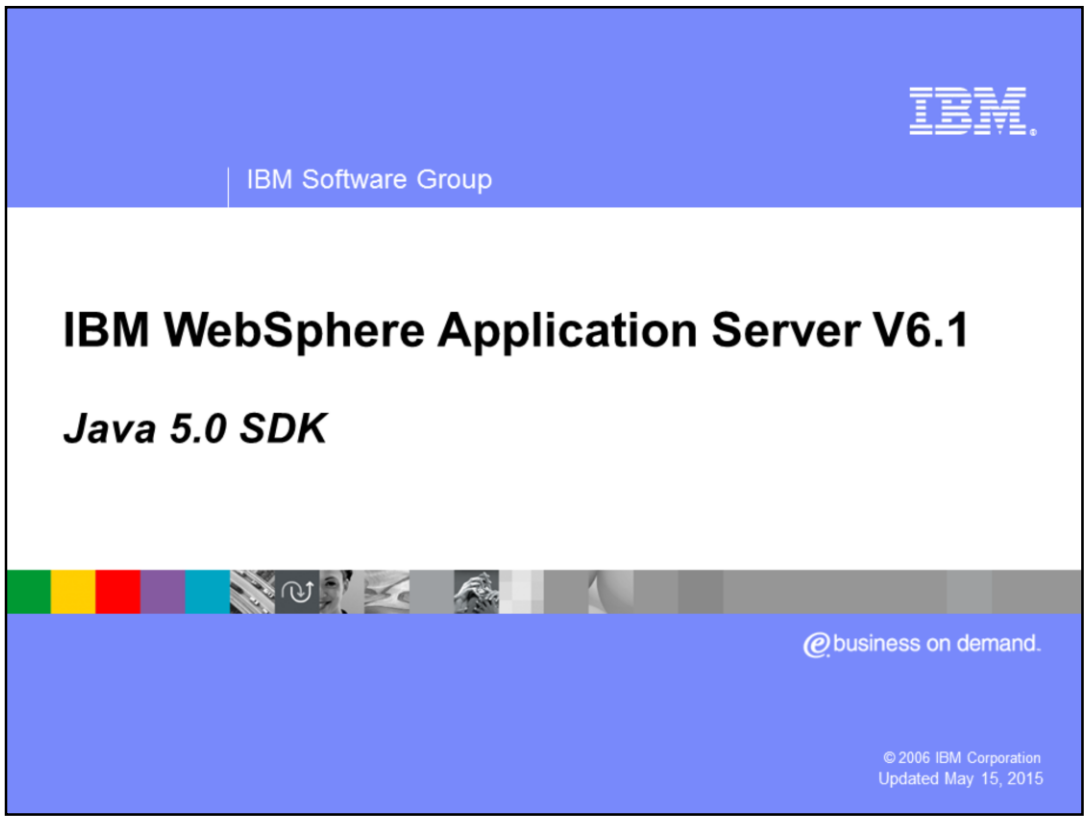
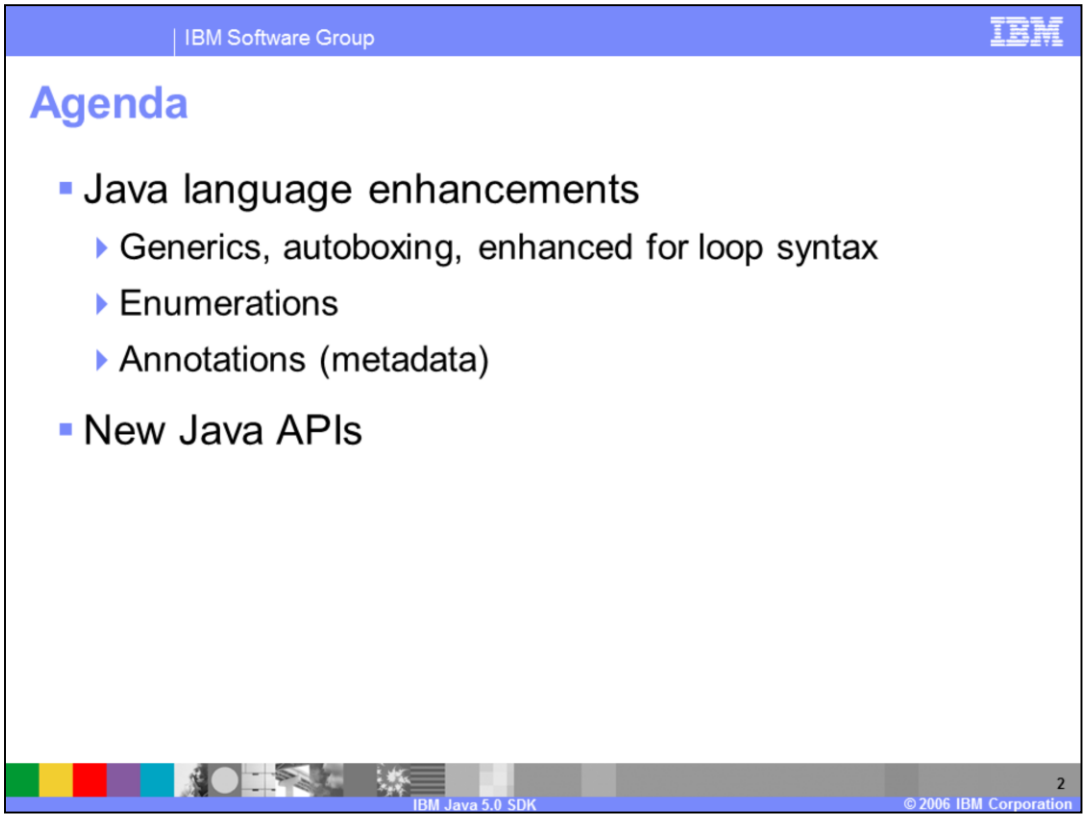
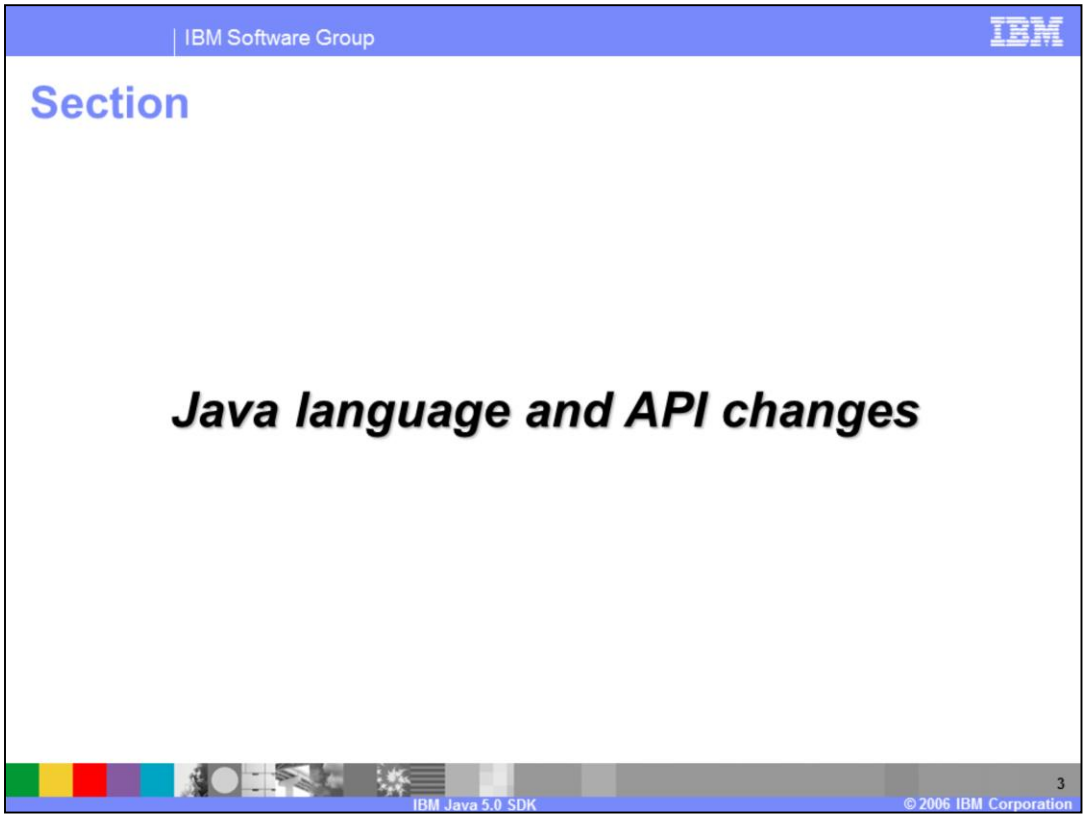


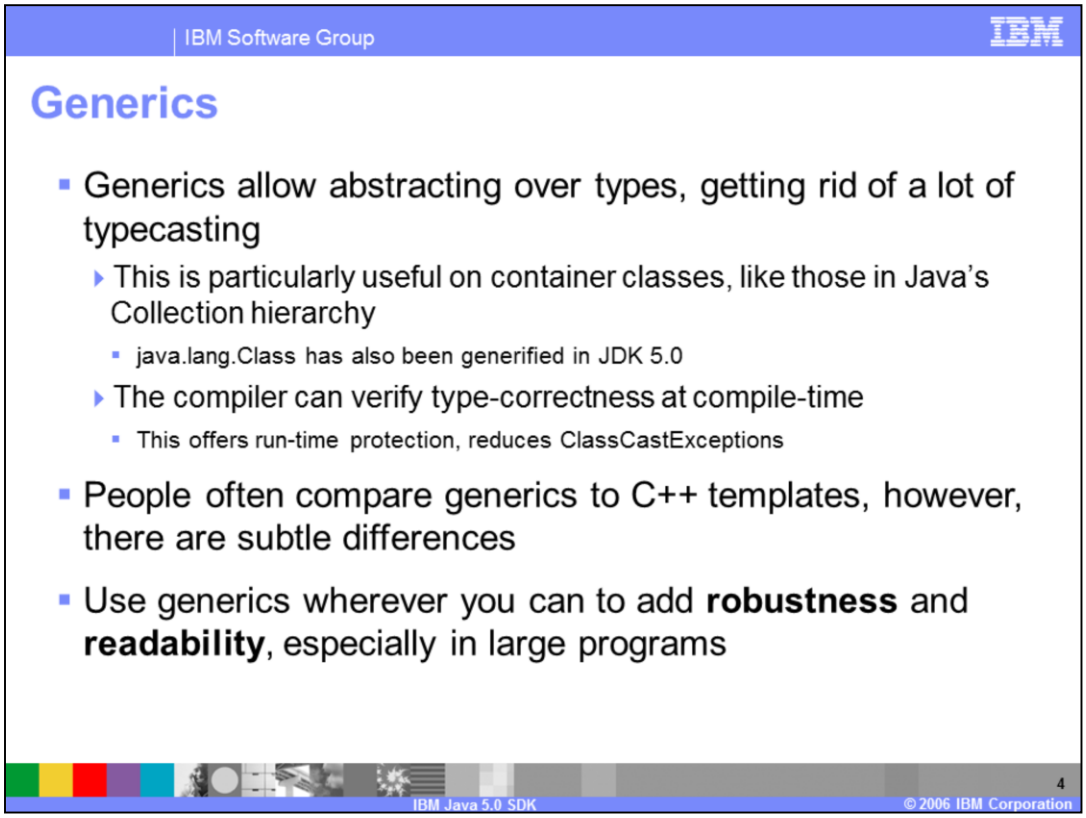
This presentation focuses on the new features in the Java™ 5.0 SDK.



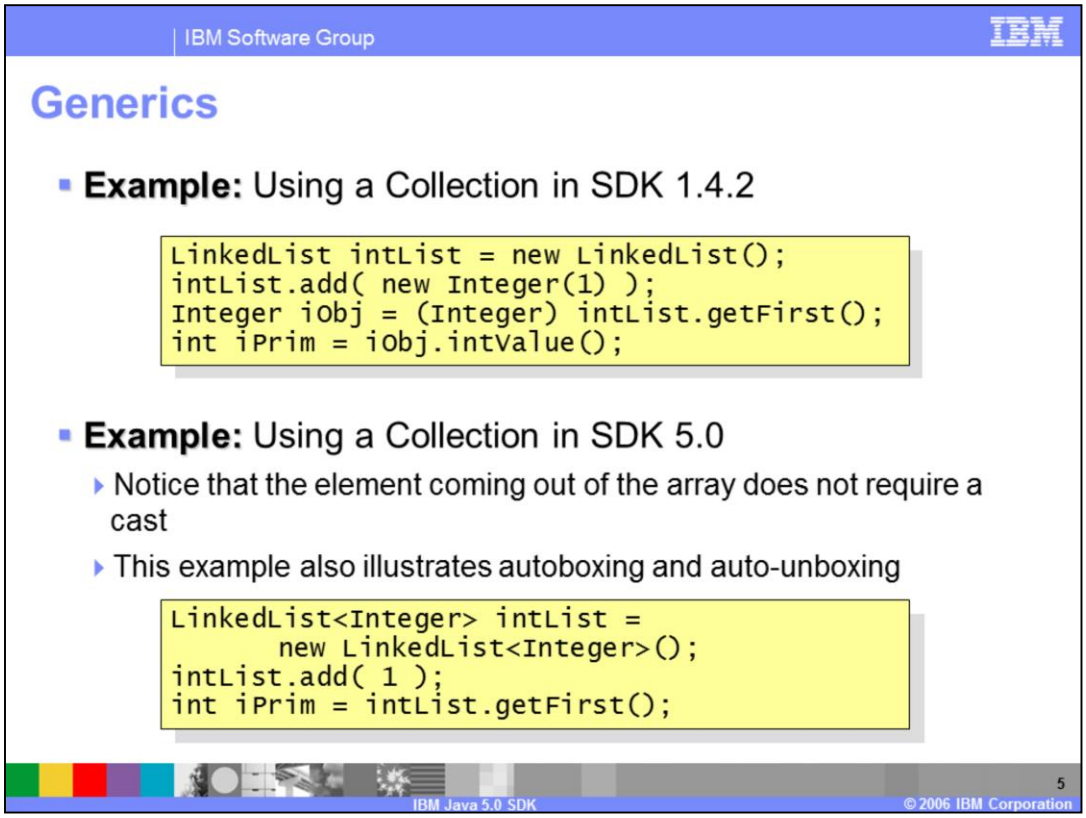
WebSphere® Application Server Version 6.1 is the first release to support Java 5.0, which is a major Java release from Sun. It incorporates many new features and APIs, including generics, autoboxing of primitives, and the enhanced for loop syntax, which you can use together to write clean, concise code. Also new in 5.0, the Java language includes built-in support for enumerations and a new metadata facility called annotations. This release also includes several new Java APIs.



This section focuses on the changes to the Java language and APIs for J2SE 5.0.

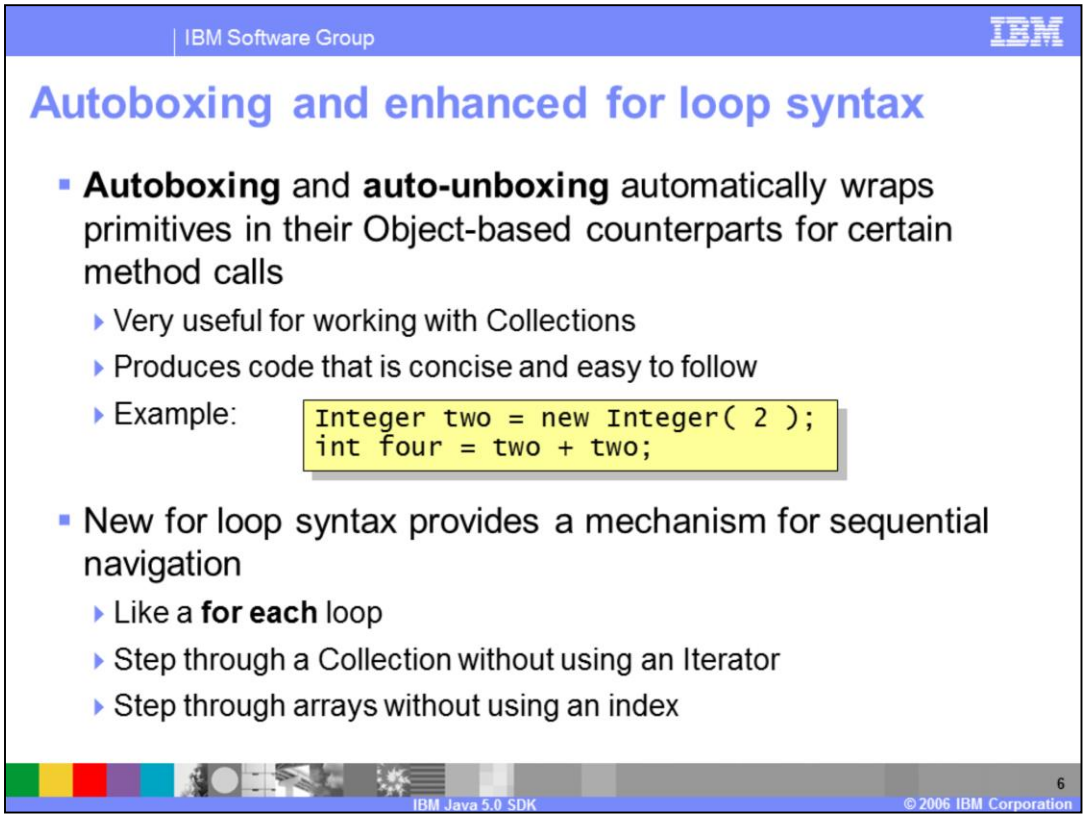


Generics allow abstracting over types. Generics are particularly useful when you are working with container classes, like those in the Java Collection hierarchy, because they offer type safety and so you no longer need to typecast your Objects to specific types. For example, when you are pulling an item out of an ArrayList of Strings, you will no longer need to cast it from a Java Object to a Java String. One of the major benefits from using generics is that the compiler will have the capability to verify type-correctness at compile-time, and this, in turn, offers you additional run-time protection and will reduce your number of ClassCastExceptions. By using generics, you will be able to improve the overall robustness and readability of your code, especially in large programs.



In the SDK 5.0 example on the bottom of the slide, you can see the syntax for creating a LinkedList of Integers. The type of object that you will be storing in the container is enclosed in brackets, and you include it both in the variable declaration and in the constructor call. Notice that, in the last line of the example, when you are pulling a value out of the LinkedList, you do not have to cast it to an Integer type.

If you ever tried to add an Object to the LinkedList, you would get a warning at compile time since this might cause type safety problems. On the other hand, if you tried to add a String to the LinkedList, your source would not compile – you would get a compile-time error.



Two additional language features in Java 5.0 that work well in conjunction with generics are autoboxing and the new enhanced for loop syntax.

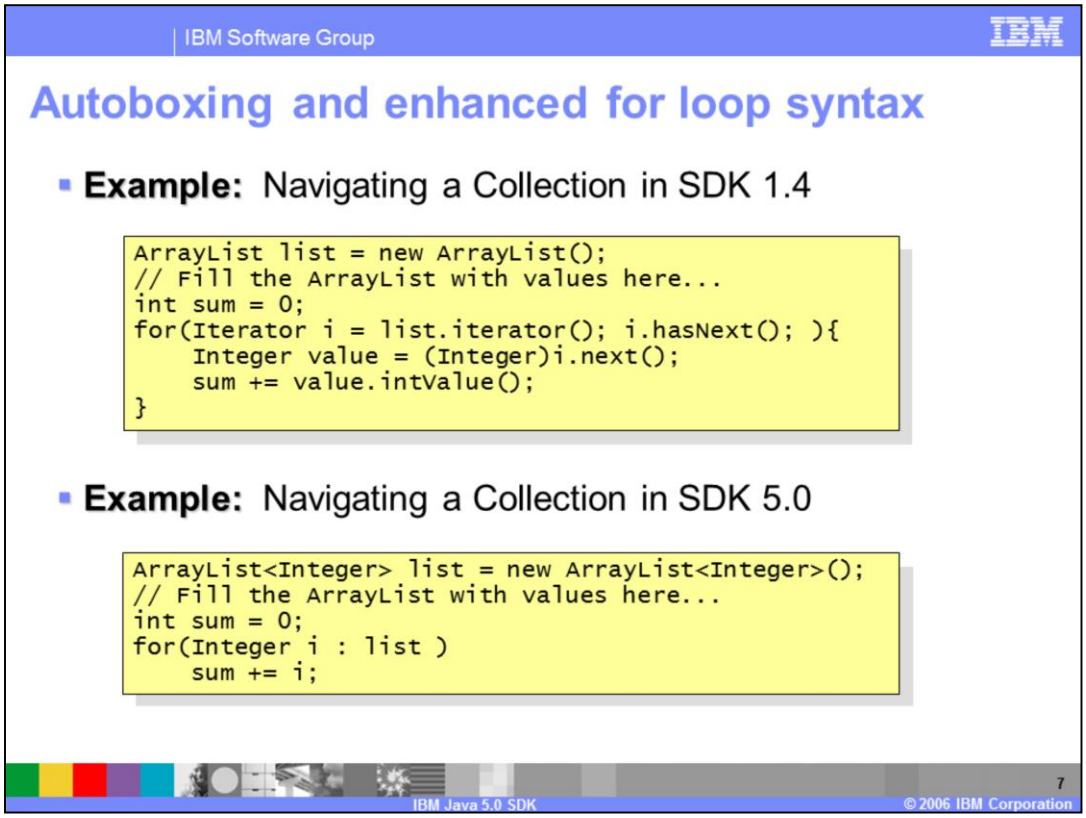
Autoboxing automatically wraps a primitive type in its Object-based counterpart for certain method calls. So, for example, when you want to add an element to Collection of Integers, you no longer have to create an intermediate Integer object with the appropriate value – you can just add the primitive int to the Collection directly. The necessary wrapping is all done for you. The same thing will happen when you are removing an Integer from a collection; you can assign the value directly to a primitive int type. This feature allows you to get rid of several extra steps and write code that is clean and concise and easy to follow.

While autoboxing is very useful in conjunction with Collections, it can also be used in other circumstances, such as:

Integer two = new Integer(2);

int four = two + two;

The enhanced for loop syntax provides a mechanism for sequential navigation, like a for each loop. You can use this new syntax to walk through a Collection without using an Iterator and also to walk through an array without using an index.

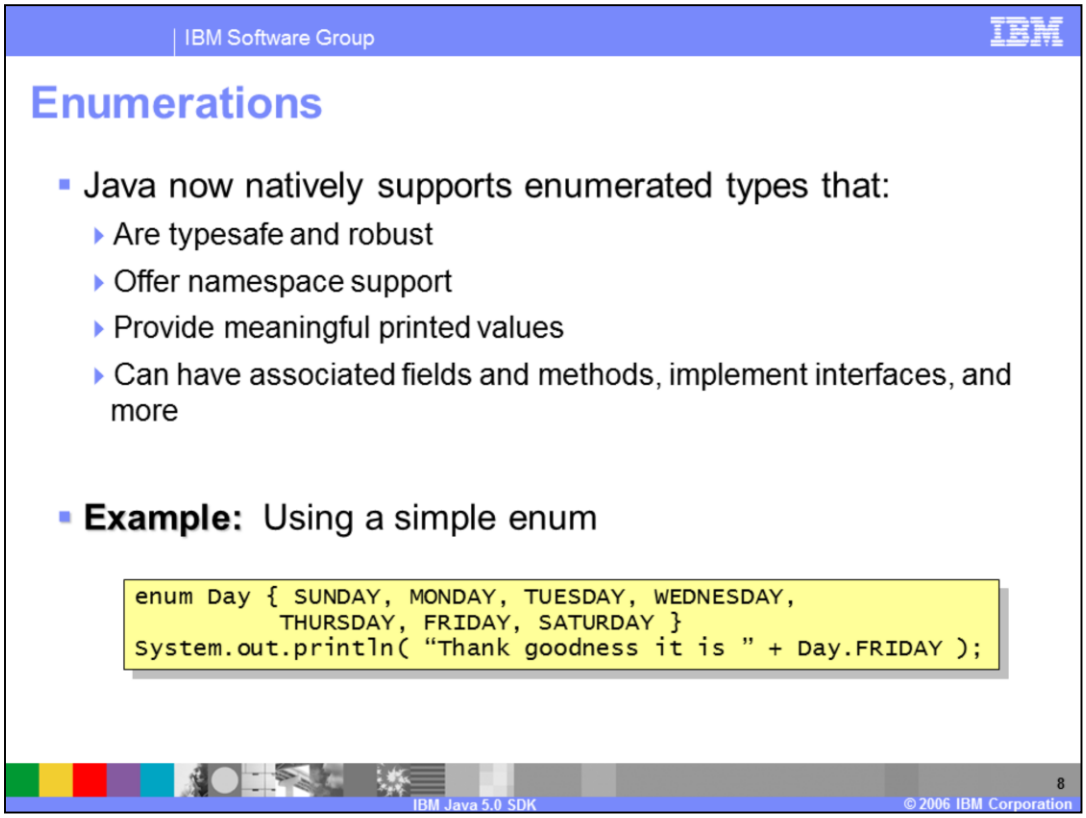


The examples on the top and the bottom of this slide are doing exactly the same thing: stepping through an ArrayList and providing a sum of all of the integers that it contains.

Note that, in that second example, the ArrayList has been created using generics, so it is an ArrayList of Integers. By creating the ArrayList in this way, you guarantee that everything going into and coming out of it is going to be of type Integer. In the first example, look at the first line inside the for loop, where the Integer is being grabbed out of the ArrayList. It is entirely possible that, at that point, someone might have thrown a String or a HashMap or something else into the ArrayList. There is no way to guarantee that the cast to type Integer will actually be successful, and you might end up with a run-time ClassCastException. In the second example, using Generics, it is not possible to have that error.

Also in the second example, you can see the enhanced for loop syntax. It uses a colon, and you only provide two arguments: an index variable to hold the values that you will be sequencing through and the list that you will be accessing.

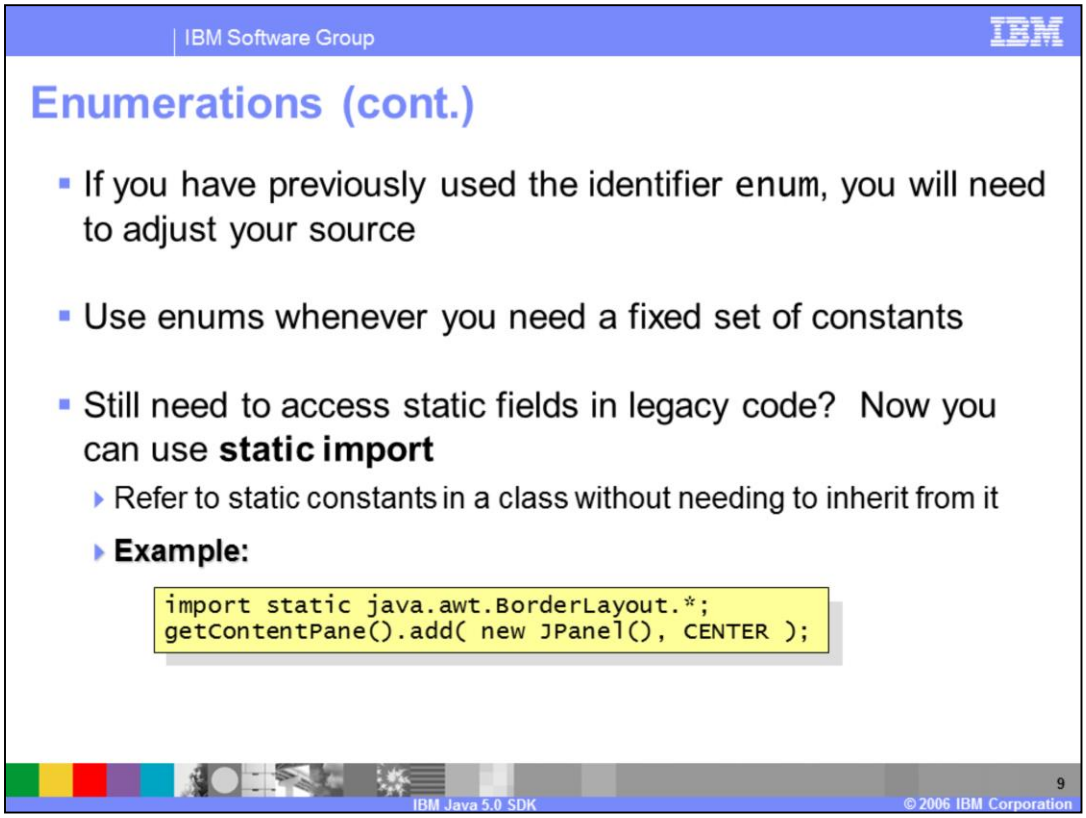
When combined into a single example, the power of these new language features becomes more apparent. Using generics, in conjunction with autoboxing and the enhanced for loop syntax, greatly simplifies the code and makes it clean and straightforward.



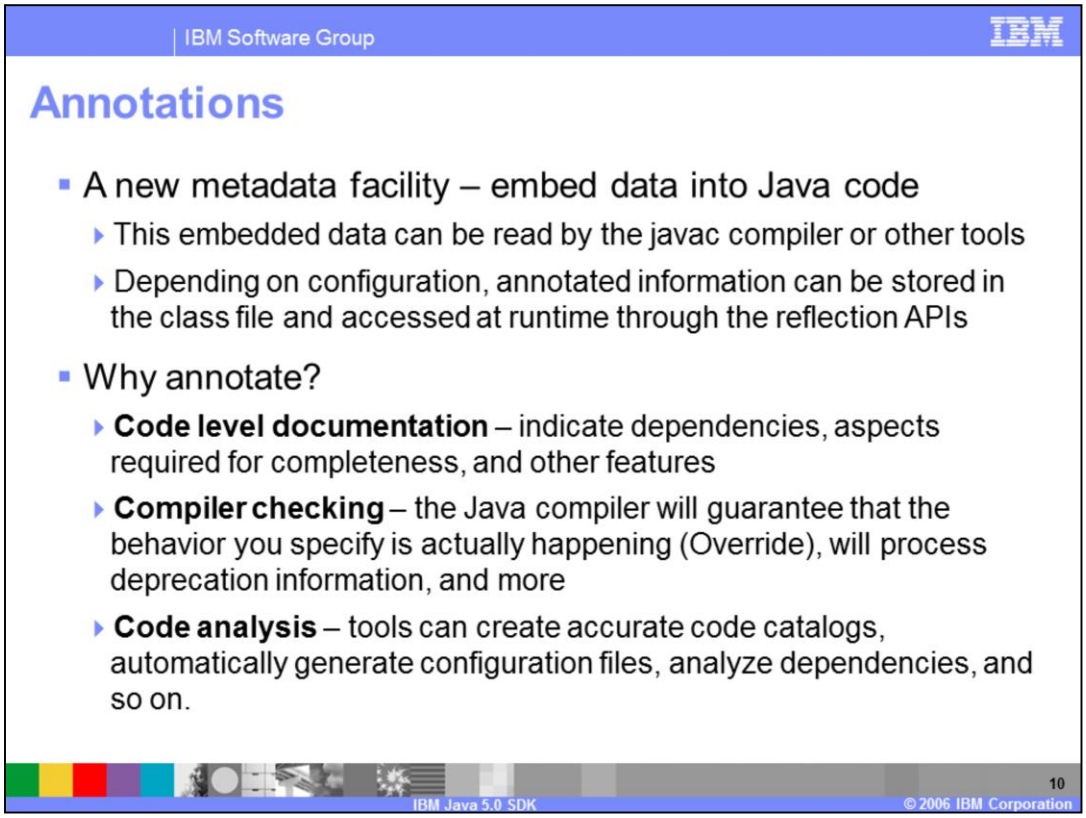
Java 5.0 is the first Java release to have native support for enumerated types. Enumerated types in Java are richer and more powerful than in other languages such as C++; they are typesafe, robust, and offer namespace support.

Enumerated types in Java also provide meaningful printed values. If, for example, you have a set of static integer constants in your code, you can give them long, descriptive names in the actual source, but if you print them out, you are only going to see a number. Java enumerated types address that issue by providing meaningful printed values.

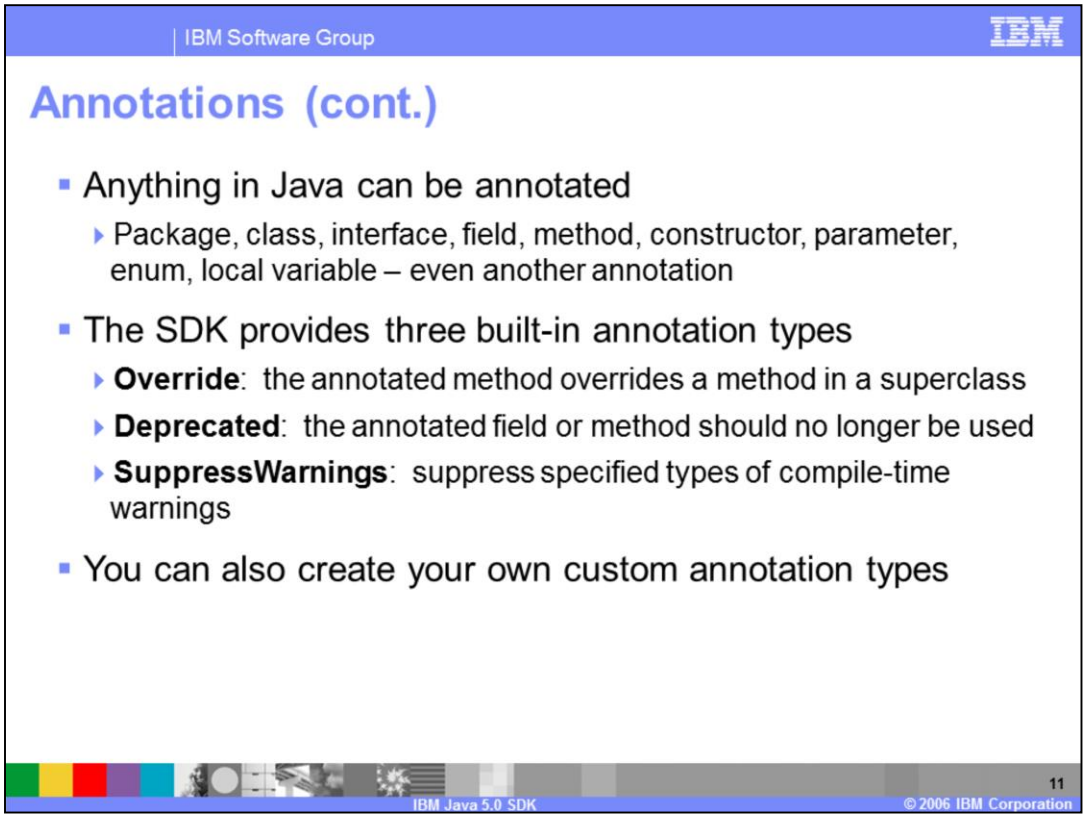
The example at the bottom of the slide shows how to create a simple enumerated type to represent the days of the week. You use the keyword enum, followed by the name that you would like your enumerated type to have, and then the content is enclosed in braces. Notice that there is no semi-colon after the closing brace for the enum. You reference values from the enum using standard dot syntax, and when you print the example sentence out, it is going to read “Thank goodness it is Friday” and not “Thank goodness it is 5.”



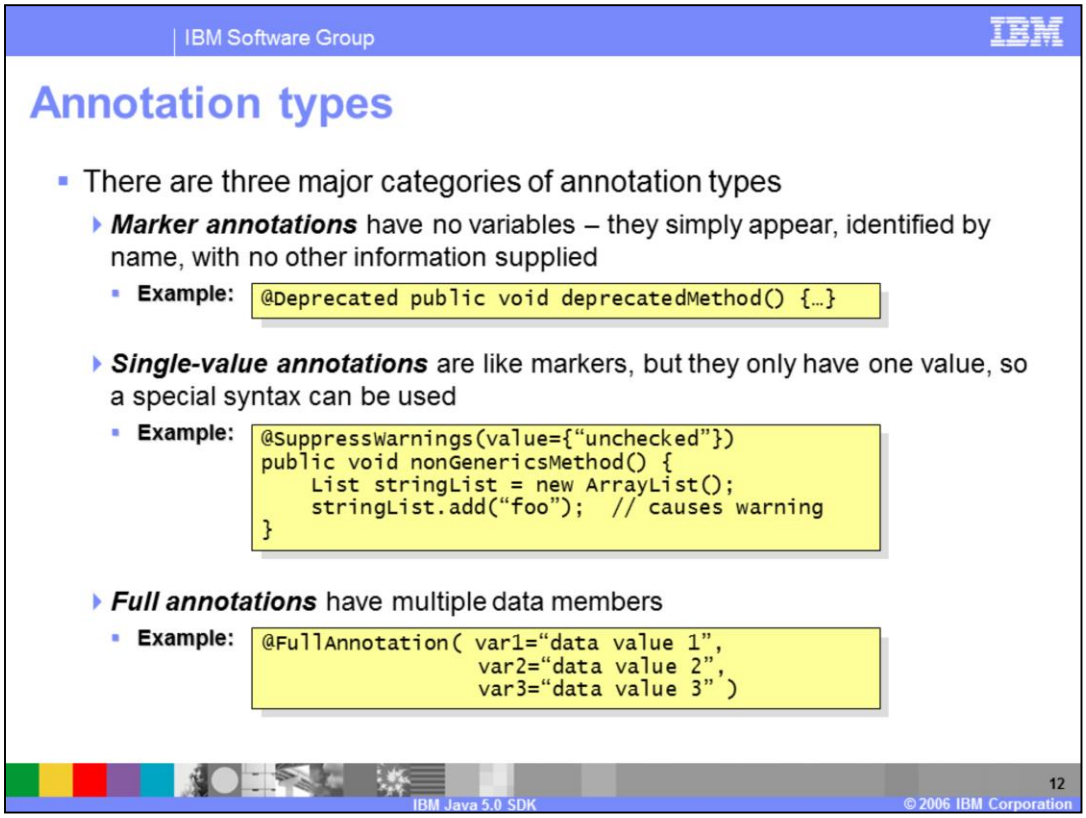
As of Java 5.0, enum is now a registered keyword, so if you have previously used it in your code, you are going to have to adjust your source. This is probably the most common change required when migrating to 5.0 from previous Java versions. Enums should be used whenever you are working with a set of static constants. The new static import functionality makes it easier to access static class fields in code. By importing a class as static, you do not have to inherit from it to use its fields. For instance, in the example, you can directly reference the field CENTER without having to use BorderLayout.CENTER.



Java 5.0 provides a new metadata facility, called annotations, that allow you to embed data directly into your Java code. This data can then be read by the compiler or other tools so that they can perform some specific tasks or generate artifacts for you automatically. Some annotated information is also available at runtime using the Java reflection APIs. There are several good reasons to use annotations. They allow you to perform code level documentation in addition to comment level documentation (annotations are a part of your Java code and not your comments). The Java compiler will look at your annotations and can guarantee that the behavior that you specify is actually happening. Code analysis tools can also use annotations to automate things for you, from code generation to code analysis.

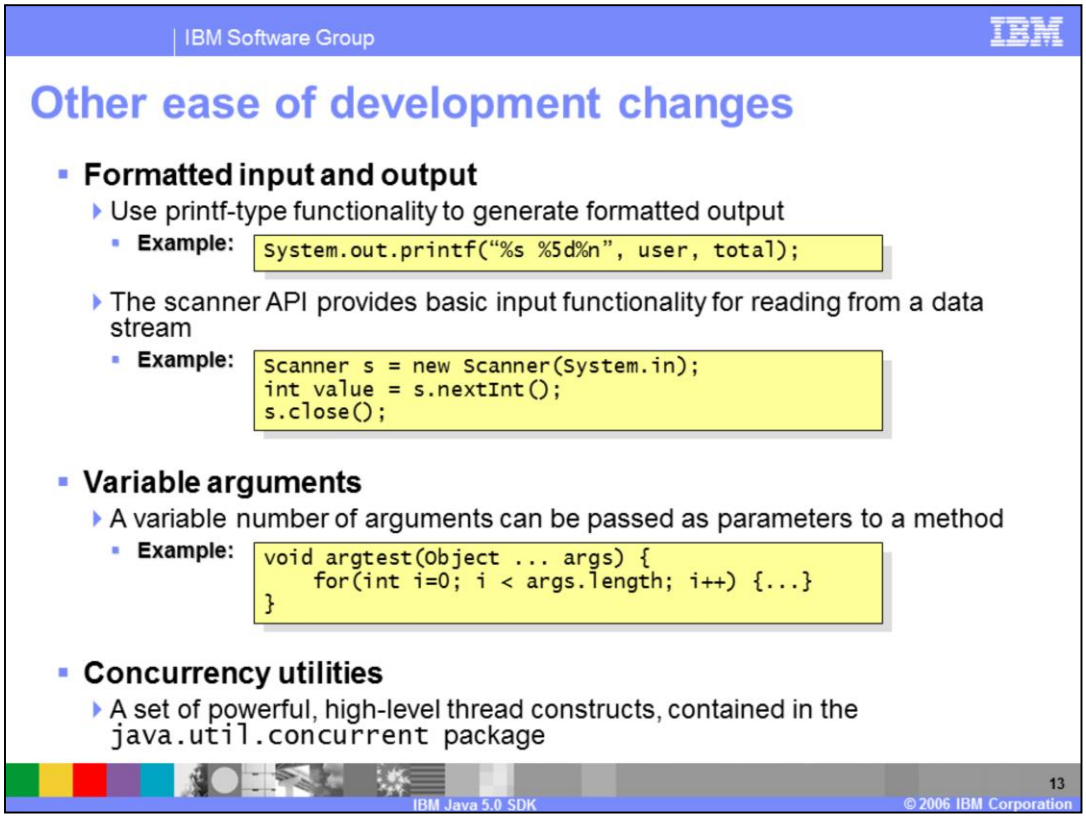


Anything in Java can be annotated, including annotations themselves, and the language provides three built-in annotation types. The Override annotation indicates that the annotated method is overriding a method in a superclass. The Deprecated annotation indicates that a field or method should not be used anymore. This is different than the previous method of indicating deprecation using javadoc comments. You can also use the SuppressWarnings annotation to suppress compile-time warnings. This can be useful if you are compiling 1.4 code with a 5.0 compiler. Since generics were not supported in 1.4, you will probably get lots of compiler warnings about performing non-typesafe operations. You can use the SuppressWarnings annotation to turn those warnings off. Some other meta-annotation types, or annotations for annotations, are also built into the language. In addition to these, you can create your own custom annotation types.

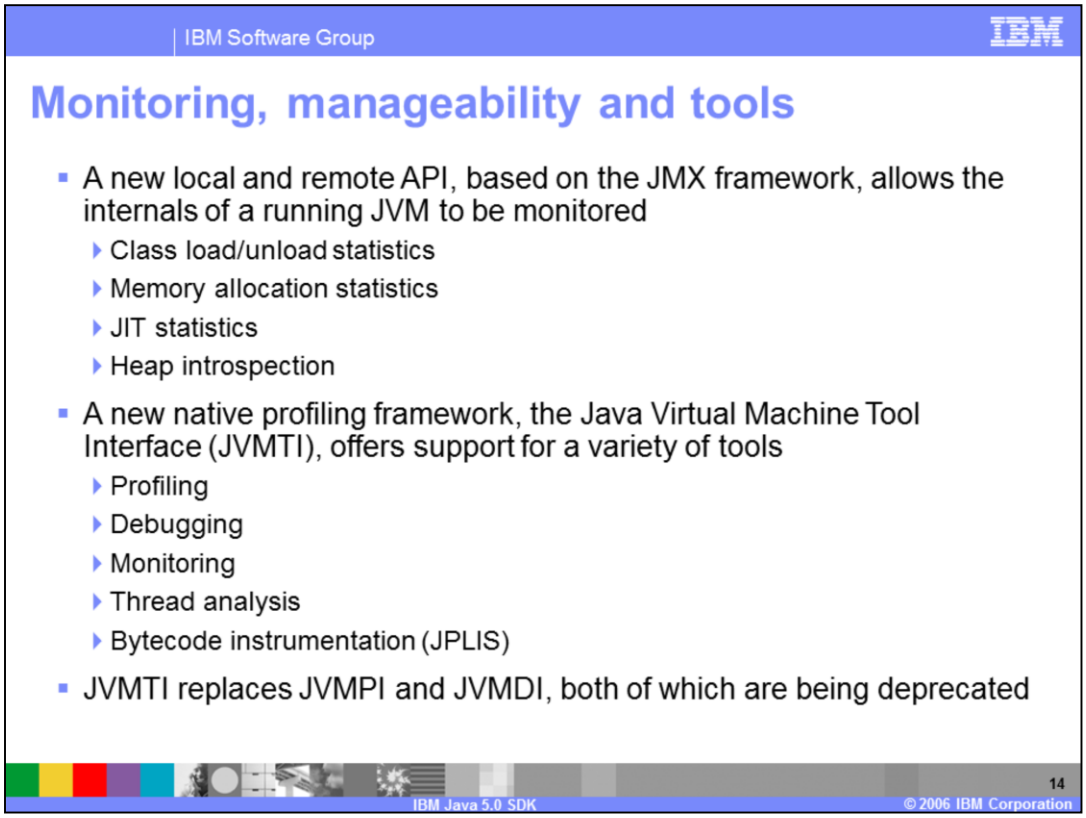


This slide has a few examples of what annotations look like. Notice that annotations are introduced by the @ symbol and can have associated values.

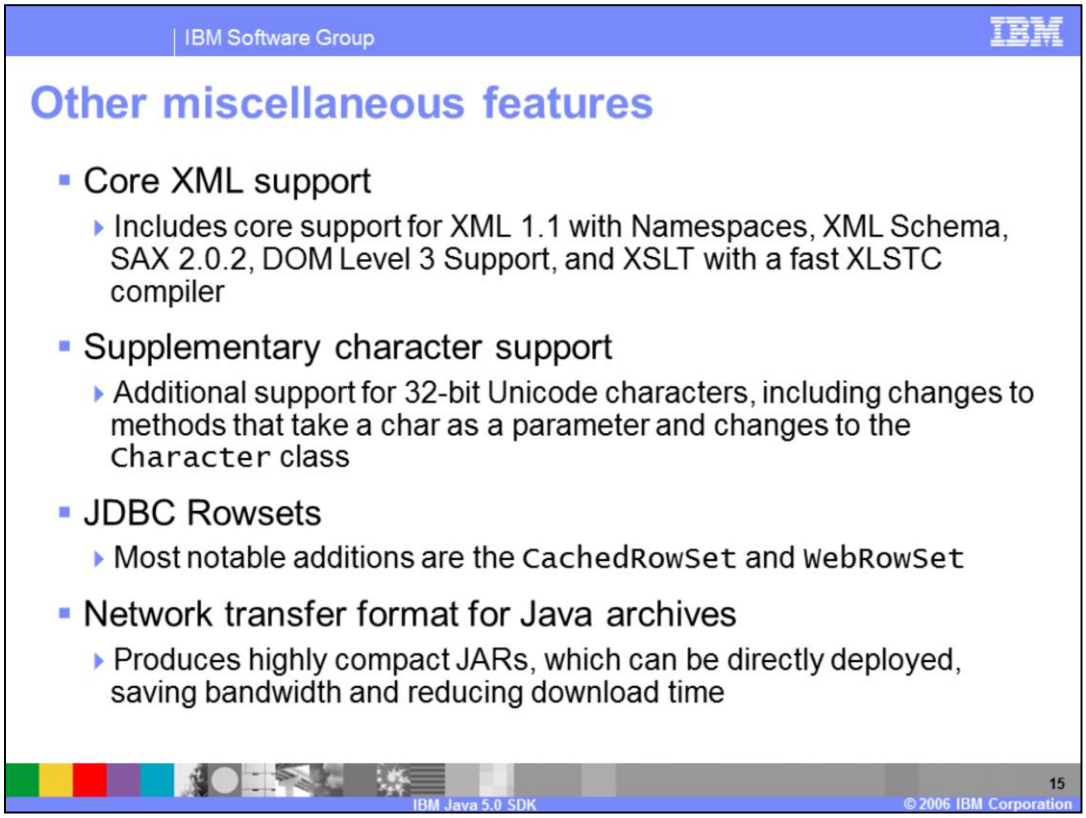
There are three main categories of annotations, which are annotations with no values, annotations with a single value, and annotations with more than one value. The categories are straightforward, but you may see annotations referred to using these terms in documentation and other reference materials.



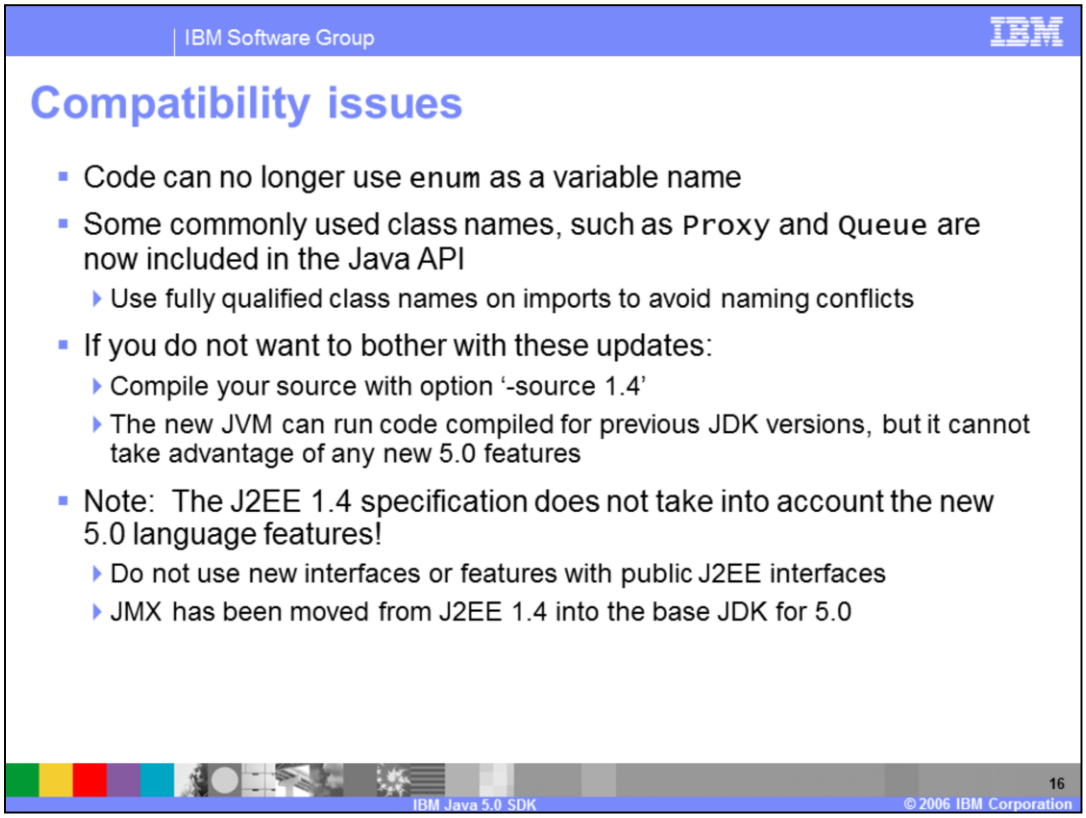
JDK 5.0 incorporates some additional ease of development changes. There are some I/O enhancements which support printf formatted output, and new Scanner APIs that simplify the process of reading data in from a stream. In order to facilitate the printf functionality, you can now also create functions with a variable number of arguments, and you can see an example of that syntax on the slide. This release also features some new concurrency utilities, including thread safe queues, timers, locks, and other synchronization primitives. These were previously available from a third-party source, so you may already be familiar with them or have used them. They have been incorporated into the base JDK in this release.



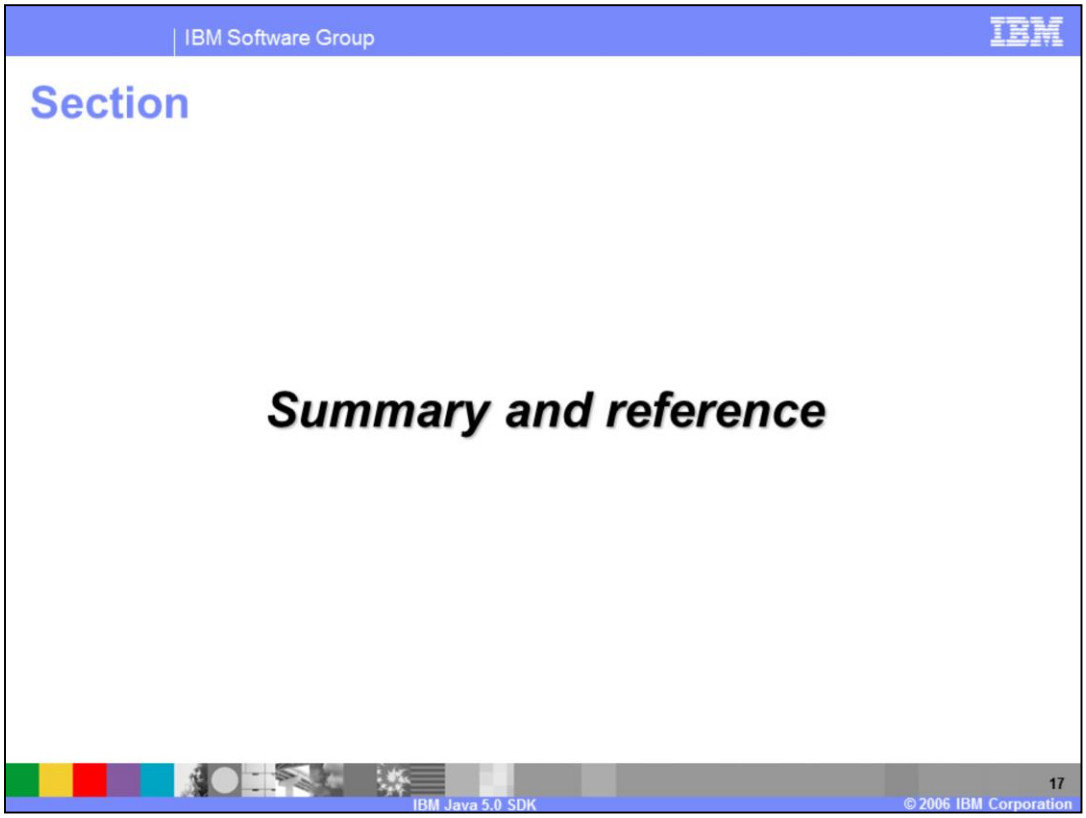
JDK 5.0 includes some manageability and tool enhancements. The JMX specification from J2EE 1.4 is now included in the standard J2SE specification for 5.0. These statistics gathering and investigative tools are now a part of the base JDK. The other major change is the deprecation of JVMPI and JVMDI which are being replaced with JVMTI, the Java Virtual Machine Tool Interface. This is a new profiling framework that supports a variety of debugging, monitoring, and code analysis tasks.



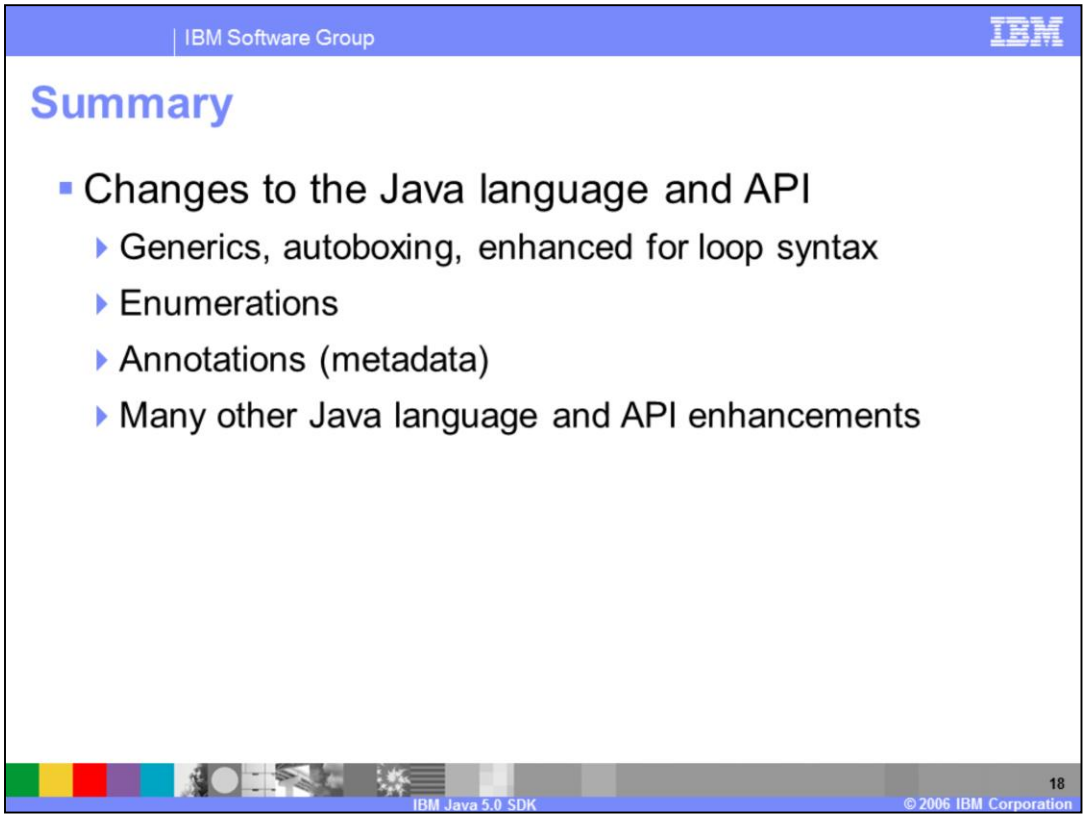
Some other API changes have been introduced in this release. Core Java XML support has been changed and updated to XML version 1.1, including namespace and schema support. Support has been added for some supplementary 32-bit Unicode characters, and the JDBC RowSet classes have been enhanced to have better support for working with data without maintaining a live connection to the data source. This release also features a new, more compact network transfer format for JAR files, which can significantly reduce the amount of bandwidth consumed and download time.



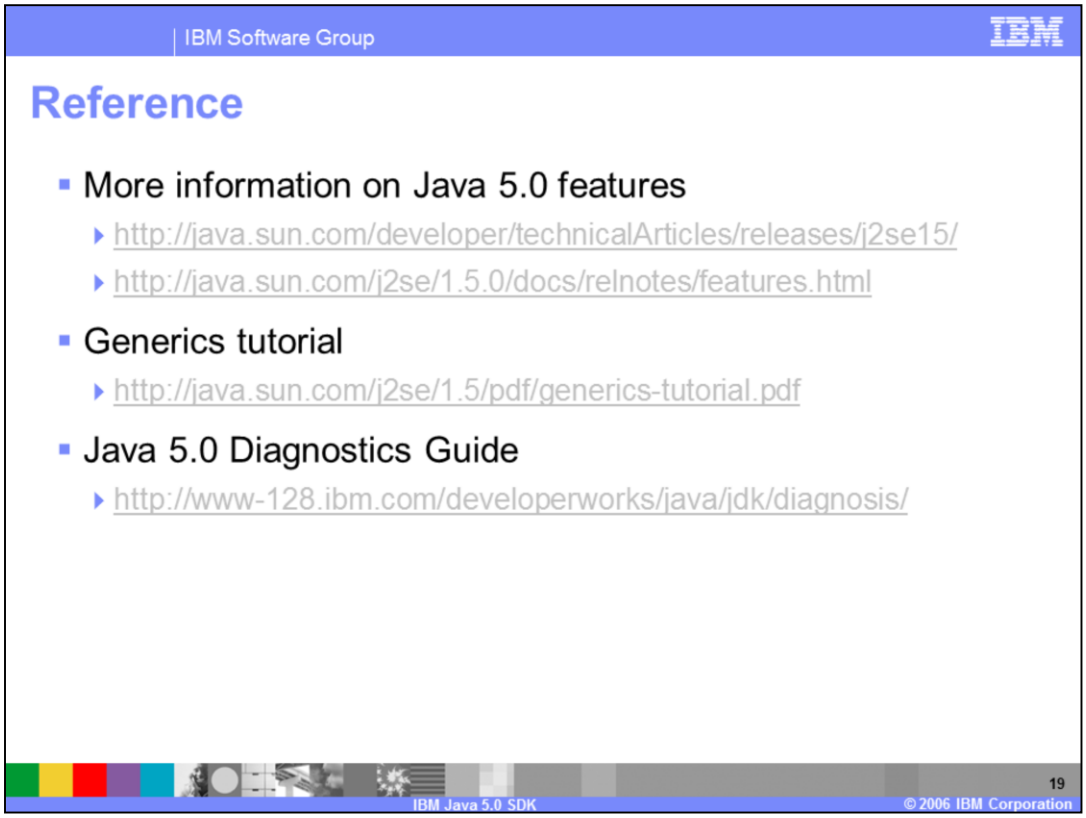
The most common compatibility issue when migrating to Java 5.0 is that your code can no longer use enum as a variable name since it has been added as a registered Java keyword. Some other commonly used names – Proxy and Queue – are now included in the Java API, and you should use fully qualified class names on imports to avoid any naming conflicts. If you do not want to take advantage of any of the new Java 5.0 features, you can use compiler options to compile your source targeted at JDK 1.4. The new 5.0 JVM is compatible with previous Java versions, so, for example, Java code compiled SDK 1.4.2 should run unchanged. That code will not, of course, be able to take advantage of any of the new 5.0 features. The J2EE 1.4 specification has no knowledge of the new Java 5.0 features. So, to avoid issues there, do not use any of the new interfaces or features with any of your public J2EE interfaces. Also, JMX has been consumed into the base JDK effective this release.



The last portion of the presentation contains a summary and references.



WebSphere Application Server Version 6.1 is the first release that supports Java 5.0. The Java 5.0 SDK includes many new language features. Generics allow abstracting over types, and autoboxing allows you to use primitive types without wrapping them in Java Objects. Use these features in conjunction with the enhanced for loop syntax to write concise, uncluttered Java code. J2SE 5.0 also has built-in support for enumerated types and the new annotations metadata facility. Several new APIs have been introduced, including many ease of development changes and the new Java Virtual Machine Tool Interface.



This slide contains references for more information on the new features in Java 5.0. Sun's Java website contains helpful articles on J2SE 5.0 features, including a nice generics tutorial that helps explain more of the details of how to use generics in your code. The Java 5.0 Diagnostics Guide provides information on IBM's Virtual Machine for Java Platforms and how to troubleshoot Java problems.

Section

Appendix

Generics: interacting with legacy code

- In legacy code, a parameterized type (like `Collection<String>`) can be assigned to a raw type (such as a Collection with no parameter type)
  - This results in an unchecked warning at compile time and could result in a run-time error
- Calling legacy code from code that supports generics is also possible, but dangerous
  - You lose all of the safety guarantees of the generic type system

Generics: wildcards

- Use wildcards to specify hierarchical relationships between generics
  - Note: A `Collection<Object>` is NOT a supertype of all kinds of collections
- Example
  - Write a method that can be used on any kind of collection

```
void printCollection(Collection<?> c) {
    for (Object e : c) {
        System.out.println(e);
    }
}
```

Generics: wildcards (cont.)

- Examples
  - Restrict allowable objects using an upper bound

```
public void printHosts
  (List<? extends InetAddress> addresses) {...}
```

  - Restrict allowable objects using a lower bound

```
public static <T extends Comparable<? super T>>
  T max(Collection<T> coll) {...}
```

  - Give a type parameter multiple bounds

```
public static <T extends Object &
              Comparable<? super T>>
              T max(Collection<T> coll) {...}
```

Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.