Java 6 and Java 5 New Features

Presented to

RMOUG
ROCKY MOUNTAIN ORACLE USERS GROUP

Presented by: John Jay King
King Training Resources - john@kingtraining.com

Download this paper from: http://www.kingtraining.com

Session Objectives

- Know how Java 6 and Java 5 differ from previous releases
- Understand how to use improved syntax features
- Be able to use new features to simplify existing programs

Who Am I?

- John King – Partner, King Training Resources
- Providing training to Oracle and IT community for over 20 years
  - Databases: Oracle, DB2, SQL Server, more…
  - Languages: PL/SQL, Java, C#, COBOL, PL/I, more…
  - Operating Systems: Linux, Unix, Windows, z/OS
  - Tools: ADF, XML, HTML, JavaScript, more…
- Leader in Service Oriented Architecture (SOA) design and implementation
- Home is Centennial, Colorado – I like to hike and drive in the mountains

Introduction to Java 5 & 6

- Java has become the dominant development language for web-based systems:
  - Since its release to the public in 1995 (Java 1.0) Java has matured improving functionality and performance with each new release
  - In September 2004 Sun released Java 5.0
  - In December 2006 Sun released Java 6.0

Java 5 & 6 “In a nutshell”

- Java SE 5 contained several major updates to the Java programming language including:
  - Annotations
  - Generics
  - Autoboxing
  - Improved looping syntax
- Java SE 6 specification focused on new specifications and APIs including:
  - XML processing and Web services
  - JDBC 4.0
  - Annotation-based programming
  - Java compiler APIs
  - Application client GUI APIs

Numbering change

- The current release's actual name is Java Standard Edition 6 (Java SE 6); the previous release was Java Standard Edition 5.0, (J2SE 5)
- Sun changed version numbering with Java 5
  - J2SE 1.5 is now J2SE 5.0 (leading "1" dropped)
  - Most Sun documentation including Javadoc references 1.5 rather than 5.0 when specifying the new version
  - Java 2 Version 5.0 (1.5) still uses "Java 2" to denote the second generation of Java and to illustrate the family nature of J2SE, J2EE, and J2ME; Java 6 does not
  - Most Sun sources use Java SE 6 for Java 6

JDK/SDK nomenclature

- Sun has resurrected the name "JDK" as in "Java SE JDK" rather than using the "SDK" moniker adopted by Java 1.2, 1.3, and 1.4
- Sun has also returned to calling the runtime environment "JRE" rather than "J2RE"
- The "official" names from Sun:

– Java™ Standard Edition 6.0	Java SE 6.0
– Java™ 2 Platform Standard Edition 5.0	J2SE™ 5.0
– Java™ SE Development Kit 6.0	JDK™ 6.0
– J2SE™ Development Kit 5.0	JDK™ 5.0
– Java SE™ Runtime Environment 6.0	JRE 6.0
– J2SE™ Runtime Environment 5.0	JRE 5.0

Java Grows Over Time

- Here is a comparison of Java library size from version to version:

– Java 1.2	1500+ Classes and Interfaces
– Java 1.3.1	1800+ Classes and Interfaces
– Java 1.4	2700+ Classes and Interfaces
– Java 5 (Java 1.5)	3200+ Classes and Interfaces
– Java 6 (Java 1.6)	3700+ Classes and Interfaces

New Java Syntax

- Java's syntax has had several improvements
- Topics covered specifically in this paper include:
  - generics
  - enhanced for loop
  - auto boxing/unboxing
  - typesafe enumerations
  - static import
  - metadata via annotations
  - formatted output
  - variable argument lists
  - simplified input processing via scanner
  - improved synchronization
  - More…

Need for Generics

- At first glance Java's Generics feels familiar to those of us who have used C++ templates, but Java Generics are so much more.
- Java provides many opportunities for manipulating objects where the actual object type is stripped and must be re-supplied using a cast when the object is used later

Need for Generics: Example

- To illustrate the need for Generics; look at this:

```
ArrayList oldStyle = new ArrayList();
oldStyle.add(new String("Hello"));
oldStyle.add(new String("there"));
oldStyle.add(new Integer(12));  // ok the old way
oldStyle.add(new String("whoops"));
// following loop raise runtime error
for (Iterator i = oldStyle.iterator();i.hasNext();)
{
    System.out.println("Entry = "
                     + (String) i.next());
}
```

  – This code generates the runtime error "ClassCastException" since the third item being retrieved from the ArrayList is not the expected data type

Generics to the Rescue!

- Using Generics allows specification of the allowable data type for a Java object so that the compiler will catch data type errors.
- Generic syntax specifies the data type inside less-than "<" and greater-than ">" symbols as follows:

```
ArrayList<String> newStyle = new ArrayList<String>();
```

Generics Example

- The example below shows the String class but the class may be any class available in your CLASSPATH
- The compiler uses the data type specified to restrict what may be placed into the object at compile time

```
ArrayList<String> newStyle = new ArrayList<String>();
newStyle.add(new String("Hello"));
newStyle.add(new String("there"));
// following line raises compile error so it is commented
//newStyle.add(new Integer(12)); // compile error
newStyle.add(new String("whoops"));
for (Iterator<String> i = newStyle.iterator();i.hasNext();)
{
   System.out.println("Entry = " + (String) i.next());
}
```

Lint is not just in your navel!

- The "lint" command has long been used in the Unix world to verify C program syntax, data type use, and portability of code
- New Java 5&6 compiler switch "-Xlint" allowing compiler to flag potential

Java Lint Syntax

```
javac UsingGenerics.java -Xlint:unchecked -deprecation
```

- Options for Xlint include:

– all	Get all lint warnings
– deprecation	Warns about deprecated API use (similar to -deprecation)
– fallthrough	Flags cases in a switch statement that "fall through" to the next case
– finally	"finally" blocks cannot complete
– path	Path directories specified do not exist
– serial	One or more Serializable classes do not have serialVersionUID defined
– unchecked	Warns of unchecked generic type use

Lint Warnings

- Here’s what Lint warnings look like:

```
C:\JavaTiger\src\samples\UsingGenerics.java:30: warning:
   [unchecked] unchecked call to add(E) as a member of the raw
   type java.util.ArrayList
       oldStyle.add(new Integer(12));
                            ^
C:\JavaTiger\src\samples\UsingGenerics.java:31: warning:
   [unchecked] unchecked call to add(E) as a member of the raw
   type java.util.ArrayList
       oldStyle.add(new String("whoops"));
```

java.util.Arrays.toString()

- java.util.Arrays class has new methods including a toString() method to print the contents of any array/collection

```
int[] anArray = { 1, 3, 5, 7, 9, 11, 13, 15, 16, 20 };
System.out.println(Arrays.toString(anArray));
```

  – Generates the following output:

```
[1, 3, 5, 7, 9, 11, 13, 15, 16, 20]
```

java.util.Arrays.deepToString()

- Arrays.deepToString() displays the contents of a multi-dimensional array:

```
int[][] apartment = new int[5][4];
```

  - The results of using Arrays.toString() and Arrays.deepToString() are illustrated below.
  - First, using Arrays.toString() the contents of the first level show as addresses (Windows PC used for example):

```
[[I@10b62c9, [I@82ba41, [I@923e30, [I@130c19b, [I@1f6a7b9]
```

  - Next, using Arrays.deepToString() the contents of the array are listed:

```
[[0, 0, 0, 0], [0, 1, 2, 3], [0, 4, 5, 6], [0, 7, 8, 9],
[0, 10, 11, 12]]
```

Other Additions to Arrays

- Arrays also added three other methods:
  - Arrays.deepEquals(array1,array1)
  - Arrays.hashCode()
  - Arrays.deepHashCode()

Enhanced “for” loop

- Java 5 added a new style of “for” loop (sometimes called "for-in" loops)

```
String[] lastName = new String[5];
lastName[0] = "Winklemann";
lastName[1] = "Ruiz";
lastName[2] = "Gandhi";
lastName[3] = "Yakazuki";
lastName[4] = "Jones";
for (String thisName : lastName) {
    System.out.println("Name is " + thisName);
}
```

  – This code loops through each entry in an array or collection object returning one value at a time for processing -- an Iterator is used without an Iterator being defined!

For-In Syntax

- The new "for" construct is equally at home with either a traditional array or an object of some collection type:

```
for (String thisName : lastName) {
    System.out.println("Name is " + thisName);
}
```

  – Data type of one item in array/collection (String above)
  – Local name used for returned item in for loop (thisName above)
  – : (colon, think of it as the word “in”)
  – Name of array or collection object that implements the new java.lang.Iterable interface (lastName above)

Object to/from Primitive Issues

- In Java 1.4 (or earlier releases), properly moving data between wrapper class objects and primitives required extra work:

```
Integer intObject = new Integer(123);
int intPrimitive = intObject.intValue();
double doublePrimitive = 123.45;
Double doubleObject = new Double(doublePrimitive);
```

Autoboxing and Unboxing

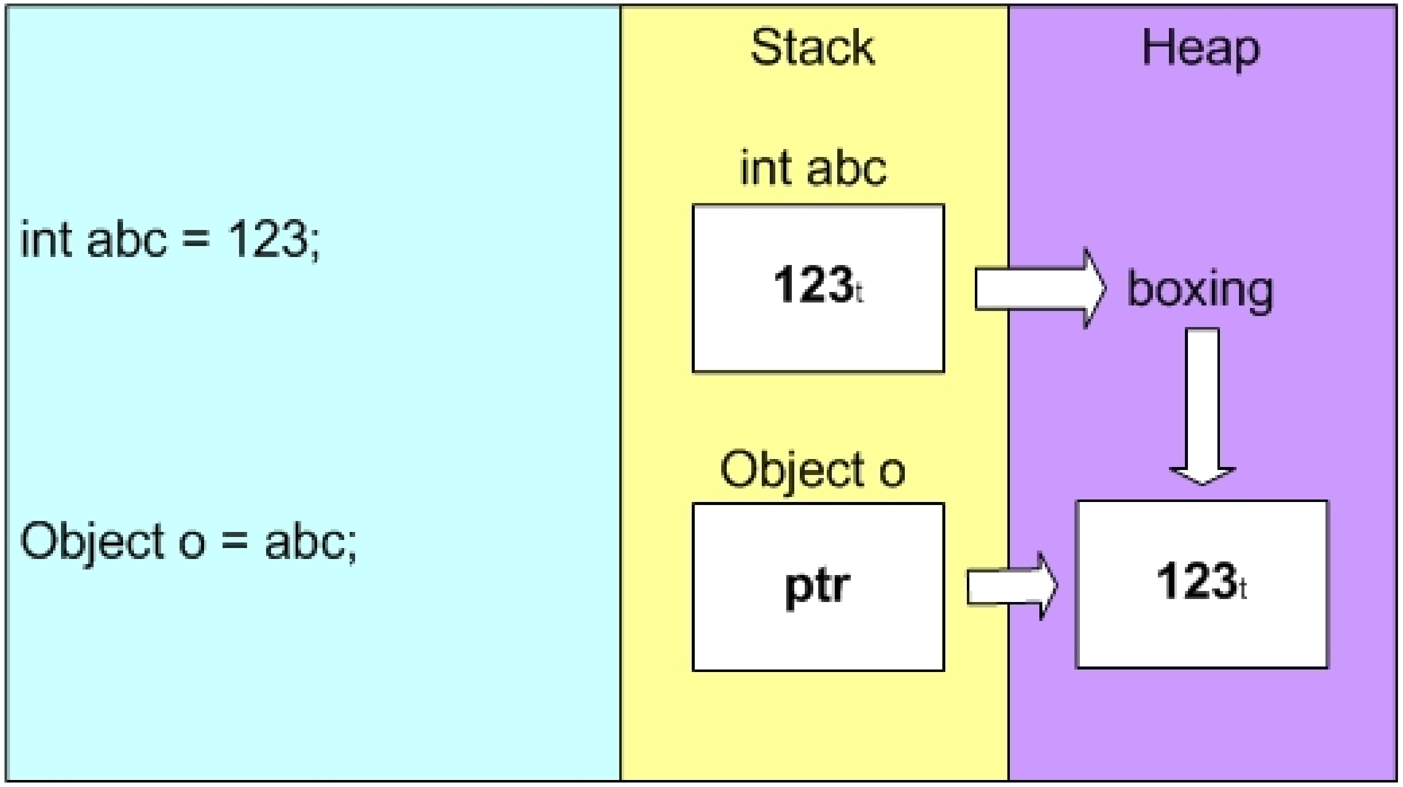
- Java 5 (Java 1.5) and later automatically “box” and ”unbox” values

```
Integer intObject = new Integer(123);
int intPrimitive = intObject;
double doublePrimitive = 123.45;
Double doubleObject  = doublePrimitive;
```

  - The advent of automatic Boxing and automatic Unboxing greatly simplifies code when using Collection and other types of objects
  - Boxing and Unboxing is also important since Primitive data and Reference Type data are stored in different places; primitives representing local variables are stored on the stack while objects are stored in heap memory.

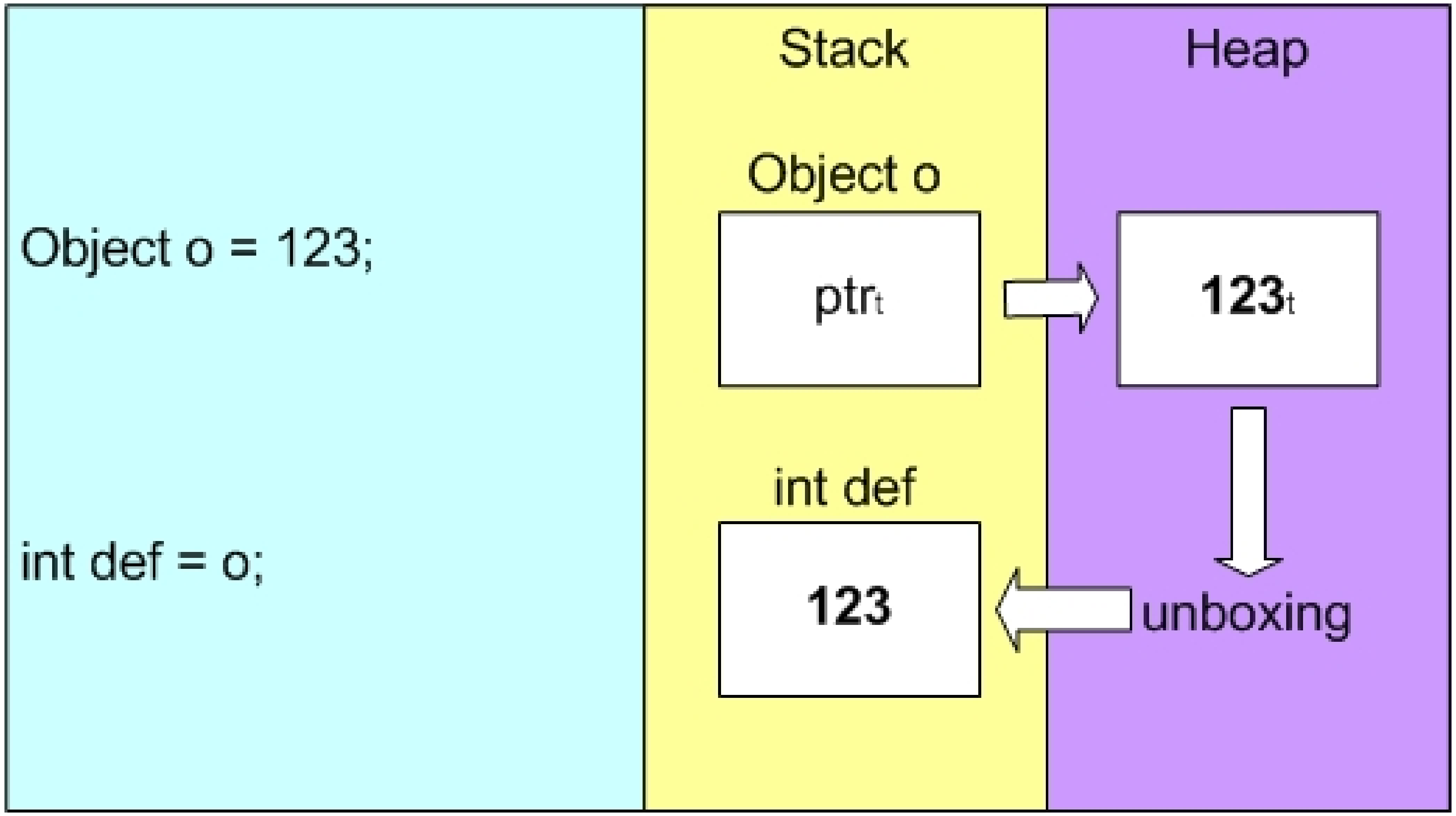
Boxing

- When an integer is assigned to an object, the system, “boxing” makes a copy of the value on the heap and points the object to the new value.



UnBoxing

- When an object assigned to an integer, “unboxing” copies the value from the heap into the variable's storage in the stack



Enum

- Another new Java 5 feature that looks familiar to C programmers is the "Enum" data type
- Enum allows assignment of a specific set of values to associated

```
public class UsingEnums {
    public enum Weekdays {
        Monday, Tuesday, Wednesday, Thursday, Friday,
        Saturday, Sunday
    };
    public UsingEnums() {
        Weekdays weekDays;
    }
    public static void main(String[] args) {
        UsingEnums myUE = new UsingEnums();
    }
}
```

Using Enum, 1

```
switch (testday) {
      case Saturday:
      case Sunday:
            System.out.println("It's the weekend!");
            break;
      case Wednesday:
            System.out.println("It's Humpday!");
            break;
      case Friday:
            System.out.println("TGIF!");
            break;
      default:
            System.out.println("Back to work!");
}
```

```
UsingEnums myUE = new UsingEnums(Weekdays.Sunday);
```

Using Enums, 2

```
public enum USHolidays {   new_years_day(11),
  ml_king_jr_day(118),
      presidents_day(215), memorial_day( 531),
  independence_day(74),
      labor_day(96),
  columbus_day(1211),veterans_day(1111),
      thanksgiving(1125), christmas(1225);

// more code

if (todayIs == USHolidays.memorial_day.getDay()
  || todayIs == USHolidays.independence_day.getDay()
  || todayIs == USHolidays.labor_day.getDay()

// more code
```

Static import

- Java 5 (Java 1.5) allows import of static items when methods or variables are used repeatedly

```
import static java.lang.Double.parseDouble;
import static java.lang.Integer.parseInt;
public class StaticImportDemo {
   public static void main(String[] args) {
        String intValue = "123";
        String dblValue = "567.89";
        double resultValue = 0;
        try {  resultValue = parseInt(intValue)
                 + parseDouble(dblValue);
               System.out.println("resultValue is "
                 + resultValue);
        }
        catch (NumberFormatException e) {
               System.out.println("Either intValue or"
                             " dblValue not numeric");
        }
        // rest of code
```

Metadata via Annotations

- Java 5 introduced a method for adding metadata to package and type declarations, methods, constructors, parameters, fields, and variables
- Java’s compiler introduces several annotations including java.lang.Override
  - java.lang.Override, indicates method overrides a superclass method allowing a “cleaner” source technique to override code
  - Java compilers also generate errors if annotations are used incorrectly of if other errors occur

Annotation: Override

- Given the following use of Override:

```
@Override
public String accountHtml() {
      // overriding code goes here
}
```

  – Compiler error if the method above does not match the signature of a superclass method

Annotation: Deprecated

- java.lang.Deprecated, flags method or element as deprecated

```
@Deprecated
public class Y2Ktools (
          // deprecated code
}
```

  – Compiler warning if code extends this class

Annotation: SupressWarnings

- java.lang.SuppressWarnings, turns off compiler warnings

```
@SupressWarnings({"unchecked","fallthrough"})
```

Formatted Output

- Even though Java offers the excellent java.text package classes and methods for formatting of data, people with C backgrounds still miss "printf" and its functionality
- Java 5 added java.util.Formatter with new capabilities to all of Java's output methods
  - Formatting may be applied using the locale and may be directed to a "sink" (often a file)
  - PrintStream class includes methods for "format()" and "printf()" using the same formatting characters; "format()" and "printf()" methods are synonymous

Format Characters, 1

• '%b', '%B'	If the argument is null, the result is "false", if it is boolean or Boolean the result is the string returned by String.valueOf(), if it is not null or Boolean the result is "true".
• '%h', '%H'	Formats boolean output as "true"/"TRUE" ("%h"/"%H"), "false"/"FALSE" ("%h"/"%H"), or "null"
• '%s', '%S'	Formats output as String data using argument's formatTo() method (if available) or toString()
• '%c', '%C'	Formats Byte, Short, Character, or Integer as a single character

Format Characters, 2

'%d'	Formats Byte, Short, Integer, Long, or BigInteger as an integer
'%o'	Formats Byte, Short, Integer, Long, or BigInteger as octal
'%x', '%X'	Value (or its hashcode) formatted as hexadecimal integer
'%e', '%E'	Formats Float, Double, or BigDecimal value (exp. notation)
'%f'	Formats Float, Double, or BigDecimal value as floating-point

Format Characters, 3

• '%g', '%G'	Formats Float, Double, or BigDecimal value with less than six significant digits using floating-point notation
• '%a', '%A'	Formats Float, Double, or BigDecimal value with less than six significant digits using floating-point notation with base-16 values for the decimal part and base-10 values for the exponent

Format Characters, 4

• '%t', '%T'	Prefix used for date/time conversions (see below)
• '%%'	Used to print a literal '%'
• '%n'	Platform-specific line separator

Time Characters, 1

- 'H' 2-digit hour using 24-hour clock (leading zero)
- 'I' 2-digit hour using 12-hour clock (leading zero)
- 'k' Hour using 24 hour clock (0-23)
- 'l' Hour using 12-hour clock (1-12)
- 'M' 2-digit minute within hour (leading zero)
- 'S' 2-digit seconds within minute (leading zero)
- 'L' 3-digit millisecond within second (leading zeros)

Time Characters, 2

- 'N' 9-digit nanosecond within second (leading zeros)
- 'p' Locale-specific morning or afternoon marker in lowercase for "%tp" (am/pm) upper case for "%Tp" (AM/PM)
- 'z' RFC 822 time zone offset from GMT, e.g. -0800
- 'Z' String representing timezone abbreviation

Time Characters, 3

- ‘s' Seconds since the beginning of the epoch starting 1 January 1970 00:00:00 UTC (Long value)
- 'Q' Milliseconds since the beginning of the epoch starting 1 January 1970 00:00:00 UTC (Long value)

Date Characters, 1

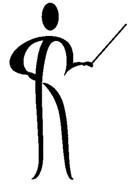
- 'B' Locale-specific full month name ("January")
- 'b' Locale-specific abbreviated month name ("Jan")
- 'h' Same as 'b'.
- 'A' Locale-specific full name of the day of the week ("Sunday")
- 'a' Locale-specific short name of the day of the week ("Sun")

Date Characters, 2

'C'	2-digit year (00-99), four-digit year divided by 100 (leading zero)
'Y'	4-digit year (0000-9999)
'y'	Last two digits of the year (leading zeros)
'j'	3-digit (Julian) day of year (001-366, leading zeros)
'm'	2-digit month (leading zero)
'd'	2-digit day of month (leading zero)
'e'	2-digit day of month (1-31)

Date/Time Characters

- 'R' Time formatted for the 24-hour clock as "%tH:%tM"
- 'T' Time formatted for the 24-hour clock as "%tH:%tM:%tS"
- 'r' Time formatted for the 12-hour clock as "%tI:%tM:%tS %Tp" (morning/afternoon marker ('%Tp') location may be locale-dependent)
- 'D' Date formatted as "%tm/%td/%ty"
- 'F' ISO 8601 complete date formatted as "%tY-%tm-%td".
- 'c' Date and time formatted as "%ta %tb %td %tT %tZ %tY",
  e.g. "Sun Jul 20 16:17:00 EDT 1969"



Special Characters

- Formatting also uses special flags to control print-related functionality like justification, signs, and zero padding.

– '-'	Right-justified output (all data types)
– '#'	Left-justified output (numeric data only)
– '+'	Output includes sign (numeric data only)
– ' '	Output includes leading-space for positive values (numeric data only)
– '0'	Output is zero-padded (numeric only)
– ','	Output uses group locale-specific group separators (numeric data only)
– '('	Output surrounds negative numbers with parentheses (numeric data only)

System.out.format Example 1

- Here is an example of a numeric value being formatted using System.out.format() -- System.out.printf() works identically:

```
double balance = 1234.56;
System.out.format("Balance is $%,6.2f",balance);

    Output:
                Balance is $1,234.56
```

System.out.format Example 2

- Here is an example of a date being formatted using System.out.format():

```
Date today = new Date();
System.out.format("\nToday is %TF %TT",
                      today,today);


    Output:
          Today is 2005-06-09 20:15:26
```

Training Resources

Formatter Examples

- The Formatter class may also be used to format String data anytime, the following example shows the use of the Formatter object and locales:

```
Formatter myUSformat = new Formatter();
Formatter myFRformat = new Formatter();
String balUS =
    myUSformat.format("Balance is $%,6.2f",
                            balance).toString();
String balFR = myFRformat.format(Locale.FRANCE,
          "Balance is $%,6.2f",balance).toString();
System.out.println("US " + balUS);
System.out.println("FRANCE " + balFR);

    Output:
          US Balance is $1,234.56
          FRANCE Balance is $1 234,56
```

Variable Argument Lists

- Java 5/6's new variable argument lists (VarArgs) allow specification of a method that can accept a final parameter of the same time with the number of values to be determined at runtime
  - Only one variable argument list is allowed per method
  - Variable list must be the last argument defined for the method
  - The ellipsis "…" is used to indicate that an argument might appear a variable number of times

Variable Argument Example

- In the Auto class (below) the constructor is expects a variable number of options for any automobile:

```
public Auto (String year, String make,
            String model, String... options) { … )
```

- A variable argument list allow specification of multiple cars with varying lists of options as shown below; (String shown, any object type may be used)

```
Auto johnsToy = new Auto("1969","Fiat","124 Spider",
  "5-speed", "disk brakes");
Auto myTruck = new Auto("1997","Ford","Expedition",
  "Automatic","Four-wheel drive","power windows",
  "power locks","air-conditioning",
  "stereo with cd changer","tinted glass");
```

Scanner Input

- Console input is not common in production programs, but it is very useful when learning Java or creating test modules
- Java programs commonly use use System.in and its "readLine()" method to access the keyboard (requiring that IOException be handled)
- Java 5 introduced the java.util.Scanner class designed specifically to reduce the amount of code needed to communicate with the keyboard

Keyboard Input – Old Style

```
String firstName;
InputStreamReader inStream = new
InputStreamReader(System.in);
BufferedReader inBuf = new BufferedReader(inStream);
System.out.print("Please enter your first name => ");
try {
       firstName = inBuf.readLine();
} // end of first try block
catch (IOException e) {
  System.out.println("Problem reading first name");
  return;
} // end catch block
```

Keyboard Input – New Style

```
String lastName;
System.out.print("Please enter your last name => ");
Scanner fromkeyboard = new Scanner(System.in);
lastName = fromkeyboard.next();
```

Scanner Object Methods

- next() returns the next input buffer String token
- next(comparePattern) or next(compareString) uses patterns to return values
- Numeric variations include:
  - nextBigDecimal()
  - nextBigInteger()
  - nextBoolean()
  - nextByte()
  - nextDouble()
  - nextFloat()
  - nextInt()
  - nextLine()
  - nextLong()
  - nextShort()

Synchronization

- Beginning with Java 5 (Java 1.5) the java.util.concurrent.locks, java.util.concurrent, and java.util.concurrent.atomic packages are available providing better locking support than provided by the "synchronized" modifier.
- All existing code still works as before
- The java.util.concurrent.xxx packages include interfaces and classes used to simplify synchronization and locking

Java.util.concurrent.locks

- The java.util.concurrent.locks.Lock interface has several methods including:
  - lock() to obtain a lock (blocks if can't get lock)
  - unlock() to release a lock
  - lockInterruptibility() gets lock, allows interruptions
  - tryLock() attempts to obtain a lock without a wait.
- The java.util.concurrent.locks.ReentrantLock class behaves like using synchronized does today
- The java.util.concurrent.locks.Condition interface allows complex and multiple conditional waits
- The java.util.concurrent.locks.ReadWriteLock interface allows separate read/write locks

StringBuilder class

- New with Java 5, java.lang.StringBuilder class provides a faster alternative to StringBuffer
  - In most ways StringBuilder works exactly the same as StringBuffer
  - StringBuilder is faster than StringBuffer because it is not ThreadSafe (multiple threads should not access StringBuilder objects without Synchronizing)
  - Use StringBuilder when speed is important in a single-thread environment and use StringBuffer if multiple threads might require access.

StringBuilder Example

```
String myString = "How";
StringBuilder myStrBldr = new StringBuilder("How");

myString += " now";
myString += " Brown";
myString += " Cow?";

myStrBldr.append(" now");
myStrBldr.append(" Brown");
myStrBldr.append(" Cow?");

System.out.println("String = " + myString);
System.out.println("StringBuilder = " + myStrBldr);
```

Java SE Version 6

- Java SE 6 incorporates a broad range of enhancements to the infrastructure of Java rather than specific syntax enhancements
(unlike Java 5)
- Java SE 6 features include:
  - XML and Web services support
  - JDBC 4.0 support
  - More Annotation types
  - More flexible annotation processing
  - Jave compiler APIs accessible from programs
  - Application client GUI enhancements for both AWT and Swing

XML & Web Services Support

- Javas SE 6 address the growth of Web services and XML processing in the Java community including support for:
  - Web Services client stack
  - Streaming API for XML (StAX)
  - Java Architecture for XML Binding (JAXB) 2.0
  - Java API for XML-based Web services (JAX-WS) 2.0 Web services metadata
  - XML digital signature API

New JDBC 4.0 Features

- Java SE 6 includes JDBC 4.0; designed to improve ease of JDBC development by:
  - Simplified access to relational data sources with utility classes
  - Use of generics and annotations
  - Addition of JDBC 4.0 wrapper pattern
  - Safe access to vendor-specific APIs
  - Automatic driver discovery
  - Enhanced connection management
  - New data types (including XML and SQL ROWID)

Annotation-based Development

- Annotations were in Java 5.0 allowing developers to embed metadata in Java source code
- Java SE 6 includes additional built-in annotation types and annotation-processing APIs including:
  - Web services metadata for the Java Platform (JSR 181)
  - Common Annotations for the Java Platform (JSR 250)
  - Pluggable Annotation Processing API (JSR 269)

Java Compiler APIs

- Java command-line compilers receive input from the file system and report errors using a stream
- Java SE 6 allows the compiler to receive input and/or send output to an abstraction of the file system
- Java programs may now specify compiler directives and process compiler output (this feature was add mostly due to software vendor requests)

Application GUI Client APIs

- Java SE 6 enhances application GUI capabilities with changes to both AWT and Swing
- AWT
  - Faster splash screens (using native code)
  - System tray support (icons & messages)
  - Access to browsers and other desktop application “helpers”
- Swing
  - Improved drag-and-drop support
  - Enhanced layout customization
  - Simplified multi-thread programming
  - Writing of GIF images

Other Java SE 6 Features

- Changes to Java class file specification (JSR 202)
- Framework to connect Java programs to scripting-language interpreters (JSR 223)
- New bi-directional (allowing backward navigation) collection classes

Conclusion

- Java 5 provides many features to make the life of developer richer allowing creation of better and more interesting programs
- Generics and the new for loop are probably exciting enough in their own right, but the other features all work together to make this new release the best Java ever
- Java 6 improves the programming environment, especially for JDBC and AWT/Swing programs

Collaborate 2010

COLLABORATE 10

Technology and Applications Forum for the Oracle Community

Save the Date: April 18-22 2010

Las Vegas, Nevada!

ODTUG 2010 KALEIDOSCOPE

Washington, D.C. ▪ June 27–July 1, 2010 ▪ Washington Marriott Wardman Park Hotel

ANNOUNCING ODTUG KALEIDOSCOPE 2010 IN WASHINGTON, D.C.

TOPICS

- Application Express
- Database Development
- Essbase
- Hyperion Application
- Hardcore Hyperion
- Middle Tier and Client-Side Development
- Oracle Business Intelligence, Data Warehousing, and Hyperion Reporting
- SOA and BPM
- Other

Kaleidoscope has it all - more than 170 technical sessions, day-long symposia, hands-on training, chats with participants and speakers, and even a community service project.

Don't take our word for it, here's what one participant said about Kaleidoscope 2009:

"Before the conference, I try to come up with a list of questions that have been bugging me all year, and as usual, I leave with my questions answered. The conference is a great opportunity to ask some tough questions to some great experts."

www.odtugkaleidoscope.com

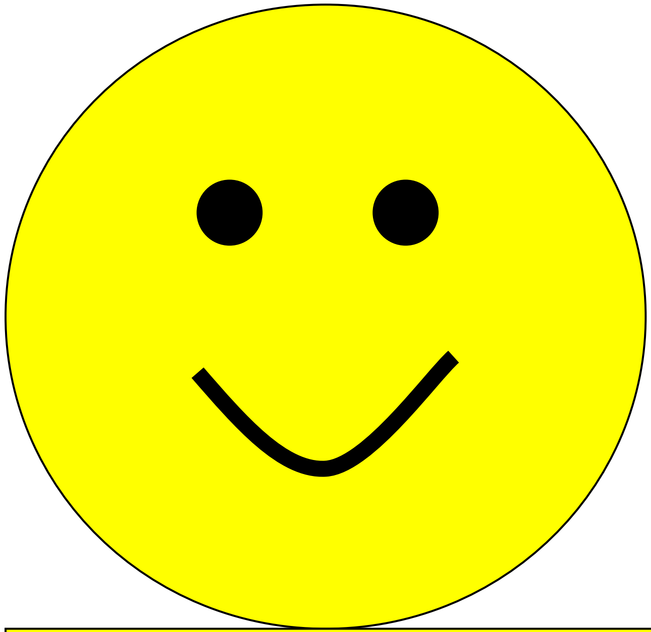
RMOUG
ROCKY MOUNTAIN ORACLE USERS GROUP

Training Days 2011

Watch for the Dates!

Please Fill Out Session Evaluations

Java 6 and Java 5 New Features



To contact the author:

John King

King Training Resources

6341 South Williams Street

Littleton, CO 80121-2627 USA

1.800.252.0652 - 1.303.798.5727

Email: john@kingtraining.com

Thanks for your attention!

Today's slides and examples are on the web:

http://www.kingtraining.com