Discovering Java

By Sergio C. Carbone

By Sergio C. Carbone

Welcome

To a new world of Java

As this course unwinds we will find that Java is really not just a programming language, but a completely new world. There are many ways Java stands apart from conventional programming languages. There are things which can be done with Java that can not be done using other tools.

To a hot emerging technology

Java is not brand new, but because of its history, it is just now starting to make a real name for itself in the systems engineering marketplace. The current release of Java is considered to be the first truly usable release. Now more than ever before companies are looking at Java to implement new approaches to solving their systems need. Like a tsunami rippling the surface of a distant sea, Java is only now starting to give an indication of what is to come. Java will be an important part of a new breakthrough in systems implementations across the world.

To a change

… and after we realized what our true goals were and what we needed to do to survive, we knew we would need to change. Funny how change is so frightening when you see where it is you need to get to, exhausting as you take each step, and most of all rewarding when you are able to look back on your progress and know you have “done good”…

In your products

Java will change the types of products your company will offer and support over the long run. It will make things possible which you haven't thought of in your current environment.

In your approach to systems solutions

Even the way in which you thought of solving systems problems will need to change. Now more than ever you will need to understand the business problems you are trying to solve. You will need to be certain that you are solving the real problem and not just a symptom.

In the way you develop

Most of systems development as you presently know it will change. The overall architecture you will be implementing, interfaces you will develop, the environment you will use, and the language structures will all be different than what you are using currently.

In the way you will do business

With all the changes to your approach to systems development, you will also change the product sets you will offer in the marketplace. Some of these changes will be in type, scale, function, time to market, and price, so you may find you even need new selling strategies.

To a learning process via discovery

The world of Java is completely extensible. Java is object oriented, and you will leverage components built or bought to provide the most innovative solutions to business problems. Because of this infinite expandability, you will find the Java learning process more a process of discovery than lecture.

Introductions

Hello

I am Sergio Carbone, and this is Discovering Java. I am very excited to be here, and I am looking forward to introducing all of you to Java.

Some of my background

I own a computer systems consulting and innovation company. My company's focus is on helping you choose and implement technology, and build better solutions. I specialize in multi-tier client server systems and object oriented technologies. Some other career highlights include:

- Senior manager of a 42 member client server team.
- Senior architect of multi-tier client server systems and object oriented technologies.
- Senior systems engineer of multi-tier client server systems and object oriented technologies.
- Lecturer on multi-tier client server systems, object oriented technologies, and visual development.

My role

- Course guide and lecturer.
- Systems engineering guide and mentor (ongoing).

What about you

Your name and a little description of your background please.

Introduction to Java

Java explained

An introductory explanation about Java. You will find Java to be much more than just a programming language.

A bit about history

To answer the questions of where did Java come from, what was it originally intended for, and where is it heading.

How JBuilder fits in

Let's meet the environment you will use to build your Java solutions.

The help facility

The most important thing you can do is to ask questions. There is a facility to help you get answers fast.

Before the basics

Some preliminary information you will need before any of this makes sense.

The victory program

We came… We saw… We understand it… Let's rewrite the world!

Assignment

Your turn to get your hands on Java.

Java explained

Overview

- Buzzword definition: A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language.
- Java is an environment and not just a programming language.
- There are four main components to Java including:
  - The Java Virtual Machine (VM).
  - The Java language.
  - The Java class library.
  - The development environment.

Simple

- Java is a simple language (but not in implementation).
- Java uses many of the same language constructs as C and C++.
- Java designers removed a number of features available in C and C++.
- Java does not use pointers.

Object oriented

- In Java you will model your application as a collection of objects that will solve the business problem.
- For you to do this you will need to step away from procedure development and start looking at your code from a responsibilities perspective.

Interpreted

- Although Java gets compiled, the Java compiler generates bytecodes for the Virtual Machine (VM), instead of real machine code.

- To run a Java program, you must have the VM interpreter to execute the bytecodes.

Architecture neutral and portable

- Write Once, Run Anywhere.
- Because Java uses bytecodes, a Java application can run on any system that has a VM available.
- By running via the VM, Java applications do not need to be 'ported', and are completely independent of the hardware they run on (if you write them to be).
- Java even controls the size of the basic data types.

Dynamic and Distributed

- Java is a dynamic language.
- Classes can be added to Java at any time.
- Java is a distributed language, because it provides high-level networking support.
- Java's classes make it as easy to read a remote file or resource as it is to read a local file.
- Remote Method Invocation (RMI) allows Java to invoke methods of remote Java objects, as if they were local objects.

Robust

- Java is for writing reliable and robust software.
- Java is a strongly typed language.
- Accesses to arrays and strings are bounds checked at run-time.
- Java's automatic garbage collection prevents memory leaks.
- Exception handling makes for more robust programs.
- Java doesn't eliminate possible of writing buggy software.

Secure

- Java is a secure language.
- This is important due to the distributed nature of Java.
- Java programs cannot point to memory, overflow arrays, or read memory outside a string.
- The Java interpreter performs bytecodes verification to ensure it does not contain illegal bytecodes
- Java employs a "sandbox model", so an applet running in the sandbox has a number of restrictions on what it can do. For example, it has no access to the local file system.
- A digital signature can be established in a secure way to allow for non-sandbox processing.
- No language or environment can be guaranteed 100% secure.

High performance

- Java is not as fast as C.
- Java's speed is fine to run interactive GUI and network-based applications.
- Critical sections of the Java run-time environment are implemented with native code.
- "Just in time" (JIT) compilers translate Java byte-codes into machine code for a particular CPU at run-time.
- With JIT, the performance is nearly as good as native C or C++.
- Portions of programs can be written in C or C++.

Multithreaded

- Java is a multithreaded language.
- It provides support for multiple execution threads.

The Java Virtual Machine (VM)

- The VM can also be called the Java interpreter or Java runtime.
- All Java programs need to be run in a VM. Even applets embedded in Hyper Text Markup Language (HTML) pages run in a VM.
- The VM provides functions that support Java programs.
- Although not an operating system, the VM can be thought of as an operating system agent, which processes Java bytecodes and works with the computer's operating system and peripherals.
- There are standalone VMs, and Java capable WEB browsers come with plug-in VMs.
- There are differences in browser / VM features, so be prepared for differences. Especially in browsers that support operating system specific features.

The Java language

- The Java language is a programming language that is used to create Java applications.
- Java is object oriented, so the concepts of object oriented programming (OOP) are very important.
- Java is based on C++, but has many differences that will make it easier to use.
- Java was designed to be small, portable, but still robust, and it gets most of its robustness from its object oriented extensibility.
- JavaScript is a reduction of the Java language, and is used directly in HTML pages.

The Java class library

- With a class library you stand on the shoulders of those who went before you, so you can pick the fruit others could not.
- A class library is a collection of pre-written objects that you are able to use to solve problems more quickly.

- The idea of a class library is to allow you to reuse code that was written earlier and is known to be bug free.
- Java relies completely on class libraries, and almost all program functions require the use of some class library component.
- Java is completely extensible through the use of new class libraries.

The development environment

- To build Java programs you will need to use one of many Java development environments.
- There are many sources for Java development environments, from free ones found in the back of books or on the WEB, to very expensive professional ones you need to purchase.
- As with many things in life, you get what you pay for, and the professional development environments have many more features than the free environments.
- The development environment will normally include:
  - ✓ An editor to write your Java programs.
  - ✓ A compiler to convert your Java source code into Java bytecodes.
  - ✓ The Java class library, and possibly other class libraries.
  - ✓ A VM testing shell.
- JBuilder, J++, Visual Café, and VisualAge are all commercial Java development environments.

A bit about history

Created by SUN

- Java was developed at Sun Microsystems at the hands of James Gosling and Bill Joy.
- Java was developed from a language for Interactive TV (ITV) called Oak. Oak was abandoned with the failing of ITV.
- In 1993, fueled by the increasing popularity of the World Wide Web, Oak was enhanced and became Java.

Filling the PDA market

- Java originally was intended to be used to power personal digital assistants (PDA) and consumer electronic products.
- After the failing of the Apple Newton, it was obvious that the PDA market was not ready.

It's the Internet now

- Sun has pushed the idea that the network 'is the computer'.
- Currently the enthusiasm Java holds is from its capabilities for building applets used on the World Wide Web
- Java's platform independence is what makes it perfect for the Internet.

Before the basics

Overview

- When you write a Java program, you can write it as an application, an applet or a bean.
- The form of your Java program will be determined by how it will be used.

Applications

- An application is a complete program that needs to be run by a Java VM.
- An application has access to all the facilities of the VM.
- An application has an entry point in the form of a main() method.

Applets

- An applet is a mini-application that needs to be run by a Java enabled Web browser, or in some other applet viewer.
- An applet is often untrusted code, and it cannot access the local file system.
- Applets do not have a main() method.
- To write an applet, you derive from the Applet class and override some of the its methods.
- The Web browser or applet viewer invokes the Applet's methods.
- The applet is not in control of the execution, and responds when the browser or viewer instructs it.

Beans

- A Java bean is a medium grain object that can be implemented in other Java programs from the development environment.
- All windowing components are beans.
- Beans can be very complex.
- Beans are written using the JavaBeans API.

The victory program

Example: Hello World

```
public class HelloWorld
{     public static void main(String[] args)
      {  System.out.println("Hello World!");
      }
}
```

Output:

```
Hello World!
```

- Every Java program consists of a public class definition that contains the main() method.
- Main() is the entry point for all Java applications.

The components of a Java program

- .JAVA Java source file.
- .CLASS Compiled class file (One for each Java class)
- .GIF Graphics file.
- .JPG Graphics file.
- .BMP Graphics file.
- .AU Sound file.
- .WAV Sound file.
- .HTML HTML document.
- .TXT Text file.
- .BAT Batch file.

Starting with basic Java syntax

Basic syntax of a Java program

Overview

- Java is case sensitive ‘Apple’ and ‘apple’ are different to Java.
- Java is completely object oriented, all coding is done through classes.
- Comments can be added to the source code.
- "/*" is used to start a comment and "*/" is used to end a comment.
- Comments in Java wrap lines from the starting "/*" to the ending "*/".
- Never nest comments.
- Java has a single line comment that starts with ‘//’.
- All statements end with a ';'.
- Groups are not statements and do not normally end with ';'.

- Groups are started with '{' and are ended with '}'.
- For easy reading try to line up the start and end of a group in one column.
- Scope or local data is declared after the starting '{' of a group.
- All code statements must be placed in class methods.
- The code statements are placed in a code grouping after the method header.

Primary method.

- The primary method is the starting place for execution.
- In Java this method is called the main method because its name is main().
- All Java applications have only one main().
- One argument is passed to main() for command line arguments.
- One of your classes must have a main().

Primitive data types

The byte data type.

- The byte is a small counting type. It can be used in mathematical implementations.
- The range for a byte is –128 to +127.
- The syntax for allocating a byte follows:

```
byte x = 0; // this allocates a byte called x and sets it to 0
```

The short data type.

- The short is a counting type. It can be used in mathematical implementations.
- It is two bytes or sixteen bits, and has a range of -32768 to +32767.
- The syntax for allocating a short follows:

```
short x = 0; // this allocates a short called x and sets it to 0
```

The int data type.

- The int is a counting type. It can be used in mathematical implementations.
- It is four bytes or 32 bits, and has a range of –2147483648 to +2147483647.

- The syntax for allocating an int follows:

```
int x = 0; // this allocates an int called x and sets it to 0
```

The long data type.

- The long is a counting type. It can be used in mathematical implementations.
- It is eight bytes or 64 bits, and has a range of –9223372036854775808 to +9223372036854775807.
- The syntax for allocating a long follows:

```
long x = 0; // this allocates a long called x and sets it to 0
```

The float data type.

- The float stores floating point numbers and can be used in mathematical implementations, but it is better to use double.
- It is four bytes or thirty-two bits.
- The range for a float is +/-3.4E+38 min or +/-1.4E-45 max.
- The syntax for allocating a float follows:

```
float x = 0.0; // this allocates a float called x and sets it to 0
```

The double data type.

- The double stores floating point numbers and is used in mathematical implementations.
- It is eight bytes or sixty-four bits.
- The range for a double is +/-1.7E+308 min or +/-4.9E-324 max.
- The syntax for allocating a double follows:

```
double x = 0.0; // this allocates a double called x and sets it to 0
```

The boolean data type.

- The boolean data type is use for True / False expression testing. In Java you must use the boolean type for this purpose.
- It is 1 bit long, and the only values it holds are true and false.
- The syntax for allocating a boolean follows:

```
boolean x = false; // this allocates a boolean called x and sets it to false
```

The char data type.

- The char data type is used for storing integer style numbers that represent Unicode characters. You can store ASCII characters in a char.
- Its size is two bytes or sixteen bits.
- The range for a char is \u0000 to \uFFFF.
- The syntax for allocating a char follows:

```
char x = 'A'; // this allocates a single char called x and sets its value to 'A'
```

Declaring Variables

- You can create variables anywhere in a code grouping.
- A variable will be in scope from the point it is declared until the point its code grouping ends with a '}'.
- Variables are automatically defaulted to an appropriate value for their type, but you should still take the time to default them yourself. Most run-time bugs are due to non-initialized data.
- You can name your variables any combination of letters and digits, as long as you start the name with a letter, a '_', or a '$'.
- The standard is to use letters only, with second word capitalization.
- Make the names work for you.
- Avoid type / scope encoding your names.

Basic Operations

Some arithmetic operators

- ++ The increment operator increases the value of a variable. It will increment by one. This alters the variable directly and is not equivalent to "x + 1".
- -- The decrement operator decreases the value of a variable. It will decrement by one. This alters the variable directly and is not equivalent to "x-1".

- \- The negation operator reverses the sign of a value, and does not alter the variable. This is also used for subtraction.
- * Standard multiplication.
- / Standard division.
- + Standard addition.
- += The assignment addition operator adds the value of the right operand to the left operand and stores the new value in the left operand.
- -= The assignment subtraction operator subtracts the value of the right operand from the left operand and stores the new value in the left operand.
- *= The assignment multiplication operator multiplies the value of the right operand with the left operand and stores the new value in the left operand.
- /= The assignment division operator divides the left operand with the value of the right operand and stores the new value in the left operand.
- Some short examples of operator usage follow:

```
int a = 1;
int b = 5;
double c = 4.5;
a++;
--b;
++b;
a += b;
```

What is meant by prefix and post-fix notation.

- Both the increment operator and the decrement operator can be used in prefix or post-fix form.
- Prefix notation is implemented when the operator is placed before the operand.
- The operand is modified before the operand's value is used in the equation.
- A short example of prefix notation follows:

```
int z = 0;
int g = 0;
g = ++z;
```

- In the preceding example, one would be the value of both g and z.

- Post-fix notation is implemented when the operator is place after the operand.
- The operand is modified only after the operand's value is used in the equation.
- A short example of post-fix notation follows:

```
int z = 0;
int g = 0;
g = z++;
```

- In the preceding example, zero is the value of g, and one is the value of z.

Java array objects.

- Arrays are implemented as objects in Java.
- An array is a sequential block of type similar and related data.
- Arrays can be made of any data type.
- Arrays are fixed in size; this size is set at compile time.
- Arrays can have multiple dimensions.
- When working with multiple dimension arrays, storage becomes a concern.
- To create an array, or reference a subscript, the "[]" are used.
- The format is "data_type variable_name[first dimension size] = initialization list;".
- Some examples of declaring arrays follow:

```
int x[5] = new int[5];
int y[4] = {3,66,54,77};
int z[3][5] = {{1,2,3,4,5},{11,22,33,44,55},{111,222,333,444,555}};
// x is an array of five integers not set to anything.  y is an array of four
// integers set to values.  z is a double dimensioned array set to values.
```

- Arrays start at zero, so an array of 3 integers is subscript 0 to subscript 2.

Java string objects.

- The String object is used to declare and manipulate strings.
- Java does not use character arrays to represent strings.
- The String class is used for constant strings when Java encounters a string in quotes.
- The following demonstrates the string class:

```
String s1 = new String ("Hello World.");
char cArray[] = {'J', 'B', 'u', 'i', 'l', 'd', 'e', 'r'};
String s2 = new String (cArray);       //s2 = "JBuilder"
int i = s1.length();                   //i = 12
char c = s1.charAt(4);                 //c = 'o'
i = s1.indexOf('l');                   //i = 2 (the first 'l')
String s3 = "abcdef".substring (2, 5)  //s3 = "cde"
String s4 = s3.concat ("f");           //s4 = "cdef"
String s5 = valueOf (i);               //s5 = "2" (valueOf()is static)
```

Java StringBuffer objects.

- The StringBuffer is similar to the String object, but the StringBuffer does more.
- StringBuffer objects can be modified (the String object cannot).
- The StringBuffer uses more memory than the String object.
- The following is an example of the StringBuffer:

```
StringBuffer s1 = new StringBuffer(10);
int c = s1.capacity();         //c = 10
int l = s1.length();           //l = 0
s1.append("Bor");              //s1 = "Bor"
s1.append("land");             //s1 = "land"
c = s1.capacity();             //c = 10
l = s1.length();               //l = 7
s1.setLength(2);               //s1 = "Bo"
StringBuffer s2 = new StringBuffer("Hello World");
s2.insert (3, "l");            //s2 = "Hello World"
```

The System object (class).

- The System object provides system level platform independent resources.
- System's methods provide functionality such as getting the current system time, forcing garbage collection, and loading dynamic link libraries.
- System's data members are used to interact with the system.
- These variables include 'in' which represents the standard input stream, 'out' which represents the standard output stream, and 'err' which represents the standard error stream.

The main sources for classes.

- The first place to look is the JDK library. This comes with the environment.
- Mostly all business specific classes will be written in-house.
- Very complex classes are sometimes given to library development groups.
- Library development groups are normally in-house teams.
- Often classes that serve a generic role are purchased from software vendors.

The layout of methods in Java.

- A method has four main areas: the name, return type, arguments, and code group.

```
return-type name(arguments)
{ code group
}
```

- The method's code group has three areas: declarations, code statements, and return.

```
int test_function()
{ int x = 0; /* local declaration */
x = x + 1; /* code statement */
return x;  /* return */
}
```

- The code grouping is started with '{' and ended with '}'.
- Local declarations and arguments are scope data, and their values are valid only until program control leaves their scope.
- Local declarations and arguments are only available to the method.
- Methods should have only one exit or return point.
- Methods should be relatively short and focused.

The method arguments.

- Java can pass data values to functions.
- Java passes all arguments by reference capability.
- Arguments are declared and placed within the parentheses.

- Arguments are separated by commas.
- An example of an argument follows:

```
void func(int x) /* passed an int */
{ x = 0;      /* x is a reference of the int s */
}
…
int s = 5;
anobject.func(s);
…
```

- The syntax of arguments is just like other data.
- Don't panic, the method tells you exactly what it wants to be passed.
- Just give the method the data it needs.

The return value.

- The return value is the data the method returns to the calling body.
- The return value is used to give an outcome of the method call.
- Return values are often used to return error status codes.
- A method can have a return value of void, meaning it has no return value.
- A method can have only one return value.
- To use a return value, set the return type to the type of data to be returned.
- At the end of the method, use the return statement with the data to be returned.
- In void functions, return is not given a value and causes an immediate return.
- An example of return values follows.

```
int func(int x) /* passed an int */
{ return x * 2; /* we return x * 2 */
}
…
int t = 1;
t = anobject.func(t); /* pass the value of t and store the
                  return in t */
…
```

Conditional Execution

How true and false works in Java.

- There is a formal Boolean data type in Java.
- Integers are not used to handle Boolean data.
- A comparison or method call with a boolean return value is needed to test for true or false.

The logical operators in Java.

- The operators that are used for logical or relational expressions are:

()	Grouping precedence.
!	Not.
>	Greater than.
>=	Greater than or equal to.
<	Less than.
<=	Less than or equal to.
==	Equal to.
!=	Not equal to.
&	And.
|	Or.
^	XOr
&&	Short-circuit And.
||	Short-circuit Or.

Executing conditionally with an if statement.

- Conditional execution is most simply handled with an if statement.
- There are two formats for an if statement in Java.
- The single statement conclusion format.

```
if (expression)
```

```
  conclusion;
```

- The multiple statement conclusion group format.

```
if (expression)
{ conclusion;
  ...
}
```

- "Then" is not used in Java if statements.
- The expression is evaluated, and if it is "true" the conclusion is executed.

Trapping condition failure with an else statement.

- There are times when code needs to be executed only when an expression is false.
- An else statement can be added to an if statement to trap false expressions.
- There are two formats for an else statement in Java.
- The single statement conclusion format.

```
if (expression)
  conclusion;
else
  conclusion;
```

- The multiple statement conclusion group format.

```
if (expression)
  conclusion;
else
{ conclusion;
  ...
}
```

- The expression is evaluated, and if it is "false" the else conclusion is executed.

General rules regarding if..else statements.

- The else is matched with the last if in its current scope.
- If statements do not need to have an else statement.
- Else statements must be immediately preceded by an if statement.
- The if conclusion and else conclusion will never both be executed.

Nesting if and if..else statements.

- Nesting is used when it is necessary to place an if or if..else statement as part of a conclusion.
- Nesting increases code complexity, but it is often required.
- Large multiple level nested if or if..else statements should be avoided by the use of function calls.
- Nesting causes problems with the matching of if..else statements. These problems can be overcome by using the multiple statement conclusion format.
- Some formats for nested if and if..else statements follow.

```
if (expression)   /* Simple nested if */
  if (expression) /* Matching if */
    conclusion;
  else            /* Matching else */
    conclusion;

if (expression)   /* Matching if */
{ if (expression) /* multiple statement conclusion*/
    conclusion;
  ...
}
else              /* Matching else */
  conclusion;
if (expression)   /* Match A */
  if (expression) /* Match B */
    conclusion;
  else            /* Match B */
    conclusion;
else              /* Match A */
  if (expression) /* Match C */
    conclusion;
  else            /* Match C */
    conclusion;

if (expression)
{ if (expression)
    conclusion;
}
```

```
else
  if (expression)   /* Else with if */
    if (expression) /* Simple nested if */
      conclusion;

if (expression)   /* Matching if */
{ if (expression)
    conclusion;
}
else              /* Matching else */
{ if (expression) /* multiple statement conclusion*/
    conclusion;
  ...
}
```

Executing conditionally with a switch..case statement.

- Switch..case statements can take the place of many if..else statements that test only for equality.
- Switch..case statements are cleaner and easier to read than nested if..else statements.
- The format for switch..case statements follows:

```
switch (variable)
{ case constant 1 :
    statements;
    ...
    break;
  case constant 2 :
    statements;
    ...
    break;
  case ...
  default:
    statements;
}
```

- The variable is compared to each constant.
- If a match is found, execution starts with the statements for that case, and continues until a break statement is processed.

- If a break is not processed, execution continues even into the next case.
- If no match is found, execution starts with the default case.
- Variables can be defined directly after the opening group symbol of a switch..case statement.
- A case does not need to have statements for it. Execution will start with the next statement found, and continue until a break is encountered.
- The switch variable must be convertible to an integer.
- The case values must be literals or constants that can be converted into an integer.
- The case values must be unique.
- Strings can not be used in switch..case statements.

The use of a default case.

- The default case is not required.
- If no matches are found, execution starts at the default case.
- If no matches are found, and there is no default case, execution continues after the switch..case statement.
- If a match is found before the default case, and no break statement is encountered, execution continues into the default case.

How the break statement is used with switch..case statements.

- The break statement is not required.
- The break statement forces execution to jump to the first statement after the switch..case statement.
- In most implementations, the break statement is used with each case.
- The break statement is not required in the very last case in a switch..case statement.
- The break statement can be conditionally executed, but this will produce confusing Java code.

Altering values conditionally with the ?: operator.

- The ?: allows one of two values to be return based on a condition.
- The format of the ?: operator follows:

```
condition ? true expression : false expression;
```

- If the condition evaluates "true", the true expression is evaluated.
- If the condition evaluates "false", the false expression is evaluated.
- Unlike an if..else statement, the ?: operator can be placed within an expression.

Nesting the ?: operator.

- The ?: operator can be nested, but the syntax is very difficult to understand.
- Some short examples of the ?: operator follow:

```
int y = (x > 5) ? 5: (x <3) ? 3 : x;
```

- Nested ?: operators can be difficult to follow, and the above example had only one nested ?:. The following only has two nested ?:.

```
int z = x < 3 ? y < x ? y : z : z > y ? z : x; /* ouch! */
```

Boolean short circuiting.

- Boolean short circuiting is a way of evaluating expressions with the fewest tests.
- Evaluation stops as soon as the expression is proven "true" or "false".
- "And" expressions need to find only one portion "false" to stop evaluation.
- "Or" expressions need to find only one portion "true" to stop evaluation.
- In the follow examples the right expressions would not be evaluated.

```
int x = 0;
int y = 5;
int z = 10;
if (x < 1 || z > y) ...
if ( x && z > y) ...
```

- If an expression is a method call or meaningful code, the function call or code would not be executed.
- Boolean short circuiting can lead to unexpected program execution.

Iterative Execution Control

An overview of iterative control.

- Looping is used to repeat a sequence of program statements multiple times.
- All loops in Java have a condition expression that must evaluate to true for the loop to continue.
- The condition expression can be made from almost any combination of code. It must evaluate to a true or false value.
- In most loops the condition is tested before the first execution of program statements takes place.
- In some loops the first execution of program statements takes place before the condition is tested.
- There are three styles of loops in Java: a "for" loop, a "while" loop, and a "do..while" loop.
- The number of iterations a loop makes can be predetermined with a counter, or the loop can be controlled by some variable condition such as end of file.

Using a "for" loop.

- A "for" loop is thought of as the most controlled loop.
- The Java "for" loop is similar to Pascal, BASIC, or many other languages.
- A "for" loop is commonly thought of as a counting loop.
- The basic syntax of a "for" loop has four main components: the initialization statement, the condition expression, the program statement or grouping, and the increment statement.

```
for (initialization; condition; increment)
  program statement;
for (initialization; condition; increment)
{  grouping
}
```

- The initialization statement is normally used to set the starting value of the loop counter. It is only executed once and occurs before the loop starts.
- The condition expression is evaluated at the start of every iteration. If it evaluates to true, the program statements are executed. If it evaluates to false, the loop is ended.
- The program statement or grouping is the code that will be executed each iteration.
- The increment statement is executed at the end of every loop and is normally used to increment the loop counter.
- Normally the initialization statement, condition expression, and increment statement all work with a loop counter, but they do not need to be related in any way.
- Although it is not recommended, the initialization, condition, or increment can be empty as long as the ';' is used as a place holder.
- A "for" loop does not need to have any program statements.
- An example of a "for" loop follows:

```
void func()
{ int x = 0;
  for (x = 0; x < 5; x = x + 1)
    system.out.println(x);
}
```

Using a "while" loop.

- A "while" loop is thought of as the simplest loop.
- The Java "while" loop is similar to Pascal, BASIC, or many other languages.
- A "while" loop is not commonly thought of as a counting loop.
- The basic syntax of a "while" loop has two main components: the condition expression and the program statement or grouping.

```
while (condition)
  program statement;
```

```
while (condition)
{  grouping
}
```

- The condition expression is evaluated at the start of every iteration. If it evaluates to true, the program statements are executed. If it evaluates to false, the loop is ended.
- The program statement or grouping is the code that will be executed each iteration.
- The program statement or grouping must have some affect on the condition or the loop will be infinite.
- A "while" loop does not need to have any program statements.
- An example of a "while" loop follows:

```
void func()
{ int a = 150;
  while (a > 0)
  { --a;
    system.out.println(a);
  }
}
```

Using a "do..while" loop.

- A "do..while" loop is thought of as the least controlled loop.
- A "do..while" loop will always execute at least once, and is not commonly thought of as a counting loop.
- The basic syntax of a "do..while" loop has two main components: the condition expression and the program statement or grouping.

```
do
 program statement;
while (condition);
do
{  grouping
} while (condition);
```

- The condition expression is evaluated at the end of every iteration. If it evaluates to true, the program statements are executed again. If it evaluates to false, the loop is ended.
- The program statement or grouping is the code that will be executed each iteration.
- The program statement or grouping must have some affect on the condition or the loop will be infinite.
- An example of a "do..while" loop follows:

```
void func()
{ int a = 0;
  do
  { system.out.println(a);
    a++;
  } while (age < 150)
}
```

Using the break statement with iterative control.

- The break statement can be used to stop a loop.
- The break statement forces execution to jump to the first statement after the loop.
- In most implementations, the break statement can be avoided.
- The break statement can be conditionally executed, but this will produce confusing Java code.
- An example of the break statement follows:

```
void func()
{ int a = 150;
  for (;;)
  { system.out.println(a);
    if (a <= 0)
      break;
  }
}
```

Using the continue statement with iterative control.

- The continue statement can be used to skip to the "bottom" of a loop.
- In a "while" or "do..while" loop the continue statement causes execution to go directly to the condition.
- In a "for" loop the continue statement causes execution to go directly to the increment statement.
- In most implementations, the continue statement can be avoided.
- The continue statement can be conditionally executed, but this will produce confusing Java code.
- An example of the continue statement follows:

```
void func()
{ int a = 150;
  do
  { a--;
    if (a <= 0)
      continue;
    system.out.println("a > 0.");
  } while (a > 0);
}
```

Compiling and running your program

Compiling

- Even though Java only gets translated into bytecodes, your Java programs still need to be compiled.
- Compiling produces a separate .class file for each class declaration and interface declaration in a source file.
- When a Java program is 'run', the VM runs the bytecodes contained in the .CLASS files.
- JBuilder uses dependencies checking, so a file is recompiled only if it depends on some program item that has changed. JBuilder keeps this dependency information in a file with the extension of .DEPENDENCY.
- You can compile your application from the IDE or the command line using the Borland Maker for Java (BMJ) or the Borland Compiler for Java (BCJ).
- To compile from the command line you will need some environmental variables to be set. You can use SETVARS.BAT to help set these variables.
- Make compiles only what needs to be recompiled based on the dependency information. To 'make' your project, select Make under Build from the menu.
- Rebuild compiles all the project components (every time you ask for a rebuild). To 'rebuild' your project select Rebuild under Build from the menu.
- Error messages are displayed in an Error pane; correct them! By double clicking an error message it will take you to the line of code that caused the error.
- If you change your compiler option remember to perform a rebuild.
- You can speed up your Java implementations by using a just-in-time (JIT) compiler, which converts the bytecodes to machine code.

Running you application

- You can run your application from within the IDE by selecting Run under Run from the menu.
- JBuilder will run JVIEW.EXE if your program is an application and the Applet Viewer if your program is an applet.
- You can run an application from the command line by running the JVIEW.EXE with the class pathname of your application's 'main' class.
- You can run an applet through a Java enabled WEB browser by opening the applet's HTML file with the browser.

More About Object Mechanics

Member Function Overloading

What Is Member Function Overloading

- Member function overloading allows two completely different member functions to share the same name.
- The members need to differ by the types of arguments that they are passed.
- The return values or argument names have no effect.
- In Java, the method name is not just the name given to the method alone. It is the combination of the text name and the argument types the member function is passed that forms the name of the member function.
- Java combines the text name with the argument types to form a distinct name at compile time.
- This process of embedding the argument type in the distinct name is known as name mangling.

- Member function overloading is useful when there are multiple functions that do similar tasks but require different arguments.
- Care must be used when implementing overload member functions.
- All member functions other than the finalizer can be overloaded.
- Member function overloading is not polymorphism.
- A short example of member function overloading follows:

```
void func(int x){…}  // Method header that takes an int.
void func(char x) {…} // Method header that takes an char.
…
int y = 0;
anobject.func(y);          // Calls func that's passed int.
anobject.func('A');        // Calls func that's passed char.
…
```

Static Data and Methods

How Static Data Members Work With Classes

- In Java, static data takes on a new and slightly different meaning when used in a class.
- Static data members are not placed with the other instance data, and are only stored once. They are not repeated within every instance, but rather are shared between all instances of a class.
- Since the data is shared, its value will be the same in each instance.
- Static data can be referenced using the class name or any instance name.
- Static data can be referenced even when no instances of the class are available.
- To declare static data, just use the static key word.
- Static data is better than other languages' global data, because static data can be given access specifiers and is better organized.
- A short example of static data follows:

```
class MyClass
{ static public int x = 0;       // Used the static keyword.
  …
```

```
};
```

How Static Methods Work With Classes

- Static methods are used to work with static members.
- When a normal member function is called, the instance must be referenced in the call.
- Java passes the address of the instance to the member function as a hidden argument by the name "this"; consequently, "this" is a reference to the instance that was referenced in the call.
- Java uses the "this" reference as a base address when referring to any class data.
- Static member functions are member functions that are not passed a "this" reference.
- Because static methods do not have a "this" reference, they can not use normal member data or call normal member functions.
- Static methods can only use static members, or non-member functions and data.
- Static methods can be called by normal member functions.
- Static methods can be called using the class name or an instance name, but even if an instance name is used to call the static method, "this" is still not passed to the function.
- A short example of static methods follows:

```
class MyClass
{ static int x = 0;            // Used the static keyword.
  static public void setx(int y){…} // A static function.
}
MyClass.setx(0);   // The call to a static method.
```

Access Specifiers

Public, Protected, and Private

- The only security available for data or functions in many languages is data hiding.
- Java provides real data security for classes through the use of access specifiers.
- Access specifiers limit the ability for non-member code to access data or call member functions.
- To use an access specifier, simply type it before the member you wish it to be applied to.
- There are three access specifiers available:
  - ✓ Public allows any function to access the member.
  - ✓ Protected allows only members, and child class members to access the member.
  - ✓ Private allows only members to access the member.
- A short example of access specifiers follows:

```
Class myclass
{ private int x;
  public int getx() { return x; }
  public void setx(int y) { x = y; }
}
… myclass obj = new myclass;
obj.setx(29);
System.out.println(obj.getx());
…
```

When To Restrict Access

- Provide enough security to prevent interference with class operation.
- Be free enough to allow for sensible reuse.
- Although many "experts" recommend complete data encapsulation, a class must be convenient or no one will use it.

Exception Handling

What Is Exception Handling

- With object oriented programming the old methods of trapping errors are not always effective.
- Exception handling is a new way of trapping and gracefully handling errors.
- Exception handling stops execution at the point an exception is thrown, and moves execution flow to error handling blocks.

How To Implement Exception Handling

- There are three components to exception handling:
  - ✓ A "try" code block notifies the compiler that a "catch" block is available.
  - ✓ One or more "catch" code blocks.
  - ✓ One or more "throw" statements.
- One catch must be provided for each type of data thrown.
- The "catch block" handles an error thrown to it.
- Throw statements transfer control to the catch block that matches the data type that was thrown.
- If no catch matches the throw, the program will not compile or terminates.
- A short example of exception handling follows:

```
class MyClass
{ void func()
  { if (…)                  // Error detected.
      throw("Error Text");  // Throw the (string) exception.
  }
}
… try                   // Allow for catching.
  { MyObject.func();    // Call func() that throws exception.
  }
  catch(string text)     // Catch a string exception.
  { …                    // Do something.
  } …
```

Packages

Overview

- A package in Java is a group of class files that where designed to be used closely together.
- These class files are declared to be part of one package by placing the keyword 'package' followed by the package name at the top of each of the files.
- The package keyword indicates to Java that the classes within the class files are closely related, and that they should be aloud to have special access privileges.

Packages and access specifiers

- In Java there is a special access privilege know as package.
- Package is similar to other access specifiers such as public, protected, or private, but it is different.
- When a class is part of a package, it has access to all classes in all the package's class files.
- When a class is part of a package, all the members of the class have complete access to all non-private members of all the package's classes.

The default package

- In Java if you do not specify a package by using the package keyword, the classes become part of a default package.
- By being part of a default package, Java treats all the classes as if they were listed as members of the same package.
- It is best to clearly label each of your class modules as the packages you intend them to be a part of.
- You take risks by allowing your classes to be part of the default package, but programming is simpler.

Inheritances

What Is Inheritance

- Inheritance is when a new class uses an existing class as a starting point of basis for its behavior set.
- The new class is referred to as the derived class or sub-class, and the existing class is referred to as the base class or super class.
- The derived class takes on all members of the base class, but it can only access the protected and public members directly.
- When the derived class inherits the properties of the base class, it can keep the access specifiers of public and protected members as they exist in the base class, or make more secure in the derived class.
- The derived class can then override or add functionality to the rule set, but it cannot remove functionality.
- In Java, the members of the base class are accessed by using the 'super' keyword.
- The order of construction is base to derived.
- Java does not support multiple inheritance.

How Inheritance Promotes Code Reuse

- Inheritance promotes code reuse by allowing the programmer to expand the behavior of an existing module by only coding the changes and not duplicating the existing module.
- When using inheritance the original class is not touched, so there is no impact on existing code.
- Inheritance allows the programmer to develop new modules and still be guaranteed of a properly working frame.
- Inheritance allows for faster debugging and testing new code.

How to Implement Inheritance

- In Java the 'extends' key word is used to derive a new class from an existing class.
- The following example shows a derived class 'b' being extended from an existing class 'a':

```
Class a
{ …
  a(int x)
  { …
  }
}
class b extends a
{ …
  b(int x)
  { super(x)
    …
  }
}
```

The Role of Abstract Functions and Classes

- Many times the base class that contains the first version of a function just has an empty function body.
- At the base class level much code is invalid because of the vast generalities of a base class.
- Often a programmer wants to force the user of the class to provide the function body in the derivations.
- By specifying a function as abstract using the abstract keyword, the class user is forced to supply the function body.
- A class containing an abstract function needs to be marked abstract, and no instances can be made from it.
- Only through derivation can the function body be provided for the abstract function, and only after all abstract functions have been satisfied can an instance be made.

How Abstractions Are Used

- Abstractions are used as patterns for a class hierarchy.
- An abstraction is too general and undefined to support precise code.
- Typically a system will be written to only use the methods of the abstraction. As long as the system deals only with the guaranteed behavior of the abstraction new derivations can be added. This eases adding new derivations and promotes true system flexibility.
- An example of an abstraction follows:

```
abstract class a
{ …
  abstract void func1();
}
class b extends a
{ …
  void func1()
  { …
  }
}
```

Where Interfaces Fits In

- An interface is a declaration that declares constants and method declarations, but not can provide any function bodies or variables.
- An interface is like an abstract class.
- Interfaces are meant to replace multiple inheritance.
- All members of an interface are public.
- The following is an example of an interface declaration:

```
interface x
{ int func1();
  int func2();
}
```

- To use an interface you can use the ‘implements’ keyword.
- A class that uses an interface must supply all of the function bodies for the interface.

- A class can implement as many interfaces as it needs to.
- The following is an example of the 'implements' keyword

```
class a implements x
{ int func1()
  { …
    return 0;
  }
  int func2()
  { …
    return 0;
  }
}
```

Accessor methods

Overview

- In object oriented programming member data is often accessed only through public member functions.
- These public functions that access the member data ensure that the data is being handled in the correct way.
- When writing or using JavaBeans all data members need to have access methods
- There is a standard for the way these access member function are declared.
- The standard is to use 'get' and 'set' methods.

Get methods

- The 'get' method is used to retrieve the value of a piece of member data.
- The method name is the same as the property, but is preceded by 'get'.
- A data member named 'x' would have a 'get' method named 'getx()'.
- Boolean data is handled with an 'is' method rather than a 'get' method.
- A boolean data member named 'y' would have an 'is' method named 'isy()'.

Set methods

- The ‘set’ method is used to set the value of a piece of member data.
- The method name is the same as the property, but is preceded by ‘set’.
- A data member named ‘x’ would have a ‘set’ method named ‘setx()’.
- Boolean data is handled as any other data.

Class Library Organization

What Makes A Good Class Library

- Very few class libraries that have been developed as class libraries are ever used.
- Most well used class libraries are derivatives of code previously implemented.
- The purpose of a class library is to make future development easy, quick, maintenance free, and reliable.
- For a class library to be easily used, it needs to be simple to implement.
- If it requires major efforts to understand and implement the classes, no one will use it.
- Or worse, only one person will know how to use it, and that person will be forced to become deeply involved with every project.
- A good class library is well focused, and will have very few required options.
- Required options should not be confused with available options.
- It is necessary for a class library to allow enough flexibility to be useful.
- For a class library to support quick application development, it must provide holistic solutions for business problems.
- Many class libraries fail because they provide only partial solutions.
- If a class library requires the application developer to invest time into coding missing portions, they will find it an inconvenience to use.
- Class libraries should provide "plug-in" solutions to entire problems.
- A good class library provides for complete solutions to a business problem, and allows "snap-together" applications development.
- For a class library to be maintenance-free, it needs to be based on a solid well-rounded design.

- Although it is very important to have the best-written and safest code in a class library, most maintenance problems surface due to poor design or retrofitting.
- There is a big difference between poor code and poor design.
- It is important for any system or class library to be "cool running".
- When changes need to be made, the class library needs to allow for quick changes that don't affect other portions of the class library.
- A class library needs to be written dynamically to eliminate limitations.
- For a class library to be reliable, it needs to provide for consistent documented behavior.
- A good class library is a library that exhibits professionalism.
- All functionality and usage requirements should be clearly documented.
- The class library must perform as documented.
- Minor version releases should only add and not change the existing working behavior.
- The class library should be subject to SCM.
- It is always better to buy rather than build.
- Most non-business problem focused libraries have already been written.
- Do not reinvent the wheel.

JavaBeans components

Overview

- A JavaBean component is a medium grain object that complies with the JavaBeans specification.
- The JavaBeans specification is a set of rules and naming standards that a component must comply with to be a bean.
- The internal operation a bean is hidden from the user.
- JavaBeans are often referred to as components.
- Beans are the building blocks of applications.
- Beans represent visible parts of the user Interface and nonvisual elements such as databases.

Life as A Bean

- A bean is any class that conforms to the JavaBeans component model.
- If a class conforms to the model, it can work with most visual design tools.
- A bean is a public class with a constructor that takes no arguments.
- All accessible properties have accessor methods.
- A bean can handle events with registration methods.
- Its appearance and behavior can be user modified.
- A bean can save its state and is said to have persistence.
- It supports introspection, so it can be used in most development tools.

User Interface (UI) Beans

- A UI bean can draw on the screen and become part of the application's user interface.
- A UI bean does not need to actually be visible on the screen, but it does need to have the ability to paint to the screen.
- Buttons, text boxes, checkboxes, labels, and image beans are examples of UI beans.
- UI components are extended from a class that can draw on the screen, and descend from java.awt.Component.
- The easiest way to make a UI bean, is to extend an existing component. By starting with an existing component, you are certain that you adhere to the JavaBeans standard.
- UI beans are sometimes known as controls.

Non-User Interface Beans

- Non-UI beans do not need to paint to the screen, but they still need to follow the JavaBeans standard.
- Non-UI beans support other controls on the user interface, and often interact with UI beans.
- An example of a non-UI bean is a database bean.

Menu Beans

- Menus are UI beans that are access from the Component Tree, not the UI Designer.
- The Menu Designer is used to visually build the menus for your application.

Composite Beans

- Composite beans are beans that are built from other beans.
- Typically these beans are collections, views, painters, editors, and view managers.
- By building a bean as a composite or composition, it is easier to separate the data from the way it is presented.

Lightweight Beans

- Originally, UI beans that were derived from JavaAWT were derived from java.awt.Canvas or java.awt.Panel and had a redundant windowing facility that the java.awt.Canvas and java.awt.Panel brought with them.
- These beans are known as heavy weight beans, because of the extra code.
- Beans that are derived from java.awt.Component or java.awt.Container do not have the extra native windows code associated with them.
- These beans are known as lightweight beans.

Selecting A Bean

- Some of the available bean libraries include Borland's JBCL beans, JavaAWT and Swing, and dbSwing.
- In most cases there is a general similarity between bean libraries, and they can be used interchangeably.
- Compare JBCL, Swing, dbSwing, and AWT beans that perform similar tasks to decide on the best bean to use.
- Often there are a number of beans that can fit your need, but one may look nicer, work nicer, be smaller, or easier to use.
- Generally, the JBCL have the most features.

The Java Class Libraries

Overview

- The purpose of this section is to discuss the types of classes found in the Java class library and other class libraries.
- Try to leverage what is in class libraries as much as possible, so you can have a head start on your development efforts.
- Always look for the simplest implementation that will satisfy your needs.
- Think forward, but don't exaggerate your needs.
- The Java class library is completely extendable and infinite.
- Remember what types of services are available, but not the details of each class.
- This section is not intended to be a reference manual.

The JDK Packages

Overview

- The JDK includes support for database connectivity, GUI design, I/O, and network programming.
- Applet supports applets.
- Language is main core of the Java language.
- Utilities support utility data structures.
- I/O supports various types of input/output.
- Beans support Java beans.
- Math supports large-scale math
- Networking supports TCP/IP and socket programming
- AWT support GUI design and event-handling
- Text supports internationalization
- Security supports cryptographic security
- RMI supports distributed programming
- Reflection supports run-time class information
- SQL supports querying databases using SQL

The Applet Package

Applet is used to create a small program that is intended to be embedded inside another application.

The Language Package

- Boolean is a wrapper class.

- Byte is a wrapper class.
- Character is a wrapper class.
- Class represents classes and interfaces in a running Java application.
- ClassLoader is implemented to extend how the VM dynamically loads classes.
- Compiler provides support for Java-to-native-code compilers and related services. Double is a wrapper class.
- Float is a wrapper class.
- Integer is a wrapper class.
- Long is a wrapper class.
- Math contains methods for performing basic numeric operations.
- Number is the superclass of classes Byte, Double, Float, Integer, Long, and Short.
- Object is the root of the class hierarchy.
- Process can be subclasses to control the process and obtain information about it.
- SecurityManager is an abstract class that allows applications to implement a security policy.
- Short is a wrapper class.
- String represents character strings.
- StringBuffer extensive string support.
- System contains a mix of class to help with general programming.
- Thread provides support for multiple threads of execution running concurrently.
- ThreadGroup is a set of threads or other groups.
- Throwable is the superclass of all errors and exceptions.
- Void is a wrapper class.

The Utilities Package

- Calendar provides calendar features.
- Date represents dates and times.
- Dictionary is the parent class of Hashtable.
- Hashtable allows key associations with values.
- SimpleTimeZone represents a time zone.
- Stack allows arrangement of objects in a stack.
- StringTokenizer breaks String up into tokens.
- Vector implements a dynamic array of objects.

The I/O Package

- BufferedInputStream implements a buffered input stream.
- ufferedOutputStream implements a buffered output stream.
- BufferedReader reads text from a character-input stream.
- BufferedWriter writes text to a character-output stream.
- ByteArrayInputStream reads from a byte array as if it was a stream.
- ByteArrayOutputStream writes to a byte array as if it was a stream.

  CharArrayReader reads from a character buffer as if it was a stream.
- DataInputStream reads primitive data types in a machine-independent way.
- DataOutputStream writes primitive data types in a portable way.
- File represents the name of a file or directory.
- FileDescriptor represents an open file or an open socket.
- FileInputStream an input stream for reading data from a File or FileDescriptor.
- FileOutputStream an output stream for writing data to a File or FileDescriptor.

- FileReader is for reading character files.
- FileWriter is for writing character files.
- FilterInputStream is the superclass of all classes that filter input streams.
- FilterOutputStream is the superclass of all classes that filter output streams.
- FilterReader an abstract class for reading filtered character streams.
- FilterWriter an abstract class for writing filtered character streams.
- InputStream is the superclass of all classes representing an input stream of bytes.
- InputStreamReader is a bridge from byte streams to character streams.
- LineNumberInputStream is an input stream filter that keeps track of the current line number.
- LineNumberReader is a buffered input stream that keeps track of line numbers.
- ObjectInputStream de-serializes primitive data and objects previously written using an ObjectOutputStream.
- ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream.
- ObjectStreamClass describes a class that can be serialized to a stream
- OutputStream is the superclass of all classes representing an output stream of bytes.
- OutputStreamWriter writes characters to an output stream
- PipedInputStream is the receiving end of a communications pipe.
- PipedOutputStream is the sending end of a communications pipe.
- PipedReader provides piped character-input streams.
- PipedWriter provides piped character-output streams.
- PrintStream prints values and objects to an output stream.
- PrintWriter prints formatted representations of objects to a text-output stream.
- PushbackInputStream is an input stream filter that provides a buffer into which data can be "unread."

- PushbackReader is a character-stream reader that allows characters to be pushed back into the stream.
- RandomAccessFile supports both reading and writing to a random access file.
- SequenceInputStream allows an application to combine several input streams serially and make them appear as if they were a single input stream.
- StreamTokenizer takes an input stream and parses it into "tokens", allowing the tokens to be read one at a time.
- StringBufferInputStream reads from a string as if it was an input stream.
- StringReader reads from a string as if it was an input stream.
- StringWriter writes to a string as if it was an input stream.

The Beans Package

- BeanDescriptor provides global information, such as displayName, about a "bean".
- Beans provides general purpose beans control methods.
- EventSetDescriptor describes a group of events that a given Java bean produces.
- FeatureDescriptor is the common baseclass for classes like PropertyDescriptor.
- IndexedPropertyDescriptor describes a property that acts like an array and has an indexed read and/or indexed write method to access array elements.
- ntrospector provides a standard way for tools to learn about bean properties, events, and methods.
- MethodDescriptor describes an public method of a bean.
- ParameterDescriptor supports a bean in providing information on method parameters.
- PropertyChangeEvent is generated if a bean changes a "bound" or "constrained" property.

- PropertyChangeSupport supports bound properties.
- PropertyDescriptor describes a property that has accessor methods.
- PropertyEditorManager is used to locate a property editor for a given type.
- SimpleBeanInfo aids in providing BeanInfo classes.
- VetoableChangeSupport supports constrained properties.

The Math Package

- BigDecimal provides immutable, arbitrary-precision signed decimal numbers, operations for basic arithmetic, scale manipulation, comparison, format conversion and hashing.
- BigInteger provides immutable, arbitrary-precision integers, and operations for modular arithmetic, GCD calculation, primality testing, prime generation, and single-bit manipulation.

The Networking Package

- ContentHandler is the superclass of all classes that read objects from a URLConnection.
- DatagramPacket is used to implement a connectionless packet delivery service. DatagramSocket represents a socket for sending and receiving datagram packets.
- DatagramSocketImpl is the base class for datagram and multicast sockets.
- HttpURLConnection is a URLConnection with support for HTTP-specific features. InetAddress represents an Internet Protocol (IP) address.
- MulticastSocket sends and receives IP multicast packets.
- ServerSocket waits for requests to come in over the network, and performs some operation based on that request.

- Socket is an endpoint for communication between two machines.
- SocketImpl a common superclass of all classes that actually implement sockets.
- URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web. A resource can be anything from a file or a directory to a database or a search engine.
- URLConnection is the superclass of all classes that represent a communications link between the application and a URL.
- URLEncoder provides a utility for converting a String into a MIME format called "x-www-form-urlencoded" format.
- URLStreamHandler is the common superclass for all stream protocol handlers.

The AWT Package

- AWTEvent is the root class for all AWT events.
- AWTEventMulticaster implements efficient and thread-safe multi-cast event dispatching for the AWT events.
- BorderLayout lays out a container, arranging and resizing its components to fit in five regions: North, South, East, West, and Center.
- Button creates a labeled button.
- Canvas represents a blank rectangular area of the screen onto which the application can draw.
- CardLayout is a layout manager for a container that treats components as a stack of cards, and allows only one card to be visible at a time.
- Checkbox a graphical component that can be in either an "on" (true) or "off" (false) state.
- CheckboxGroup is used to group together a set of Checkbox buttons.
- CheckboxMenuItem a check box that can be included in a menu.
- Choice presents a pop-up menu of choices.

- Color encapsulates colors using the RGB format.
- Component is an object having a graphical representation and is displayed on the screen to interact with the user.
- Container is a component that can contain other AWT components.
- Cursor represents the mouse cursor.
- Dialog is a dialog that takes input from the user.
- Dimension encapsulates the width and height of a component in a single object.
- Event encapsulates events from the platform's Graphical User Interface in the Java 1.0 event model.
- EventQueue a class that queues events.
- FileDialog displays a dialog window from which the user can select a file.
- FlowLayout arranges components in a left-to-right flow like lines of text in a paragraph.
- Font produces font objects.
- FontMetrics gives information about the rendering of a font.
- Frame is a top-level window with a title and a border.
- Graphics is the abstract base class for all graphics contexts.
- GridBagConstraints specifies constraints for components that are laid out using the GridBagLayout class.
- GridBagLayout is a layout manager that aligns components of different size vertically and horizontally.
- GridLayout is a layout manager that lays out a container's components in a rectangular grid.
- Image is the superclass of all classes that represent graphical images.
- Insets specifies the space that a container must leave at each of its edges.
- Label is a component for placing text in a container.
- List is a scrolling list of text items.
- MediaTracker is a utility class to track the status of a media objects.

- Menu is a pull-down menu component.
- MenuBar encapsulates the platform's concept of a menu bar bound to a frame.
- MenuComponent is the super class of all menu-related components.
- MenuItem an item in a menu
- MenuShortcut is a keyboard accelerator for a MenuItem.
- Panel provides space in which an application can attach any other component.
- Point represents a location in a two-dimensional (x, y) coordinate space.
- Polygon encapsulates a description of a closed, two-dimensional.
- PopupMenu is a menu which can be dynamically popped up at a specified position.
- PrintJob initiates and executes a print job.
- Rectangle defines a rectangle.
- ScrollPane provides automatic horizontal and/or vertical scrolling for a component.
- Scrollbar provides a scroll bar.
- SystemColor represents the color of GUI objects on a system.
- TextArea is a multi-line region that displays text.
- TextComponent is the superclass of any component that allows the editing of some text.
- TextField is a text component that allows for the editing of a single line of text.
- Toolkit is the abstract superclass of classes in the Abstract Window Toolkit.
- Window is a top-level window with no borders and no menubar.

The Text Package

- BreakIterator implements methods for finding the location of boundaries in text.
- ChoiceFormat attaches a format to a range of numbers.

- CollationElementIterator is used as an iterator to walk through each character of an international string.
- CollationKey represents a String under the rules of a specific Collator object.
- Collator performs locale-sensitive String comparison.
- DateFormat is an abstract class for date/time formatting subclasses.
- DateFormatSymbols is a public class for encapsulating localizable date-time data.
- DecimalFormat is a subclass of NumberFormat.
- DecimalFormatSymbols represents the set of symbols needed by DecimalFormat to format numbers.
- FieldPosition is a subclasses to identify fields in formatted output.
- Format is an abstract base class for formatting locale-sensitive dates, messages, and numbers.
- MessageFormat provides a means to produce concatenated messages in language-neutral way.
- NumberFormat is the abstract base class for all number formats.
- ParsePosition is used to keep track of the current position during parsing.
- RuleBasedCollator is a subclass of Collator that provides a data-driven table collator.
- SimpleDateFormat is a class for formatting and parsing dates in a locale-sensitive manner.
- StringCharacterIterator implements the CharacterIterater protocol for a String.

The Security Package

- DigestInputStream is a transparent stream that updates the associated message digest using the bits going through the stream.

- DigestOutputStream is a transparent stream that updates the associated message digest using the bits going through the stream.
- Identity represents identities that can be authenticated using keys.
- IdentityScope represents a scope for identities.
- KeyPair is a simple holder for a key pair (a public key and a private key).
- KeyPairGenerator is used to generate pairs of public and private keys.
- MessageDigest provides the functionality of a message digest algorithm, such as MD5 or SHA. Message digests are secure one-way hash functions that take arbitrary-sized data and output a fixed-length hash value.
- Provider is a "provider" for the Java Security API.
- SecureRandom provides a crytpographically strong pseudo-random number generator based on the SHA-1 hash algorithm.
- Security centralizes all security properties and common security methods.
- Signature is used to provide the functionality of a digital signature algorithm, such as RSA with MD5 or DSA. Digital signatures are used for authentication and integrity assurance of digital data.
- Signer is used to represent an Identity that can also digitally sign data.

The RMI Package

- Naming is the bootstrap mechanism for obtaining references to remote
- objects based on Uniform Resource Locator (URL) syntax. The URL for a remote object is specified using the usual host, port and name: rmi://host:port/name
- host = host name of registry (defaults to current host)
- port = port number of registry (defaults to the registry port number)
- name = name for remote object
- RMISecurityManager defines a default security policy for RMI applications.

The RMI Registry Package

- LocateRegistry is used to obtain the bootstrap Registry on a particular host including a local host.

The RMI Server Package

- LogStream is a mechanism for logging errors.
- ObjID is used to identify remote objects uniquely in a VM.
- Operation holds a description of a Java method.
- RMIClassLoader provides static methods for loading classes over the network. RMISocketFactory is used by the RMI runtime in order to obtain client and server sockets for RMI calls.
- RemoteObject implements the java.lang.Object behavior for remote objects. RemoteServer is the common superclass to all server implementations.
- RemoteStub is the common superclass to all client stubs.
- UID is for creating identifiers that are unique.
- UnicastRemoteObject defines a non-replicated remote object whose references are valid only while the server process is alive.

The Reflection Package

- Array provides static methods to dynamically create and access Java arrays.
- Constructor provides information about a single constructor for a class.
- Field provides information about a single field of a class.
- Method provides information about a single method on a class.
- Modifier provides static methods and constants to decode class and member access modifiers.

The SQL Package

- Date allows JDBC to identify the date as an SQL DATE value
- DriverManager provides a basic service for managing a set of JDBC drivers.
- DriverPropertyInfo supplies property for connections.
- Time allows JDBC to identify the time as an SQL TIME value.
- Timestamp allows JDBC to identify the date as an SQL TIMESTAMP value.
- Types defines constants that are used to identify SQL types.

The Swing Packages

Overview

- Swing provides controls derived from Jcomponent.
- Swing components were designed to be combined and extended to create custom components.

The Swing Package

- AbstractAction provides default implementations for the JFC Action interface. AbstractButton defines the common behaviors for the JButton, JToggleButton, JCheckbox, and the JRadioButton classes.
- AbstractListModel defines the data model that provides a List with its contents.
- BorderFactory allows vending standard Border objects.
- Box is a lightweight container that uses a BoxLayout object as its layout manager.
- BoxLayout is a layout manager that places each of its managed components from left to right or from top to bottom.
- ButtonGroup is used to create a set of buttons where only one button will be allowed to be "on" at a time.
- ColorChooserPanel is the abstract superclass for color choosers.
- DebugGraphics is a subclass supporting graphics debugging.
- DefaultBoundedRangeModel is a generic implementation of BoundedRangeModel.
- DefaultButtonModel is the default implementation of a Button component's data model.
- DefaultCellEditor is the default editor for table and tree cells.

- DefaultDesktopManager is an implementaion of the DesktopManager.
- DefaultFocusManager implements the java.util.Vector API and notifies the JListDataModel listeners when changes occur.
- DefaultListSelectionModel is the default data model for list selections.
- DefaultSingleSelectionModel is a generic implementation of SingleSelectionModel.
- GrayFilter is an image filter that "disables" an image by turning it into a grayscale image.
- ImageIcon is an implementation of the Icon interface that paints Icons from Images.
- Japplet is an extended version of java.applet.Applet.
- Jbutton is an implementation of a "push" button.
- JcheckBox is an implementation of a CheckBox.
- JcheckBoxMenuItem is a menu item that can be selected or deselected.
- JcomboBox is an implementation of a ComboBox.
- Jcomponent is the base class for the Swing components.
- JdesktopPane keeps a reference to a DesktopManager object.
- Jdialog is the main class for creating a dialog window.
- Jframe is an extended version of java.awt.Frame.
- JinternalFrame is a lightweight object that provides many of the features of a native frame.
- Jlabel is a display area for a short text string.
- Jlist allows the user to select one or more objects from a list.
- Jmenu is an implementation of a menu.
- JMenuBar is an implementation of a MenuBar.
- JMenuItem is an implementation of a MenuItem.
- JoptionPane supports a pop up a dialog box that prompts users for a value or informs them of something.

- Jpanel is a panel.
- JpasswordField allows the editing of a single line of text where the view indicates something was typed, but does not show the original characters.
- JpopupMenu is a Popup Menu.
- JprogressBar displays an integer value graphically within a bounded interval.
- JradioButton is an implementation of a radio button
- JradioButtonMenuItem is a menu item that is part of a group of menu items in which only one item in the group can be selected.
- JscrollBar is a scrollbar.
- JscrollPane is a scrollable view.
- Jseparator is a menu item divider.
- Jslider is a component that lets the user graphically select a value by slding a knob within a bounded interval.
- JsplitPane is used to divide two components.
- JTabbedPane lets the user switch between a group of components by clicking on a tab with a given title and/or icon.
- JTextArea is a multi-line area that displays plain text.
- JTextField allows the editing of a single line of text.
- JtoggleButton is a two-state button..
- JToolBar provides a component for displaying commonly used Actions or controls. JtoolTip displays a "Tip" for a Component.
- JTree displays a set of hierarchical data as an outline.
- Jwindow a window implementation.
- KeyStroke represents a key being typed on the keyboard.
- LookAndFeel characterizes a look and feel from the point of view of the pluggable look and feel components.
- OverlayLayout is a layout manager to arrange components over the top of each other.

- ProgressMonitor monitors the progress of some operation.
- SizeRequirements calculates information about the size and position of components.

The JBCL Packages

Overview

- The JavaBeans Component Library (JBCL) is a full-featured API for Java application development.
- The JBCL is built on the com.sun.java.Swing base architecture.
- The borland packages and their groups of classes are listed below:
  - ✓ The datastore package provides for persistent row storage and caching for JBuilder DataSet objects, Java Objects, and arbitrary files.
  - ✓ The dbSwing package contains data-aware extensions to components in the com.sun.java.swing package and includes controls, comboBoxes, text fields and model classes.
  - ✓ The control package contains UI components such as controls, dialogs, and containers, including data-aware, model-view components.
  - ✓ The dataset package contains components for general data connectivity. It provides support for data-aware controls and visual development of data applications.
  - ✓ The io package contains specialized stream classes for input and output.
  - ✓ The layout package contains layout manager and constraint classes, such as XYLayout for rapid prototyping.
  - ✓ The model package contains model interfaces and classes to store data items for composite components, for easy updating of the items shown in UI controls.
  - ✓ The util package contains utility classes and interfaces, such as interfaces containing groups of constants.

- ✓ The view package contains views, item painters, and item editors. Used with classes in the borland.jbcl.control package to form composite components.
- ✓ The borland.sql.dataset package contains components for JDBC-specific database connectivity.

The JGL Packages

Overview

- The JDK has been designed to provide the minimal subset of features used by the majority of Java developers.
- If you are embarking on any serious Java development, you will need a professional package of collections and algorithms.
- JGL was designed to provide you with collections and data processing algorithms for use with those collections.
- JGL includes eight Java packages, each carefully designed to include a set of related functionalies.
- The following is a list of the eight Java packages and their contents.
  - ✓ com.objectspace.jgl contains the interfaces, collection classes, and iterators.
  - ✓ com.objectspace.jgl.adapters contains the Java native array adapters.
  - ✓ com.objectspace.jgl.algorithms contains all the JGL algorithms.
  - ✓ com.objectspace.jgl.functions contains function objects that can be used to modify algorithms.
  - ✓ com.objectspace.jgl.predicates contains predicate classes used to specify element ordering within containers.
  - ✓ com.objectspace.jgl.util contains a few miscellaneous utility classes.
  - ✓ com.objectspace.jgl.voyager contains a version of the collection classes that can be used in a distributed fashion with ObjectSpace Voyager.
  - ✓ com.objectspace.jgl.voyager.algorithms contains distributed algorithms for use with ObjectSpace Voyager.
- JGL comes complete with the following:
  - ✓ A single jar file containing the JGL classes. Individual jar files for each Java package are also available for download.

- ✓ Fully commented source code
- ✓ Complete HTML API javadoc documentation
- ✓ A comprehensive HTML user guide packed with over 100 examples
- ✓ Online versions of every example in the user documentation
- ✓ A suite of performance benchmarks
- ✓ FREE support

KL Group Packages

Overview

- The KL Group provides new or improved components that can be used
- in Java programs along with, or instead of, their JDK equivalents.

The BWT Package

- BWT provides new or improved GUI components that can be used
- in Java programs along with, or instead of, their AWT equivalents.
- JClass BWT provides new or improved versions of various AWT components and
- containers. JClass
- BWT provides the following features:
  - ✓ Written entirely in Java;
  - ✓ Components and containers that are easier to use and offer more features than their AWT equivalents;
  - ✓ A uniform MS-Windows 95 "look-and-feel" across all platforms;
  - ✓ Consistent operation across platforms;
  - ✓ Ability to create Applets incorporating JClass BWT components whose parameters can be set from either HTML or comma-delimited files;
  - ✓ GUI components are double-buffered, providing faster on-screen redrawing;
  - ✓ Callbacks and extra methods reduces the need to subclass a component to obtain extra functionality, saving time and reducing the complexity of Java programs;
  - ✓ BWT components are easily subclassed;
  - ✓ Provision for callbacks and event tracking enables Java programs to easily notify and interact with BWT components;

- ✓ Better keyboard traversal and focus management.

The Chart Package

- ➢ JClass Chart is a charting/graphing component.
- ➢ Chart has been designed from the ground up to take advantage of object-oriented
- ➢ techniques, but not at the expense of usability or efficiency.
- ➢ JClass Chart offers users a choice of interaction techniques:
  - ✓ as a component user by setting JClass Chart properties programmatically.
  - ✓ as an OO developer by using and extending JClass Chart objects.
  - ✓ as a Bean developer/user by setting JClass Chart properties using a third-party Integrated Development Environment.
  - ✓ as a Web page developer by setting JClass Chart properties exposed through HTML parameters.
- ➢ Shown below are just some of the things that can be done with JClass Chart:
  - ✓ multiple data series
  - ✓ multiple chart types on the same chart
  - ✓ multiple data sets plotted against different axes
  - ✓ multiple x and y axes
  - ✓ three-dimensional appearance for bar, stacking bar and pie charts
  - ✓ rich text (JCString) for headers, footers, axis titles and even axis annotations.
  - ✓ Developers can mix text font, color, placement with images and URLs.
  - ✓ chart interactions including rotation, zooming, scaling, translation and point editing
  - ✓ unmatched axis labelling and layout options

- ✓ flexible external data model, with common data sources already implemented (files, URLs, input streams).

- The chart types supported by this release of JClass Chart are Plot, Scatter Plot,
- Area, Bar, Stacking Bar, Pie, Hi-Lo, Hi-Lo-Open-Close and Candle.
- JClass Chart supports external data sources.

The LiveTable Package

- JClass LiveTable is a Java component that displays rows and columns of user-interactive text, images, hypertext links, and other Java components in a scrollable window.
- You can set the properties of JClass LiveTable components to determine how the table will look and behave. You can control:
  - ✓ the number of rows and columns in the table
  - ✓ labels for columns and rows
  - ✓ colors, fonts, borders, alignment, and spacing for cells and labels
  - ✓ sorting columns and multiple columns
  - ✓ adding, deleting, moving, and dragging rows and columns
  - ✓ scrolling and attaching default or custom scrollbars
  - ✓ cell selection and traversal
  - ✓ cell editing
  - ✓ attaching custom and stock data sources