for image copying no matter how the images are created. When Java2D detects copies from an image to another destination image or window, the library creates a cached/accelerated version of that image. The application automatically takes advantage of the acceleration, whether the image was created using `VolatileImage, Component.createImage()`,or even if it was constructed manually using new `BufferedImage()`. The use of the accelerated version no matter how the acceleration is achieved-whether it was from using DirectX on Windows, or OpenGL on all platforms (depending, of course, on the platform and runtime flags).

1.2.3 Graphic Accelerations

New BufferStrategy class

In 1.4, the BufferStrategy class was introduced to make graphics-buffering easier. Double-buffering of graphics produces smooth animations and rendering updates, as Swing does with its back buffer. Using the new class makes it easier to implement that feature. The new class also offers more powerful functionality, compared to doing it manually via older image types or the new VolatileImage type.

Support for DirectX on Windows

As of 1.4 Java2D began using the DirectX graphics library on Windows to achieve acceleration for basic GUI and graphics objects. That implementation allows objects such as the Swing back buffer to reside in Video Memory (VRAM) and to benefit from acceleration of simple rendering operations to and from such off-screen surfaces. Use of DirectX can result in graphics artifacts in some instances. Java2D provides some command-line flags that can used to help isolate the problems as well as work around them in user applications:

- **`-Dsun.java2d.d3d=false`** — this flag disables Java2D's use of Direct3D for drawing lines and other entities. Since some video cards have issues with their Direct3D drivers, using this flag makes it possible to identify problems that are related to the 3D driver.
- **`-Dsun.java2d.noddraw=true`** — this is the most drastic flag. It disables all hardware acceleration on Windows (including Direct3D for lines and DirectDraw for all other acceleration features).

Support for OpenGL on all platforms

As of 5.0, Java2D can be run using OpenGL, so applications can achieve high performance using hardware acceleration for such advanced rendering features as translucency, anti-aliased text, and transform operations (as well as the more basic line/fill/copy operations used in GUIs). OpenGL is supported in all Sun-provided platforms (Windows, Linux, and Solaris).

However, due to inconsistent driver support, OpenGL rendering is not enabled by default on any platform. To enable the acceleration, use the following command-line flag:

```
-Dsun.java2d.opengl=true
```

Note: Performance and robustness of this rendering approach varies on different hardware platforms. Applications should only enable it on particular platforms after careful testing. Enabling it on unknown platforms may not give the results you want.

1.2.4 X11-related Improvements

Programs running on Solaris and Linux under X11 will see major performance improvements. This section summarizes the changes and shows how to take advantage of them. It also describes flags you can set to further tune performance.

Accelerated-image reading performance improvements

Previously, translucency and scaling operations to and from an accelerated image were slow, as a result of frequently reading the image from VRAM, which is much slower than reading it from system memory. Translucency operations require frequent reads because alpha compositing performs read-modify-write operations on the destination. Scaling from an accelerated image requires reading from the source or destination image in order to perform the operation successfully.

As of 1.4, Java2D transfers the surface to system memory if the image is experiencing frequent reads. If reads occur less frequently, Java2D transfers the surface back to VRAM. If you are working in a UNIX® environment, you can override this heuristic using the `J2D_PIXMAPS` environment flag. For more information, see "Environment Flag for Solaris and Linux" in Java2D System Properties.

Remote X server performance improvements

When working with graphics in a UNIX or Linux environment, you can perform your graphics computations on a remote client by running an X Server from your machine. However, off-screen images experienced poor performance in this scenario, because the off-screen image was created on the client side. Every time you needed to re-render the image to the destination, the image had to be copied from the remote X client to the display. If you were using the off-screen image for double buffering, the image was copied across the network whenever the screen was repainted.

As of 1.4, accelerated off-screen images are available, so the off-screen image is created on the server side, local to the screen. Java2D uses X protocol requests, which tell the X server what and how to render to the off-screen image located on the server side of the network. Only X protocol requests are sent over the network; the image itself stays on the server side. This change improved Swing performance, as well, because Swing uses double-buffering-do a Swing application no longer has to wait for the back buffer to be copied over the network for every screen refresh.

One drawback with both remote X and DirectDraw is that neither antialiasing nor alpha-blending can be accelerated. In fact, antialiasing and alpha blending operations on remote X are usually much slower than they were in 1.3 because the image must be copied to the X client to perform one of those operations, after which the new image must be copied back to the server. The Java2D team is looking into solutions to this problem for future

releases. (This issue is generally not a problem for Swing applications, since most do not use alpha blending or antialiasing.)

Local X server performance improvements
As of 1.4, Java2D uses the Shared Memory Extension in the local display environment on Solaris and Linux. The Shared Memory Extension allows an X server and X client running on the same machine to jointly access shared memory, which enables faster data transfers. This change produces better performance when rendering to the screen and handling images.

When DGA is not available, toolkit images and images created with `Component.createImage` or `GraphicsConfiguration.createCompatibleImage` are stored in pixmaps instead of system memory, enabling faster copies to the screen using X protocol requests. You can override this behavior with the pmoffscreen runtime flag, described in the "Runtime Flag For Solaris and Linux" in Java2D System Properties.

Pixmaps that are not in shared memory might be stored in VRAM, which allows for fast copies to the screen, but reading from these images is very slow, as described in the previous section, Accelerated-Image Reading Performance Improvements. Since the Shared Memory Extension is now accessible in the local display environment, images that experience frequent reads can be stored in Shared Memory Pixmaps, which always reside in system memory. Java2D can automatically transfer the image to the appropriate memory, depending on how frequently the image is read or copied. You can control which memory is used by setting the `J2D_PIXMAPS` environment variable, as described in "Environment Flag for Solaris and Linux" in Java2D System Properties.

Support for new display pixel formats
As of 1.4.2, displays with 3-byte RGB pixel format are supported. This display format is commonly used on Linux systems.

Support for 12-bit PseudoColor visuals was also implemented in 1.4.2.

A number of performance and quality-related enhancements were added to support the grayscale image formats, ByteGray, 12-bit UShortGray that are commonly used in medical industry applications.

1.3 Support for Supplementary Unicode Characters

A *supplementary character* is a Unicode characters whose 32-bit numeric value (*code point*) is above U+FFFF, and which therefore cannot be described as single 16-bit entity such as the `char` data type in the Java programming language. Such characters are generally rare, but some are used, for example, as part of Chinese and Japanese personal names.

Support for supplementary characters has been introduced into the Java platform with an approach that enables most character-handling applications to run without change. Applications that interpret individual characters can also run unchanged, so long as the character data does not include supplementary characters. Applications that interpret individual characters which do include supplementary characters can use the new code point-based APIs in the `Character` class and various `CharSequence` subclasses.

In many cases, you can avoid doing character-conversions programmatically by using (generally more convenient) 5.0 APIs that already support supplementary characters. For example, instead of using:

```
System.out.println("Character " + String.valueOf(char) + " is invalid.");
```

You can use the print formatting API, which supports supplementary characters:

```
System.out.printf("Character %c is invalid.%n", codePoint);
```

Using this higher-level API is not only simpler, it avoids the concatenation that makes the message hard to localize, and it reduces the number of strings that need to be moved into a resource bundle from two to one.

In detail:

- You do **not** have to change — Applications that deal with text only in the form of `char` sequences in all forms (`char[]`, implementations of `java.lang.CharSequence`, implementations of `java.text.CharacterIterator`), and only use Java APIs that accept and return such char sequences. In these cases, the implementation of the Java platform APIs handles supplementary characters for you.
- You do **not** have to change — Applications that interpret individual characters; pass individual characters to Java platform APIs; or call methods that return individual characters, when supplementary characters are not processed. For example, if an application scans a char sequence for HTML tags, checking each char individually, it knows that those tags only use characters from the Basic Latin block, so no supplementary characters will not be processed-even if supplementary characters are included in the UTF-16 based `char` sequence.
- You do have to change — Applications that interpret individual characters; pass individual characters to Java platform APIs; or call methods that return individual characters when those character values can include supplementary characters.
- You also have to consider — whether standard or modified UTF-8 is required when converting to and from UTF-8, so you use the proper Java platform facilities in each case. Modified UTF-8 is used by internal APIs in the Java platform. Standard UTF-8 is used for all externally-facing APIs. For more information, see Modified UTF-8.

When converting a character-handling application:

- Where a parallel API is available — one that uses `char` *sequences*, rather than simple `char` *values* — the best approach is to convert the application to use those APIs.
- When a parallel API is not available — use the new code point-based APIs in conjunction with `Character.toCodePoint(char high, char low)` API to convert two UTF-16 code units to a single 32-bit code point. (The `Character.toChars(int codePoint)` API goes the other way, converting a code point to one or two UTF-16 code units wrapped in a `char[]`.)
- For maximum simplicity — convert all text into code point representation (say, an `int[]`) and process it in that representation. Then you never need to worry about character conversions.
- For greater convenience and maximum performance — continue using `char` sequences in the application, and only convert to code points when needed. There are more Java platform APIs that use char sequences, and using char sequences also saves memory space.

For an excellent tutorial on the subject of supplementary characters, code points, and Unicode representations, see: Supplementary Characters in the Java Platform. For additional information, see Internationalization Enhancements.

1.4 Networking

These changes affect networking applications.

1.4.1 URL Connection Processing

Prior to 1.4, `URLConnection.getInputStream` threw a `FileNotFoundException` if the file type was known and the response code was greater than or equal to 400. Otherwise no exception would be thrown.

As of 1.4, the correct behavior is implemented. `URLConnection.getInputStream` throws an `IOException` for all `http` errors regardless of the file type. In throws `FileNotFoundException` (a sub-class of `IOException`) only when the http response indicates that the resource was not found. In other words, the `FileNotFoundException` is thrown only if the response code is 404 or 410.

In addition:

- `HttpURLConnection.getErrorStream` can be used to read the error page returned from the server. Prior to 1.4, getErrorStream() always returned null.
- The `HttpURLConnection.getResponseCode` method works correctly.

1.4.2 URI Format

Beginning with version 1.4.2, class `java.net.URI`, a hostname in the host component of a hierarchical URI that comprises only a single domain label can start with a digit. Previously, a URI such as `s://123/p` would not have its authority component parsed as a server-based authority and it would thus be considered a registry-based authority. As part of this change, the specification for `URI.getHost()` has been revised. The updated specification now reads: "The rightmost label of a domain name consisting of two or more labels begins with an alpha character".

1.5 Security

This section describes security changes.

1.5.1 Java™ Secure Socket Extension (JSSE)

In 1.4, the system property `com.sun.net.ssl.dhKeyExchangeFix` has a default value of `true`. Previously the JSSE 1.0.2 optional package defaulted to `false`. Cipher suites that use the DH key exchange are affected by this change in property value. (Such suites are used rarely.)

The change was made because the original optional had a bug that caused incorrect encoding of DSA signatures when those signatures were used as part of the server key exchange message. This bug meant that JSSE was not in conformance to the key exchange specification, and it was the source of incompatibilities between JSSE and SSL implementations from other vendors. The system property was introduced to remedy situation, and the default value was `false` for compatibility with previous releases. As of 1.4, the default value is `true` for compatibility with other SSL/TLS implementations.

1.5.2 System Property for Encoding of Policy File

As of 1.4.2, a new system property was introduced: `sun.security.policy.utf8`. If this system property is set to `true`, the policy file is read in using UTF-8 (1.4.0 and 1.4.1 behavior). If the system property is set to `false`, the policy file is read in using the default encoding (pre-1.4.0 behavior). When the system property is not set, the default value is `true`.

Prior to J2SE 1.4.0, the character encoding scheme for security policy files was unspecified, and the files were read in using the default character encoding. Starting in 1.4.0, the policy files were required to be encoded in UTF-8. While this requirement allowed a policy file to be used across different locales, it broke existing policy files that contained characters in the default encoding.

1.5.3 Serializing cryptographic Key objects

A new interface was added to J2SE 5.0, `java.security.KeyRep`, which represents the standard serialized representation for cryptographic `Key` objects. Existing serialized `Key` objects will continue to be serializable and deserializable within the same vendor virtual machine, but are not deserializable across different vendor virtual machines. This behavior has not changed.

A new `Key` class implementation that uses `KeyRep` as its serialized representation can be serialized and deserialized across different vendor virtual machines, as well as within a single vendor's virtual machine.

However, a Key class that uses `KeyRep` as its serialized representation can not be deserialized in any Java version prior to 5.0. As of 5.0, all Key classes implemented in Sun's cryptography providers were modified — use `KeyRep` for their serialized representation.

1.5.4 KerberosKey.serialVersionUID

As of 5.0, the `javax.security.auth.kerberos.KerberosKey` class defines its own private `serialVersionUID field`. This new field now hides the `serialVersionUID` field previously inherited from the `java.security.Key` interface, which `KerberosKey` implements.

The change does not introduce any runtime issues. Previously compiled code that references `KerberosKey.serialVersionUID` runs correctly. However, the change does introduce a source incompatibility, since application code that references `KerberosKey.serialVersionUID` won't compile in 5.0.

1.6 Serialization

This section describes serialization incompatibilities.

1.6.1 Serial Version UID Changes

The computed value of the default serial version UID for both the nested class and its enclosing class changed between 1.3 and 5.0. In 5.0, references to a class literal produce a bytecode instruction. Previously, they produced a reference to a static method. As a result of the change, any serializable class that references a class literal (for example, `Foo.class`) won't work when running in the 5.0 VM.

To isolate your code from changes in compilers between versions (or from different vendors), add an explicit serial version UID to your serializable classes. (Use the `serialver` tool to obtain the serial version UID of classes compiled with the 1.3 `javac` compiler.)

For more information, see bug 4786115.

1.6.2 Serializable Permission Required for Stream I/O Subclasses

1. If you have created a subclass of `ObjectOutputStream` that overrides putFields() and the subclass invokes `ObjectOutputStream`'s public one-argument constructor, you need to ensure that the `SerializablePermissionenableSubclassImplementation` is active in the current context. (As a security precaution, `ObjectOutputStream` checks with the `SecurityManager` to be sure that the permission is present.)
2. If you have created a subclass of `ObjectOutputStream` that overrides readFields() and the subclass invokes `ObjectOutputStream`'s public one-argument constructor, you need to ensure that the `SerializablePermissionenableSubclassImplementation` is active in the current context. (As a security precaution, `ObjectOutputStream` checks with the `SecurityManager` to be sure that the permission is present.)

1.6.3 Method Inheritance

In 1.3, the javac bytecode compiler used `-target 1.1` by default. In 5.0, the default is `_target 5.0`. One result of this change is that the compiler no longer generates and inserts method declarations into class files when the class inherits unimplemented methods from interfaces. These inserted methods, like all other non-private methods, are included in the default `serialVersionUID` computation. As a result, if you define an abstract serializable class which directly implements an interface but does not implement one or more of its methods, then its default serialVersionUID value will vary depending on whether it is compiled with 1.4 version of javac or a previous javac.

For more information on the `target` options, see the javac compiler's Reference Page. For background information on the methods inserted by earlier versions of javac, see bug 4043008.

1.7 Swing

This section describes compatibility issues that affect Swing GUI components.

1.7.1 Button Colors

Buttons with a customized background color might require code changes to be rendered as intended with the 5.0 Java look and feel theme, Ocean. Ocean draws a gradient on buttons, by default. If you don't want the gradient, either set the `contentAreaFilled` property to true or set the background to a `Color` that is not a `UIResource`. In most cases this is as simple as:

```
button.setBackground(Color.RED);
```

If, for some reason, you are picking up a `UIResource` you can create a new `Color` that is not a `UIResource` like this:

```
button.setBackground(new Color(oldColor));
```

For more information, see bug 4908404.

1.7.2 DefaultTreeModel

Beginning with version 1.4, javax.swing.tree.DefaultTreeModel allows a null root node. In previous versions, DefaultTreeModel did not allow a null root.

1.7.3 DefaultHighlighter.DefaultPainter

As of 1.4, public static field DefaultPainter in class javax.swing.text.DefaultHighlighter is final. Previously, it was non-final.

1.7.4 Drag and Drop

As of 1.4, Swing supports drag and drop (DnD). Applications that use AWT's drag and drop support on Swing components (specifically `DropTarget`) may experience conflict with Swing's `DropTarget`. For more information, see bug 4485914.

New applications should always use Swing's built-in DnD support, instead of customized solutions, because Swing's built-in support deals with the details of initiating drag gestures and showing the drop location.

For more information on Swing's drag and drop support refer to:

http://java.sun.com/j2se/1.5.0/docs/guide/swing/1.4/dnd.html

For a full tutorial, see:

http://java.sun.com/docs/books/tutorial/uiswing/misc/dnd.html

1.7.5 Focus Changes

The lightweight Swing component library has been modified for increased consistency and interoperation with AWT. As of 1.4, the default focus traversability for all Components is `true`. Previously, the focus traversability of lightweight Swing components defaulted to `false`.

1.7.6 JTable Indexing

In `JTree` and `JList` keyboard actions have always manipulated the lead index. For example, if the lead is

on row four in a `JList` and the user presses the up key, the lead moves to row three, selecting the third item. With these components, the lead is considered the focused index. The components tell their renderers to draw the focus indicator for a given index when that index is the lead.

As of 5.0, `JTable` operation is consistent with `JList` and `JTree`. Previously, `JTable` was doing the opposite and using the anchor index in the same manner that `JTree` and `JList` use the lead. A request to correct this inconsistency was made as RFE number 4759422 and eventually fixed as part of 4303294.

This change affects developers that assumed the previous behavior, so an application that uses the `JTable` anchor to determine which cell is being shown as the focused cell could behave incorrectly. For more information, see bugs 4759422 and 4303294.

1.7.7 Look & Feel Support for XP and GTK

Swing added support for XP and GTK look and feels in 1.4.2. If you are using the `WindowsLookAndFeel`, either directly by way of its class name, or indirectly by way of `UIManager.getSystemLookAndFeelClassName()`, you automatically get an XP look and feel when running on Windows XP. You get the GTK look and feel on a machine running Gnome.

Supporting both the XP and GTK look and feels posed a number of potential incompatibilities with how Swing has typically drawn widgets:

- Both look and feels typically use images to render widgets-so replacing a Border now often times results in no visual change. Previously replacing a Border would result in no border being drawn.
- For a `JButton`, painting of the background image is now conditional on the `contentAreaFilled` property, so if you invoke `setContentAreaFilled(false)` on a `JButton` you get a flat button with no image.
- Because the background is drawn from an image, changing the background color has no visual effect. This behavior can also be changed using the `contentAreaFilled` property.
- In Swing an opaque component always paints its background by way of `ComponentUI`'s update method. But a non-opaque component in GTK may or may not paint the background, depending upon the engine. To correctly provide a GTK look and feel a number of Components that previously were opaque have been made transparent (by way of the `ComponentUI` subclass's `installUI` method). You may see problems with components if you expect them to be of a certain opacity rather than setting the desired value. You may also see problems if you are expecting `setOpaque(false)` to indicate that a background should not be painted.

1.8 XML Processing

JAXP 1.1 (Crimson) was part of the 1.3 platform. JAXP 1.3 (Xerces) is part of the 5.0 platform. These are two entirely different implementations, accessed by a common API-JAXP.

Although Crimson was small and fast, it was ultimately less functional than Xerces — an open-source implementation hosted at Apache. In addition, the JAXP standard has evolved from 1.1 to 1.3. Those two factors combine to create compatibility issues.

Since JAXP 1.1 was also part of the 1.4 platform, the JAXP Compatibility Guide for 5.0 (which discusses the migration from 1.4 to 5.0) also covers the issues that arise when converting a 1.3 XML-processing application to 5.0. Consult that guide for additional detail on the changes described here.

1.8.1 DOM

As of 5.0, JAXP supports the DOM Level 3 family of APIs.

- New methods have been added to the DOM interfaces, so some existing applications will not be able to com pile with the new interfaces.
- Some applications will also encounter a `NoSuchMethodException` at runtime that can only be resolved by recompiling the sources against the 5.0 libraries.
- The way whitespace is handled differs between the two libraries, so applications that expect to write out readable, "pretty-printed" XML need to be modified to accommodate the differences.

1.8.2 SAX

As of 5.0, JAXP supports SAX 2.0.2. In general, SAX 2.0.2 is a bug-fix release, with no API changes. However, a few clarifications implemented in the SAX 2.0.2 release may create compatibility issues:

- `ErrorHandler, EntityResolver, ContentHandler`, and `DTDHandler` can now be set to null by applications. SAX 2.0 required the XML processor to throw `java.lang.NullPointerException` in this case. (The JAXP parser implemented in 5.0, like most implementations, reacts to null by using the default settings.)
- The `resolveEntity` method in `DefaultHandler` and the `EntityResolver` subclass throws `IOException` and `SAXException`. Previously it threw only `SAXException`. Code that invokes `resolveEntity` needs to be modified to handle `IOException`, as well as `SAXException`.

1.8.3 XSLTC

As of 5.0, XSLTC is the default transformer, because the Apache community decided to make XSLTC the default processor for developing XSLT 2.0. Previously the default transformer was Xalan. Compatibility issues include:

- Xalan has bugs that XSLTC does not, and vice-versa. Application code that depends on the behavior of Xalan bugs is likely to fail.
- XSLTC does not support all the extensions that Xalan does. These extensions are beyond the definition of the JAXP and XSLT specifications. To work around that problem, you can download and use the Xalan classes from Apache. (Going forward, however, you can expect to see more and more of the extensions supported in XSLTC.)
- An application that explicitly uses the Xalan XPath API to evaluate a standalone XPath expression (one that is not part of an XSLT stylesheet), needs to be recoded to use the standard XPath API included in JAXP 1.3. Alternatively, the application can include Xalan libraries downloaded from Apache.

1.8.4 Security Enhancements

New system and parser properties have been added to address security issues:

- A new secure processing feature lets the application configure the SAXParserFactory or DocumentBuilderFactory to get a secure XML processor. Setting this feature to true sets the entity expansion limit to 64000 to prevent denial of service attacks.
- Alternatively, the `entityExpansionLimit` can be used to constrain the total number of entity expansions
- The `disallow-doctype-decl` parser property prevents an incoming XML document from containing a DOCTYPE declaration.

1.8.5 Package Name Changes

In 5.0, the org.apache classes, have moved in 5.0 to com.`sun.org.apache.package.internal` so that they won't clash with more recent, developer-downloaded versions of the classes.
This change does not affect applications that confine themselves to using the standard JAXP APIs. It does affect applications that depend on implementation-specific features:

- The property-values that were used to access the internal implementations must be changed.
- Applications that used internal APIs from the Xalan implementation classes must change the import statements that gave them access to those APIs.
- Applications that used internal APIs from other Crimson libraries must take into account the new package names by:

  a. Recoding the application so it uses only the supported interfaces that are part of JAXP.

  b. Including the org.apache classes downloaded from Apache in the classpath.

1.9 Other Runtime Changes

These changes are of interest to the developer of end-user applications.

1.9.1 CORBA

Changes were made to the 1.4 CORBA APIs to make them compliant with the CORBA 2.3 mapping as specified by the OMG documents referenced in CORBA Compatibility Information. Follow the link for information on the API changes, as well as a listing of all OMG specifications with which J2SE 1.4.0 complies.

1.9.2 Default Encoding for non-ANSI Files (Windows)

As of J2SE 1.4.2, the Microsoft Windows `file.encoding` system property is derived from the system default locale. As a result, applications that assume `utf-16le` encoding as the default for non-ANSI codepage locales will fail. Such applications should use the `file.encoding` system property, instead.

The change was made to fix bug 4459099. Previously, if the user's locale setting had no corresponding ANSI code page in the Control Panel (for example, Hindi), the `file.encoding` system property was set to `utf-16le`. All readers and writers would then default to using that encoding, and exceptions were generated as files on the system were read and written using the `utf-16le` converter.

1.9.3 HTML Forms

As of 1.4, HTML forms are modeled differently than they were in 1.3. Previously, attributes of a form were stored in the `attributeset` of the children character elements. Now, an element is created to represent the form. The new element matches the content of the html file, which allows for better modeling of the form and consistent writing of the form. This change was made to address bug 4200439.

This change only affects developers who relied on forms being handled loosely. If you had previously been expecting the attributes of the leaf Elements to contain the Form's attributes, you now have to get the attributes from the Form Element's `AttributeSet`.

For example, pre-1.4.0 implementations previously treated the following invalid html

```
<table>
   <form>
</table>
</form>
```

as:

```
<form>
   <table>
   </table>
</form>
```

But as of 1.4, it is instead treated as:

```
<table>
   <form>
   </form>
</table>
```

1.9.4 java.vm.info property (added value)

To reflect the class sharing feature introduced in 5.0, the `java.vm.info` property, which is reflected in the text displayed by `java -version`, now specifies the sharing mode. Any code that parses all the way to the end of the `java.vm.info` property value or the output of `java -version` might need to be changed. For more information, see bug 4964160 and Class Data Sharing.

1.9.5 Java I/O Changes

These changes affect subclasses of ObjectInputStream and ObjectOutputStream which override the putFields or readFields methods when those subclasses do not also override the rest of the serialization infrastructure.

1. Beginning with J2SE 1.4.0, ObjectOutputStream's `public one-argument constructor` requires the enableSubclassImplementation SerializablePermission when invoked (either directly or indirectly) by a subclass which overrides `ObjectOutputStream.putFields` or `ObjectOutputStream.writeUnshared`.
2. Also beginning with J2SE 1.4.0, ObjectInputStream's public one-argument constructor requires the enableSubclassImplementation SerializablePermission when invoked (either directly or indirectly) by a subclass which overrides ObjectInputStream.readFields or ObjectInputStream.readUnshared.

1.9.6 Java™ DataBase Connectivity (JDBC™) / BigDecimal API Change

A BigDecimal method changed its behavior between 1.4 and 5.0, causing JDBC drivers to malfunction. To resolve the problem, use the 5.0 version of the JDBC driver.

1.9.7 JDBC Time / Date Comparisons

As of 5.0, comparing a `java.sql.Timestamp` to a `java.util.Date` by invoking `compareTo` on the `Timestamp` results in a `ClassCastException`. For example, the following code successfully compares a `Timestamp` and `Date` in 1.4.2, but fails with an exception in 5.0:

```
aTimeStamp.compareTo(aDate)  //NO LONGER WORKS
```

This change affects even pre-compiled code, resulting in a binary compatibility problem where compiled code that used to run under earlier releases fails in 5.0. The problem is expected to be fixed in a future release.

For more information, see bug 5103041.

1.9.8 Logging

Previously, the `java.util.logging.Level(String name, int value, String resourceBundleName)` constructor allowed a null name argument, but the parse method did not. In 5.0, the constructor now throws a `NullPointerException` when the name is null. The compatibility risk is mitigated in that you had to

subclass Level to use this constructor and would get a `NullPointerException` when using a Level name of null for subsequent calls, except for simple calls such as `toString()`.

For more information, see bug 4625722.

Chapter 2

Deployment Issues

This section summarizes deployment issues. For more detailed information, consult the Java Deployment Guide.

2.1 Applets

2.1.1 Java Control Panel

A new Java Control Panel, introduced in 5.0, consolidates the Java Plug-in Control Panel and the Java Web Start Application Manager, providing a single configuration interface. The compatibility impact pertains to Applet Caching, discussed in the next section. For other information on the Java Control Panel, see http://java.sun.com/j2se/1.5.0/docs/guide/deployment/deployment-guide/jcp.html

2.1.2 Applet Caching Changes

As of 5.0, applets have an independent cache that is shared across browsers. Previously, they were cached in each browser, so there was likely to be a cached copy of the applet for each browser (or browser version) used on a given machine.

As a result of the change, most aspects of applet caching are managed from the Java Control Panel, instead of from the browser. Those aspects include:

- Cache location
- Cache size limit
- Cache compression
- Cache management tool
- Cache removal policy

As a result, browser settings for these values have no effect on applet caching in 5.0.

2.1.3 Certificate Verification for a Signed Applet

In 5.0, the root Certificate Authority (CA) certificates used for signature verification come from:

- the Java™ runtime environment (JRE) `cacerts` file (always enabled)
- the `browser` keystore (enabled by default, but can be disabled in the Java Control Panel)

Previously, signature verification used root CA certificates from the browser.

2.1.4 Timestamped Applet Signatures

As of 5.0, deployed Java™ Archive (JAR) files no longer have to be re-signed annually. Instead, `jarsigner` can be used to generate signatures that include a timestamp. Systems and deployment programs (like Java Plug-in) can then use the new APIs that obtain timestamp information to see if the JAR file was signed while the signing certificate was still valid.

Previously, the signature generated by`jarsigner` did not contain a timestamp. With no other information available, systems and deployment programs generally checked the validity of the signing certificate to confirm the validity of a signed JAR file. But that check failed when the signing certificate expired, which typically happened every year.

For more information, see http://java.sun.com/j2se/1.5.0/docs/guide/security/time-of-signing-beta1.html

2.2 Libraries

The following packages are now part of the Java 5.0 release, so they no longer have to be distributed as optional packages:

- Java Secure Socket Extension (JSSE)
- Java™ Authentication and Authorization Service (JAAS)
- Java™ Cryptography Extension (JCE)
- Java Web Start (JWS)
- Java™ Management Extensions (JMX™)

Note: Use the `extcheck` utility included in the release to detect version conflicts between a target jar file and currently installed extension jar files.

2.3 Installation

2.3.1 Windows Online Installer

As of 5.0, a new “online installation option” is available. It’s excellent for use with a fast connection. With a slow connection, it starts rapidly and it downloads fewer bytes overall, but when the installer begins downloading data, the progress indicator only advances when a complete cab file has been downloaded. There is no indication of how long any given download will take so, for a slow connection, the offline installation option is often a better choice. Even though the total size of the download is larger, your browser’s progress bar can estimate the likely completion time.

2.3.2 Name Changes

As of 5.0, the names for directories, bundles, packages, registries, and Linux RPMs have been changed, so scripts that depend on the old pathnames need to be updated to reflect the new names. The new naming conventions are as follows:

Old Name	New Name
j2se	java
j2re	jre
j2sdk	jdk

Capitalization is in accordance with platform conventions. Additional platform-specific details are shown on the next page. For more information, see J2SE Naming and Versioning and J2SE 5.0 Name and Version Change.

Solaris

The Java™ Development Kit (JDK™) packages install in:

```
/usr/jdk/jdk<version>
```

The prefix used for all packages has changed from “SUNWj3” (used in 1.3 and 1.4) to “SUNWj5”.

Linux

The JRE and JDK RPMs install in:

```
/usr/java/jre<version>
/usr/java/jdk<version>
```

The RPM database name is the value displayed in the Name field when doing an RPM query. This value is appended to the Version and Release fields to get the fully qualified name — for example, `jre-1.5.0-fcs`. The RPM database can be queried to determine what is provided by a given package. The JRE and the JDK both provide “jre” as part of the new naming convention, as well as “j2re” for backwards compatibility. The JDK also provides “jdk” and, for backwards compatibility, “j2sdk”. Sun provides the old names in accordance with standard EOL policy, but new scripts and RPMs should use the new convention.

UNIX

The tarball expands to:

```
./jre<version>
./jdk<version>
```

Microsoft Windows

The JRE and JDK install in:

```
%ProgramFiles%\Java\jre<version>
%ProgramFiles%\Java\jdk<version>
```

The registry keys haven’t changed. They continue to use the full names of “Java Runtime Environment” and “Java Development Kit”.

2.4 Virtual Machine (Solaris)

As of 5.0, server-class Solaris/SPARC machines run the server VM by default, rather than the client VM. In general, the throughput of the server VM is much better, but the startup time is somewhat worse. Previously, the default virtual machine (VM) for Solaris/SPARC was the client VM. However, many Solaris/SPARC boxes are used as servers, for which the server VM delivers better performance.

Note: A “server-class machine” is currently defined to be one with 2 or more processors and 2 or more gigabytes of memory. For more information, see Server-Class Machine Detection and Garbage Collection Ergonomics .

Chapter 3
Compilation Issues

These are issues you'll run into when compiling your 1.3 code to run in 5.0.

3.1 API Changes

3.1.1 JDBC

The JDBC 3.0 API that is part of the 1.4 platform introduces two new interfaces and adds several new methods to existing interfaces. Although drivers and applications compiled under 1.3 will run with out problem, the sources will not compile due to these changes. Drivers and applications that implement the JDBC interfaces must be therefore updated to reflect the changes in order to compile successfully. For a complete list of requirements, see Chapter 6 of The JDBC 3.0 Specification.

3.1.2 New Proxy Class

The `java.net.Proxy` class was added in 5.0, making two classes named `Proxy`:

- java.lang.reflect.Proxy
- java.net.Proxy

The introduction of the new class prevents existing code from compiling when the following conditions are met:

- It has the following import declarations:

```
import java.lang.reflect.*;
import java.net.*;
```

- There is no import declaration that specifically imports one of the Proxy classes.
- The code refers to the `Proxy` class by its simple name, rather than using a fully-qualified name like `java.lang.reflect.Proxy`.

In this case, a compile-time error occurs, because the reference is ambiguous.

To resolve the ambiguous reference in favor of `java.lang.reflect.Proxy.`, add a third import statement:

```
import java.lang.reflect.Proxy;
```

With this third import statement in place, the source code will compile and have the same behavior as in previous versions.

3.1.3 Socket API / SocketImpl Subclasses

As of 1.4 the Socket API added a new abstract method to the `SocketImpl` abstract class. Because of the new method, a subclass of `SocketImpl` created prior to 1.4 fails to compile because there is no implementation for the new method. (The binary will run as expected, however.)

Few applications subclass `SocketImpl`. But for those that do, there are two ways to deal with this change:

- Use a class file compiled on J2SE 1.3.x (or earlier).
- Provide an implementation for the new methods. The following code provides a simple example. The implementation of `connect()` assumes a straight TCP/IP implementation; and it ignores the timeout value. (Modify it to respect the timeout value appropriate for your application.) The implementation for `sendUrgentData()` simply throws an exception. It's all you need, provided that supports `UrgentData()` hasn't been overwritten to return "`true`".

```
/**
 * Creates a socket and connects it to the specified address on
 * the specified port.
 * @param address the address
 * @param timeout the timeout value in milliseconds,
 *        or zero for no timeout.
 * @throws IOException if connection fails
 * @throws  IllegalArgumentException if address is null or is a
 *          SocketAddress subclass not supported by this socket
 * @since 1.4
 */
protected void connect(SocketAddress address, int timeout)
throws IOException {
   if (address == null
   || !(address instanceof InetSocketAddress))
      throw new IllegalArgumentException(
"unsupported address type");
   InetSocketAddress addr = (InetSocketAddress) address;
   if (addr.isUnresolved())
      throw new UnknownHostException(addr.getHostName());
   this.port = addr.getPort();
   this.address = addr.getAddress();
   try {
      connect(this.address, port);
      return;
   } catch (IOException e) {
      // everything failed
      close();
      throw e;
   }
}
/**
 * Send one byte of urgent data on the socket.
 * The byte to be sent is the low eight bits of the parameter
```

```
     * @param data The byte of data to send
     * @exception IOException if there is an error sending the data.
     * @since 1.4
     */
    protected void sendUrgentData (int data) throws IOException {
        throw new IOException("Unsupported operation");
    }
```

3.2 Generics

As of 5.0, the collection classes, the `Class` class, and other core libraries have been generified by adding generic type parameters and arguments to existing classes and methods. Taking advantage of generics produces cleaner, more readable code, and eliminates one source of runtime errors by making it unnecessary to cast a value to declare its type to the compiler.

Most existing source code will compile successfully when using the generified libraries in 5.0, but some will not. The simplest workaround for code that fails to compile due to the generification changes (or simply to avoid warnings) is to specify `-source 1.4` on the `javac` command line.

Note: Many of today's IDEs can automatically convert existing code to make full use of the generified libraries. The conversion operation is generally provided as part of their refactoring capabilities. For information about generics and the generification of the core libraries, see JSR 14 and the generics tutorial (PDF).

3.3 New Reserved Words

The following words were added to the Java language between 1.3 and 5.0, so they are no longer available for use as field or method identifiers:

- `assert` (added in 1.4)
- `enum`

Class files are not affected by this change but, because the keywords are reserved, existing programs that use the new keywords as an identifier won't compile, unless they are compiled with a compatibility switch:

- `-source 1.3`: Disables all of the keywords, allowing them to be used as identifiers.
- `-source 1.4`: Enables the `assert` keyword. Your program can use asserts, but cannot use "assert" as an identifier. Other keywords are disabled, and may be used as identifiers.
- `-source 1.5 (default)`: Enables all of the keywords and prevents their use as identifiers.

Note: Support for 1.3 source compatibility is likely to be phased out over time.
If any option other than `-source 1.5` is specified, all 5.0 language features are disabled. For more information on those features, see the Java Programming Language Enhancements. That page is an index of tutorials and whitepapers. It's highly recommended for anyone who wants to produce the most elegant, readable, and maintainable code possible.

3.4 Compiler Changes

3.4.1 Default Target Change

In 1.3, the javac bytecode compiler used `-target 1.1` by default. In 5.0, the default is `_target 5.0`. This change affects applications that are intended to run on the 1.1 version of the Java platform. For more information on the `target` options, see the javac compiler's Reference Page.

3.4.2 Stricter Adherence to the Language Spec

As of 1.4, the Javac bytecode compiler in J2SE 1.4.0 became more strict in enforcing compliance with the Java Language Specification. As a result, existing code that does not strictly conform to the Java Language Specification may not compile, even though it may have compiled in earlier versions.

Here are two examples of situations in which the compiler is stricter:

- The compiler now detects unreachable empty statements as required by the language specification. Here are two examples of fairly common cases that the compiler now rejects:

  ```
  return 0;/* exit success */;
  ```

  and:

  ```
  {
    return f();
  } catch (Whatever e) {
    throw new Whatever2();
  };
  ```

  In each case, the compiler now correctly regards the extra semicolon as an unreachable empty statement.

  Note: Some automatically generated source code may generate unreachable empty statements.

- The compiler now rejects import statements that import a type from the unnamed namespace. Previous versions of the compiler accepted such import declarations, even though they were arguably not allowed by the language, because the type name appearing in the import clause is not in scope. (The specification has been clarified to state clearly state that you cannot have a simple name in an import statement, nor can you import from the unnamed namespace.)

  As a result, this syntax is no longer legal:

  ```
  import SimpleName;
  ```

  Nor is this syntax, which would import a nested class from the unnamed namespace:

  ```
  import ClassInUnnamedNamespace.Nested;
  ```

  To fix such problems in your code, move all of the classes from the unnamed namespace into a named namespace.

Chapter 4

Changes that Affect Tool Developers and Platform Implementers

These changes generally affect tool developers and Java platform implementers, rather than application developers.

4.1 Applet Data Streaming / Container Implementations

As of 1.4, applet container classes (classes that implement the `AppletContext` interface, such as those in Java Plug-in and `appletviewer`) have to be modified to implement the revised `AppletContext` API.

The revised specification for `AppletContext` lets applet developers stream data and objects for persistent use during a browser session. This change eliminates the need for developers to use static classes to cache data and objects, but it introduces a binary incompatibility for applet containers

4.2 Class Files / Inner Classes / Instrumented Code

The class file format changed in 5.0. As a result, programs that instrument class files to take performance measurements or to perform debugging generate invalid classes.

The names generated for inner classes also changed, affecting instrumentation programs and other programs that identify inner classes by their naming pattern.

4.3 Class Initialization after Evaluating a Class Literal

As of 5.0, evaluating a class literal (for example, `Foo.class`) does not cause the class to be initialized. Previously, it did.

The new behavior is a consequence of the fact that the VM now supports class literals in the constant pool. The old behavior remains in classes compiled with a pre-5.0 compiler or with the `-target 1.4` flag, even if run in the 5.0 VM.

Code that depends on the previous behavior should be rewritten like this:

```
//... Foo.class ...          //OLD CODE
... forceInit(Foo.class) ... //NEW CODE
```

```
/**
* Forces the initialization of the class pertaining to
* the specified <tt>Class</tt> object. This method does
* nothing if the class is already initialized prior to
* invocation.
*
* @param klass the class for which to force initialization
  * @return <tt>klass</tt>
*/
public static <T> Class<T> forceInit(Class<T> klass) {
    try {
       Class.forName(klass.getName(), true,
                     klass.getClassLoader());
    } catch (ClassNotFoundException e) {
              throw new AssertionError(e);  // Can't happen
    }
    return klass;
  }
```

For more information, see Initialization of Classes and Interfaces (section 12.4) in The Java Language Specification.

Note: The language specification hasn't changed; it never listed class literal evaluation as an initialization trigger.

4.4 ClassLoader Method Arguments

Previously, it was possible to specify a non-binary class name to `ClassLoader` methods that take a `String` class name argument. This unintended behavior was not compliant with the long-standing specification of class names. As of 5.0, parameter checking of these `ClassLoader` methods has been modified to comply with the specification, and any class name that is not a binary name is treated like any other unrecognized class name.

Since the APIs that explicitly require or return class names (for example, `Class.forName` or `Class.getName`) use the binary name for reference types, this change affects few developers. For more information, see the definition of binary name in the Java Language Specification, Second Edition. Also see the evaluation of bug 4986512.

4.5 Debugging and Profiling APIs

1. As of 5.0 the Java Virtual Machine Debug Interface (JVMDI) is deprecated. **JVMDI will be removed in the next major release**. Any new development should use JVMTI. Existing tools should begin moving to JVMTI.
2. As of 5.0 the Java Virtual Machine Profiling Interface (JVMPI) is deprecated. **JVMPI will be removed in the next major release**. Any new development should use JVMTI. Existing tools should begin moving to JVMTI.
   For more information on these changes, see the JVMTI documentation.

Chapter 5

References

The AWT Focus Subsystem
http://java.sun.com/j2se/1.5.0/docs/api/java/awt/doc-files/FocusSpec.html

Bug Reports
http://developer.java.sun.com/developer/bugParade/bugs/bugNumber.html

Class Data Sharing
http://java.sun.com/j2se/1.5.0/docs/guide/vm/class-data-sharing.html

CORBA Compatibility Information
http://java.sun.com/j2se/1.4/compatibility-CORBA.html

Garbage Collection Ergonomics
http://java.sun.com/j2se/1.5.0/docs/guide/vm/gc-ergonomics.html

Generics Tutorial
http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf

High Performance Graphics
http://java.sun.com/products/java-media/2D/perf_graphics.html

Internationalization Enhancements
http://java.sun.com/j2se/1.5.0/docs/guide/intl/enhancements.html

Java 1.3 Drag and Drop API Specification
http://java.sun.com/j2se/1.3/docs/api/java/awt/dnd/DropTargetListener.html#dragExit(28java.awt.dnd.DropTargetEvent)

Java2D System Properties
http://java.sun.com/j2se/1.5.0/docs/guide/2d/flags.html

Java 5.0 API Specification
http://java.sun.com/j2se/1.5.0/docs/api/

Java 5.0 Name and Version Change
http://java.sun.com/j2se/j2se-namechange.html

Java Database Connection (JDBC) 3.0 Specification
http://java.sun.com/products/jdbc/download.html

Java Deployment Guide
http://java.sun.com/j2se/1.5.0/docs/guide/deployment/deployment-guide/contents.html

Java Naming and Versioning
http://java.sun.com/j2se/naming_versioning_5_0.html

Java Programming Language Enhancements
http://java.sun.com/j2se/1.5.0/docs/guide/language/index.html

Java Language Specification, Second Edition
http://java.sun.com/docs/books/jls/

Java API for XML Processing (JAXP)
http://java.sun.com/xml/jaxp/index.jsp

JAXP 1.1 (Crimson)
http://xml.apache.org/crimson/

JAXP Compatibility Guide for 5.0
http://java.sun.com/j2se/1.5.0/docs/guide/xml/jaxp/JAXP-Compatibility_150.html

JAXP Secure Processing
http://java.sun.com/j2se/1.5.0/docs/api/javax/xml/XMLConstants.html#FEATURE_SECURE_PROCESSING

JSR 14: The Java Specification Request for Generics
http://jcp.org/en/jsr/detail?id=14

JVMTI documentation
http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/index.html

Modified UTF-8
http://java.sun.com/developer/technicalArticles/Intl/Supplementary/#Modified_UTF-8

New Java2D Features in 1.4
http://java.sun.com/j2se/1.4.2/docs/guide/2d/new_features.html

New Java2D Features in 5.0
http://java.sun.com/j2se/1.5.0/docs/guide/2d/new_features.html

Print Formatting API
http://java.sun.com/j2se/1.5.0/docs/api/java/util/Formatter.html

Reference Page for Tools and Utilities, including the javac compiler
http://java.sun.com/j2se/1.4.2/docs/tooldocs/tools.html

Server-Class Machine Detection
http://java.sun.com/j2se/1.5.0/docs/guide/vm/server-class.html

Standard End-of-Life (EOL) Policy
http://java.sun.com/products/archive/eol.policy.html

Supplementary Characters in the Java Platform (excellent Unicode tutorial)
http://java.sun.com/developer/technicalArticles/Intl/Supplementary/

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 1-650-960-1300 or 1-800-555-9SUN **Web** sun.com

The Network is the Computer™

Sun Worldwide Sales Offices: Argentina +5411-4317-5600, Australia +61-2-9844-5000, Austria +43-1-60563-0, Belgium +32-2-704-8000, Brazil +55-11-5187-2100, Canada +905-477-6745, Chile +56-2-3724500, Colombia +571-629-2323 Commonwealth of Independent States +7-502-935-8411, Czech Republic +420-2-3300-9311, Denmark +45 4556 5000, Egypt +202-570-9442, Estonia +372-6-308-900, Finland +358-9-525-561, France +33-134-03-00-00, Germany +49-89-46008-0, Greece +30-1-618-8111, Hungary +36-1-489-8900, Iceland +354-563-3010, India–Bangalore +91-80-2298989/2295454; New Delhi +91-11-6106000; Mumbai +91-22-697-8111, Ireland +353-1-8055-666, Israel +972-9-9710500, Italy +39-02-641511, Japan +81-3-5717-5000, Kazakhstan +7-3272-466774, Korea +822-2193-5114, Latvia +371-750-3700, Lithuania +370-729-8468, Luxembourg +352-49 11 33 1, Malaysia +603-21161888, Mexico +52-5-258-6100, The Netherlands +00-31-33-45-15-000, New Zealand–Auckland +64-9-976-6800; Wellington +64-4-462-0780, Norway +47 23 36 96 00, People's Republic of China–Beijing +86-10-6803-5588; Chengdu +86-28-619-9333; Guangzhou +86-20-8755-5900; Shanghai +86-21-6466-1228; Hong Kong +852-2202-6688, Poland +48-22-8747800, Portugal +351-21-4134000, Russia +7-502-935-8411, Saudi Arabia +9661 273 4567, Singapore +65-6438-1888, Slovak Republic +421-2-4342-94-85, South Africa +27 11 256-6300, Spain +34-91-767-6000, Sweden +46-8-631-10-00, Switzerland–German 41-1-908-90-00; French 41-22-999-0444, Taiwan +886-2-8732-9933, Thailand +662-344-6888, Turkey +90-212-335-22-00, United Arab Emirates +9714-3366333, United Kingdom +44-1-276-20444, United States +1-800-555-9SUN or +1-650-960-1300, Venezuela +58-2-905-3800, or online at sun.com/store

SUN™ Sun, Sun Microsystems, the Sun logo, Java, the Java Coffee Cup logo, Solaris, J2SE, JSSE, JAAS, JCE, JWS, JMX, JRE, JDK AND JDBC are trademarks, registered trademarks or service marks of Sun Microsystems, Inc. in the United States and other countries.

UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

Printed in USA 00/00 XX0000-0/#K