

# Introduction

Welcome to *Java 6 New Features: A Tutorial*.

Java 6, code-named Mustang, is the first Java release for which Sun Microsystems has invited outside developers to contribute code and help fix bugs. (See <https://mustang.dev.java.net/collaborate.html>). True that the company has in the past accepted contributions from non-employees, like the work of Doug Lea on multithreading, but this is the first time Sun has posted an open invitation. The company admits that they have limited resources, and outside contributors help them cross the finish line sooner.

This didn't make Mustang an open source project, though. Not yet, even though Sun has announced its plan to open source Java in the near future. The one reason Sun regularly cited in the past was the fear of incompatibility. Making Java open sourced, Sun argued, could lead to a situation where there were many different and incompatible versions of Java. A well-known case of this is how Microsoft, then a Java licensee, added Windows-specific features to Java, thus undermining the one feature that Java was best known for: portability.

Open-sourced or not, Java SE 6 is here. It is specified in JSR 270, "Java Standard Edition 6 Release Contents" (<http://jcp.org/en/jsr/detail?id=270>). Unlike other JSRs, however, JSR 270 does not define specific features. Rather, it acts as an "umbrella" JSR that enumerates features in other JSRs. Here are the "member" JSRs.

- JSR 105, XML Digital-Signature APIs
- JSR 173, Streaming API for XML (StAX)
- JSR 181, Web-Services Metadata
- JSR 199, Java Compiler API
- JSR 202, Java Class-File Specification Update
- JSR 221, JDBC 4.0

- JSR 222, Java Architecture for XML Binding (JAXB) 2.0
- JSR 223, Scripting for the Java Platform
- JSR 224, Java API for XML-Based Web Services (JAX-WS) 2.0
- JSR 250, Common Annotations
- JSR 269, Pluggable Annotation-Processing API

As there are a myriad of big and small changes involved, it is not possible to cover all in a single book. Therefore, I could only attempt to include the most important ones. The section “About This Book” later in this introduction provide more details about this book.

This introduction also provides two other important sections, “Java Naming Convention” and “Those New to Java 5.” The first talks briefly about Java history, especially with regard to the naming convention. The second is for those who have decided to skip Java 5 entirely. Admittedly, there are several important Java 5 new features that anyone attempting to upgrade to version 6 is strongly recommended to learn. This section provides necessary references. In addition, the appendixes explain the three most important new features in Java 5: enums, generics, and annotations

---

## Java Naming Convention

Sun Microsystems introduced Java in 1995 and Java—even though it had been a general-purpose language right from the start—was soon well known as the language for writing applets, small programs that run inside web browsers and add interactivity to static web sites. The growth of the Internet had much to contribute to the early success of Java.

Having said that, applets were not the only factor that made Java shine. The other most appealing feature of Java was its platform-independence promise, hence the slogan “Write Once, Run Anywhere.” What this means is the very same program you write will run on Windows, Unix, Mac, Linux, and other operating systems. This was something no other programming language could do. At that time, C and C++ were the two most commonly used languages for developing serious applications. Java seemed to have stolen their thunder since its first birthday.

That was Java version 1.0.

In 1997, Java 1.1 was released, adding significant features such as a better event model, Java Beans, and internationalization to the original.

Java 1.2 was launched in December 1998. Three days afterwards, the version number was changed to 2, marking the beginning of a huge marketing campaign that started in 1999 to sell Java as the “next generation” technology. Java 2 was sold in four flavors: the Standard Edition (J2SE), the Enterprise Edition (J2EE), the Micro Edition (J2ME), and Java Card (that never adopted “2” in its brand name).

The next version released in 2000 was 1.3, hence J2SE 1.3. 1.4 came two years later to make J2SE 1.4. J2SE version 1.5 was released in 2004. However, the name Java 2 version 1.5 was then changed to Java 5.

The official name for Mustang, the latest version, is Java Platform, Standard Edition 6. Note that 6 is the product version. The developers still often call it version 1.6, which therefore is the developer version.

### Note

See <http://www.java.com/en/javahistory/> and <http://java.sun.com/features/1998/05/birthday.html> for more detail on the history of Java. Java naming is also discussed in the article “Building and Strengthening the Java Brand” that can be found at <http://java.sun.com/developer/technicalArticles/JavaOne2005/naming.html>. For version 6, also see this link: <http://java.sun.com/javase/6/webnotes/version-6.html>

---

## Those New to Java 5

This book focuses on Mustang’s new features. However, Java 5 introduced many language changes that you should be familiar with to understand the new features in Java 6. One of such Java 5 changes is the enhanced **for**. Others are enums, generics, and annotations.

With the enhanced **for**, you can easily iterate over an array or a collection. Use this syntax to iterate over an array:

```
for (Type variable : arrayName)
```

Where *arrayName* is the reference to the array, *Type* is the component type of the array, and *variable* is a variable that references each component of the array.

For example, the following code iterates over an array of **Strings**.

```
String[] names = { "John", "Mary", "Paul" };
for (String name : names) {
    System.out.println(name);
}
```

The code prints the following on the console.

```
John
Mary
Paul
```

To iterate over a **Collection** without the need to call the **iterator** method, you can use this syntax.

```
for (Type identifier : expression) {
    statement(s)
}
```

In which *expression* must be an **Iterable**. Since **Collection** extends **Iterable**, you can use enhanced **for** to iterate over any **Collection**. For example, this code shows how to use **for** to iterate over a collection.

```
for (Object object : myList) {
    System.out.println(object);
}
```

Using **for** to iterate over a collection is a shortcut for using **Iterator**. In fact, the code that uses enhanced **for** above is translated into the following by the compiler.

```
for (Iterator iterator = myList.iterator(); iterator.hasNext(); ) {
    String element = (String) iterator.next();
    System.out.println(element);
}
```

As previously mentioned, other important changes in Java 5 include enums, generics, and annotations. Enums are discussed in Appendix A. Generics are covered in Appendix B and are important because many class and method signatures now show the presence of generics. For example, you need to master generics to understand the **<M>** part after the class name in the signature of **javax.swing.RowSorter**, a new class in Mustang.

```
public abstract class RowSorter<M> extends Object
```

Besides enums and generics, Java 5 introduced annotations and provided a small number of default annotation types. Java 6, however, adds dozens more. Naturally, to understand Mustang's new annotations you need to first know what annotations are. Appendix C, "Annotations" can help you.

---

## About This Book

This section presents the overview of each chapter.

Chapter 1, "Core Libraries" explains the new features in the core libraries, including the new **isEmpty** method in **java.lang.String**, the new methods in **java.io.File** for modifying file attributes and enquiring about hard-disk space, array reallocation, the new types in the Collections Framework, enhancements to the floating point, and password prompting using **java.io.Console**.

Chapter 2, "Dynamic Compilation" shows off the reference implementation for JSR 199, Java Compiler API. Also called the Java Compiler Framework, this new API features classes and interfaces in the **javac.tools** package. Chapter 2 introduces this technology and provides a few examples.

JSR 223, Scripting for the Java Platform enables collaboration between Java and scripting languages. Java 6 comes with a reference implementation that supports JavaScript, however other scripting languages are supported through the Scripting project. Chapter 3, "Scripting" presents the API and teaches how to use them.

Chapter 4, "Networking" explains networking-related features Mustang brings to the table. These includes the APIs to work with client-side cookies and a web server to test your web applications and web services. Plus, Mustang also adds types for working with internationalized domain names, internationalized resource identifiers, and interface addresses.

Chapter 5, "Swing Updates" teaches you the many new features in Swing that Mustang brings in. They range from new Windows and GTK look and feels to better support for Drag and Drop to **JTable** sorting and filtering.

Chapter 6, "Abstract Window Toolkit" explains the new features and improvements Java 6 brings to the AWT, including improved dialog modality,

splash screens, system tray support, desktop help, GIF writers, and text antialiasing.

Chapter 7, “Internationalization” discusses the internationalization and localization improvements in Mustang. These include the new supported locales, the Locale Sensitive Services SPI, the new methods in **ResourceBundle**, and the new class **ResourceBundle.Control**.

Chapter 8, “Java Database Connectivity 4.0” provides examples of how to use the new features in JDBC 4.0, including automatic driver loading, ease of development features, support for national character sets, the RowId type, and the **SQLXML** interface. Mustang includes the reference implementation for this latest release of JDBC.

Chapter 9, “XML Digital Signature API” deals with the XML Digital Signature API that defines a standard Java API for digitally signing XML documents. Two major tasks you perform using this API are signing documents and validating XML signatures. This chapter provides two examples that explain how to complete these tasks.

Chapter 10, “Streaming API for XML” discusses the new API for accessing and manipulating XML. Unlike SAX that employs a push technology, the Streaming API for XML (StAX) is based on a pull technology. Therefore, the client of the parser controls the parsing process. There are four main interfaces in this API, all of which are members of the **javax.xml.stream** package: **XMLStreamReader**, **XMLEventReader**, **XMLStreamWriter**, and **XMLEventWriter**. The first two interfaces are used for reading XML documents and the last two for creating or writing ones. This chapter presents examples that use these interfaces.

Chapter 11, “Java Architecture for XML Binding” covers JAXB 2.0, which, unlike JAXB 1.0, supports Java-to-schema binding. Like other chapters, this one begins with the necessary background and guides you through the more important classes and interfaces in the API.

Chapter 12, “Web Services” introduces two web services related technologies, Java API for XML-based Web Services (JAX-WS) 2.0 and Web Services Metadata for the Java Platform. These two APIs make developing Java web services much easier than using JAX RPC 1.0.

The JavaBeans Activation Framework (JAF) has been part of many Java programmers’ lives for years, without often given much attention to. A

required API in many Java mail applications, JAF is now officially a member of a Java SE. Chapter 13, “JavaBeans Activation Framework” reintroduces this technology and explains what it really is and how it can help you with its services.

Chapter 14, “User-Defined MXBeans” discusses the API for Java Management Extensions (JMX) 1.4. New to this version of JMX are user-defined MXBeans, even though standard MXBeans have actually been part of the Java SE since the Tiger era. This chapter teaches you how to write user-defined MXBeans as well as explains the annotation types **MXBean** and **DescriptorKey**. An introduction to JMX and standard MBeans, sisters of MXBeans, are also given at the beginning of this chapter for those new to the technology.

Chapter 15, “Concurrency Updates” talks about new additions to the Concurrency Utilities. Specifically, it explains the **BlockingDeque** interface and provides an example that shows how the Concurrency Utilities spares you from having to deal with low-level concurrency primitives such as the **wait**, **notify**, and **notifyAll** methods in **java.lang.Object** as well as the **synchronized** and **volatile** keywords. In addition, this chapter covers the new methods in the **TimeUnit** enum and the new constructor in **ConcurrentHashMap**.

Appendix A, “Enums” explains the new type introduced in Java 5. You need to be familiar with this data type as many new enums are introduced in Java 6.

Appendix B, “Generics” presents the most important feature in Java 5 that provides stricter type checking at compile time. This is a must-read if you’re new to Java 5 as Java 6 contains types that have been parameterized. This appendix teaches you to use and write generic types.

Appendix C, “Annotations” discusses annotations, a new feature in Java 5. Annotations are used to instruct the Java compiler to do something to the annotated program element. Any program element can be annotated, including Java packages, classes, constructors, fields, methods, parameters, and local variables. This Appendix explains the standard annotation types and taught how to create custom annotation types.