What's Perking? An Introduction to Java

John Jay King
King Training Resources
6341 South Williams Street
Littleton, CO 80121-2627 USA
www.kingtraining.com
800.252.0652 or 303.798.5727

Objectives

- Upon completion of this session, students will be able to:
  - Discuss Java portability, strengths, and weaknesses.
  - Understand the difference between Java applications and Java applets.
  - Learn about Sun’s SDK and related documentation
  - Be familiar with Java capabilities supported by the Oracle database and its tools.
  - Understand Java class and inheritance concepts
  - Know the differences in Java GUI applications using AWT or Swing.
  - Understand Oracle-JDBC interface basics.
  - Become aware of available third-party Java tools.

Java Overview

- Java is a programming language created and owned by Sun Microsystems
- Java syntax was largely based upon C++ syntax
- The computing industry has adopted Java quickly, no ANSI or ISO standard exists
- Java promises that code can be "write once, run anywhere" and generally delivers on the promise
- Java program compilation creates a partially-compiled product called "byte code" that is then interpreted at execution
- On some platforms, Just In Time (JIT) compilers are completing the compilation of Java code in the local environment

Java Implementations

- Created by Sun Microsystems engineers led by James Gosling and Patrick Naughton
- James Gosling named the language "Oak" after a favorite tree visible from his office - it was renamed due to trademark issues
- The first breakthrough event was when Java was used to create the browser now known as HotJava entirely in Java (1995)
- Netscape 2.0 is Java-enabled (1996)
- Java 1.02 was released later in 1996
- Java 1.1 (1997) widely used
- Java 1.2 (also known as Java 2), released in late 1998, includes “Swing” toolkit and the Java 2 Enterprise Edition
- Java SDK 1.3 released in mid-2000

Write Once, Run Anywhere

- Compiling to machine-specific code is the best way to maximizes performance, but, cannot be shared
- The Java compiler creates an object file called "bytecode" that is portable to any platform that supports a Java Virtual Machine (JVM)
- Java code can be written without regard to runtime
- Native datatypes and methods are the same everywhere
- Interpretive approach is portable, not high-performance
- Just In Time (JIT) compilers create optimized machine-specific code on demand
- Java changes so rapidly that true independence is limited
- Latest versions of Netscape Navigator and Microsoft Internet Explorer support Java as does Sun's Hot Java

Java Virtual Machine (JVM)

- Bytecode is generic and transportable to any Java environment that supports a Java Virtual Machine (JVM)
- A JVM specific to the environment compiles the bytecode and generates machine-specific code
- JVMs available for every major execution environment
- Java enabled web-browsers include a JVM
- A JVM may also be installed on a computer using software included in Sun's Software Developer's Kit (SDK) or Java Developer's Toolkit (JDK)
- JVM supports the runtime environment including the construction of class objects, garbage collection, optimization, space management
- Specification for the JVM at http://java.sun.com

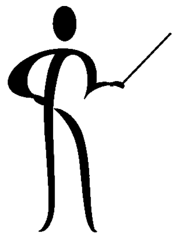
Security

- Java prevents programmers from causing problems in computers other than where the code is executing:
  - No direct manipulation of memory or pointers
  - All code is interpreted by the JVM
  - Java security manager checks all actions against the Java run-time library for alterations
  - Applets execute within a security "Sandbox" ensuring that local code is trusted to have full to system resources while downloaded code is not trusted (Java 1.1 and Java 2 provide mechanisms for “trusting” an Applet)

Java "Sandbox"

- Applet security in particular must exist within a "Sandbox" here are the original rules:
  - Applets cannot execute local programs
  - Applets may not read or write files on local computer
  - Applets have access only to Java-specific information from the local computer, specifically, they cannot access user name, email address, or other personal data of the computer user's
  - New windows "popped-up" by applets display warning
  - Applets cannot communicate with any server other than the one where the applet was downloaded from
  - Applets may be "signed" for added security

SDK/JDK



- Sun's Software Developer's Kit (SDK) was formerly known as the Java Developer's Toolkit (JDK)
- The current release is called the J2 SDK (Java 2 Software Developer's Kit) it includes:
  - Java 2 SDK (Java 1.3)
  - Tools including Java Workshop and Appletviewer
  - Java Runtime Environment
  - Application Programming Interfaces (APIs)
  - Earlier Java Development Kit versions
  - The JDK may be downloaded from the Sun Microsystems website: http://java.sun.com

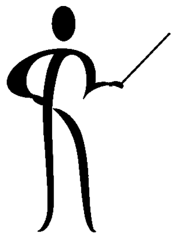
Third-Party Tools

- Java development tools from third-party vendors include:
  - IBM Visual Age for Java
  - Symantec Visual Café
  - Oracle JDeveloper
  - Inprise JBuilder
  - Microsoft J++

Web Interfaces

- Java is designed to be downloaded via the Internet
- Only Java bytecode is downloaded, the source code stays wherever the bytecode was compiled
- "Applets" are Java programs designed specifically to be run inside of web pages
- To run an applet the browser must be Java-enabled, unfortunately, support is haphazard:
  - Netscape 2 and Netscape 3 are Java 1.02-enabled
  - Internet Explorer 3, Internet Explorer 4, and Netscape 4 are Java 1.1-enabled, but, support slightly different subsets of Java (later versions support most Java)
  - Sun supports Java Plug-In modules that support the latest Java features in Internet Explorer and Netscape

Java Components



- Programs
- Applications
- Applets
- Classes
- Methods
- Properties

Welcome.java Application

```
/**
 * Java application version of "Hello World" program
 *
 * @version 1.01 04/01/2000
 * @author  John Jay King
 */
public class Welcome
{
  public static void main(String[] args)
  {
    System.out.println("Hello SUPER STUDENT, welcome to Java!");
  } // end of main method
} // end of class Welcome
```

Welcome.java Application

/** ... */	Comments
class Welcome	Class name
{	Begin class
public static void main (String[] args)	Method head
{	Begin main
System.out.println("Hello SUPER STUDENT, welcome to Java!!");	Print text
}	End main
// end main method	Comments
}	End class
// end class Welcome	Comments

The class name MUST match the file name

Applications

- Applications (like Welcome.java) are stand-alone Java programs that execute without a browser
- Java applications are not subject to applet security
- The HotJava browser is an application
- Protocol handling sometimes uses Java applications
- Applications are sometimes called Console Applications
- A method named "main" is required; stand-alone applications must have a starting point
- "main" must be public and static, it must return void, and always expects an array of String arguments:

```
public static void main ( String args[] )
```

Applets

- Applets are attached to web pages that run in browsers
- Applets require a Java-enabled browser in order to run
- Applets do not require (or use) a "main" method
- Java applets must conform to a rigid security model that forbids or limits access to system resources
- Only the latest browsers support Java 1.2 features
- Java-enabled browsers recognize a special HTML tag

```
<applet code=WelcomeAppl width=600height=200></applet>
```

```
– applet HTML tag
– code   Name of applet class file
– width/height Width and Height of applet
```

WelcomeAppl.java

```
/**
 * Java applet version of "Hello World" program
 *
 * @version 1.01 04/01/2000
 * @author  John Jay King
 */
 public class WelcomeAppl extends java.applet.Applet {
    public void paint( java.awt.Graphics gc ) {
        gc.drawString("Hello SUPER STUDENT, welcome to Java
  Applets!", 100, 90 );
    } // end paint method
} // end WelcomeAppl class
```

WelcomeAppl.java

- Note there is not a "main" method in this program
- This applet extends (inherits from) a predefined class named Applet (more on inheritance later)
- This applet uses a "drawString" method from a predefined class named Graphics to display the text message (Graphics is part of the jawa.awt package)

WelcomeAppl.html

```
<html>
<head>
  <title>Sample HTML Page for Applet
  Execution</title>
</head>
<body>
    <applet code=WelcomeAppl width=600
  height=200></applet>
</body>
</html>
```

- Be sure that your class name is typed properly
- Test the html using your browser or Appletviewer

HTML Connectivity

```
<html>
<head>
  <title>Sample HTML Page for Applet with Parameters</title>
</head>
<body>
  <applet code="SpinCup.class" width="100" height="100">
      <param name="link” value="http://www.kingtraining.com/">
      <param name="image"     value="coffee.gif">
      <param name="bgcolor"   value="ffffff">
      <param name="delay"     value="100">
      <param name="number"    value="4">
  </applet>
</body>
</html>
```

SpinCup.java

```
/** Modification of SpinCup.java sample applet shippedSDK/JDK
...
public class SpinCup extends Applet implements Runnable
{
...
  private Sprite sprite;
...
  String value = getParameter("bgcolor");
...
  value = getParameter("number");
...
  value = getParameter("delay");
...
  value = getParameter("loop");
...
  value = getParameter("link");
...
  value = getParameter("image");
...
         sprite = new Sprite(number, size, loop);
...
```

JAR Files

- JAR stands for Java Archive, it is an archive of files needed by a Java applet
- Compressed using the ZIP file format
- Referenced in HTML using the ARCHIVE attibute

```
<APPLET CODE="Animator.class"
    ARCHIVE="animation.jar"
    WIDTH=500
    HEIGHT=200>
</APPLET>
```

- JAR files contain a manifest created when the JAR file is created automatically, or, if specified directly

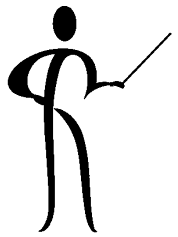
Signed Applets

- Today, Applet signatures are somewhat implementation dependent
- JDK 1.1 signatures require browser-specific tools when using Netscape Navigator or Microsoft Internet Explorer
- Both Internet and Intranet applets might be signed
- A Java browser Plug-in evaluates the
- The jarsigner utility is used to sign a file after the jar utility is used to create the jar

Servlets and JSPs

- Recent improvements to Java and web servers allow the creation of server-side Java components, this allows the server to do the "heavy lifting" of database I-O
- Java Server Pages (JSPs) are a simplified mechanism for creating/using servlets that will provide user information to via HTML and XML to the browser
- Using JSPs and servlets removes the need to download large Java applets and avoids many security issues since the code will actually be running on the web server

Encapsulation



- Encapsulation: the process of hiding the internal workings of a class to support or enforce abstraction
- This requires a distinction between a class's interface, which is public, and its implementation which is private
  - Interface: describes what a class can do
  - Implementation: how a class performs operations
- Encapsulation exposes only relevant properties; the user views an object in terms of the operations it can perform, not in terms of the internal data structure
- A truly encapsulated class surrounds or "hides" its data with; access is allowed only by calling class methods
- Ideally, a class has no knowledge of the data structures used by other classes, referring to them only through their interfaces (methods)

Inheritance

- OOP provides the ability to define a hierarchy of types
- In Java, you define one class as a subclass, or special category, of another class (the superclass) by extending the superclass (derivation)
- You can also define subcategories of a single broad category by deriving them all from one superclass (a form of abstraction)
- A superclass specifies what derived classes have in common, so you can concentrate on those shared traits
- Superclass is generalization; Subclass is specialization
- There are two main advantages to class hierarchies:
  – The subclass can share the superclass's code
  – The subclass can share the superclass's Interface

Inheriting Code

- To incorporate the functionality of a class into a new one, simply extend the new class from the existing one
- If you're implementing several classes at once that share features, a class hierarchy can reduce redundant coding
  - Describe and implement them once in the superclass, rather than repeatedly in each subclass
  - A class hierarchy designed for code sharing has most of its code in the superclasses; enabling code reuse
  - The extended (derived) classes represent specialized or extended versions of the superclass

Class

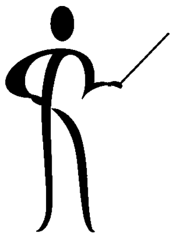
- A Java class represents a collection of Properties (called Variables) that describe an object and the Actions (called Methods) that may be performed upon that object
- Classes usually correspond to some object in the real world: Invoice, Customer, Person, Student, P.O.
- A class includes:
  – Definitions of all variables (properties)
  – Methods (actions) that control activity with variables
- A Class is like a blueprint, a model from which one or many objects (instances) may be created (instantiated)

– Cookie Cutter	Class
– Cookie	Object/Instance/Instantiation

Properties and Actions

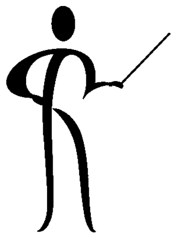
- Class Properties (variables) describe the data structure of the class, for instance, a Person class might include: LastName, FirstName, PhoneNumber, EmailAddress, and more...
- Class Actions (methods) manipulate the class and its properties, for instance, a Person class might include:
  - SetLastName
  - GetLastName
  - SetFirstName
  - GetFirstName
  - and more...

Field/Variable



- Class properties are most correctly called "instance variables" describing properties of an actual occurrence (instance) of the class (usually called an object)
- Variables must be defined before they are used in Java
- Variables may be defined using any of Java's built-in datatypes
- Variables may be defined using any existing class, either one built-in to Java (like String), or one created by your own team
- It is generally considered good practice to allow modification and access of variables only through the methods defined as part of the class

Method



- Methods are the program code used to: intialize variables, get and set variable values, describe actions that a class may take and the actions that might involve properties of the class
- Methods and Variabless might be public, private, or protected

– public	Available to entire world (used for most methods)
– private	Available only inside this class
– protected	Available inside the class and inside subclasses that extend the class (used for most variables)

Method, 2

- If a method returns no values, it is called on a line by itself:

```
public void calcFootage(float width, float length) { ... }
```

```
calcFootage(inWidth,inLength);
```

- If a method returns a value, it must be part of another statement:

```
public float calcFootage(float width, float length) { ... }
```

```
myVar = calcFootage(inWidth,inLength);
```

or

```
if (calcFootage(inWidth,inLength) > 100)
```

Sample Class: Room.java

```
/** Sample Room Class */
public class Room {
  protected double length;
  protected double width;
  protected double footage;
  public void calcFootage() {
   footage = width * length;
   }
  public void displayRoom() {
       System.out.println("Room width  = " + width);
       System.out.println("Room length = " + length);
       calcFootage();
       System.out.println("Square footage   = " + footage);
   }
  public void setWidth(double widthArg) {
    width = widthArg;
   }
  public void setLength(double LengthArg) {
    length = LengthArg;
   }
}
```

Sample Class: RoomDemo.java

```
/**
 * Sample Use of Room Class
 */
public class RoomDemo {
  public static void main(String[] args) {

  Room myRoom;
  myRoom = new Room();

  myRoom.setLength(10);
  myRoom.setWidth(12);
  myRoom.displayRoom();
}
```

Though it is valid for RoomDemo to use Room variables directly, manipulations **should** use class methods

Fully-qualified Names & Packages

- In the RoomDemo class, the Room object myRoom was used to qualify the names of methods used in the Room class
- Normally common routines share a directory; classes in a directory/subdirectory are often referred to as a package
- The Java 2 SDK API includes many packages such as: java.applet, java.awt, java.beans, java.io, java.lang, java.math, java.net, java.security, java.sql, java.text, java.util, javax.swing
- Package java.lang includes class System that has a variable named out; out is an object of the PrintStream class and has a method named println
- Full name of method is: **java.lang.System.out.println**
- But, we did not have to fully qualify the name... Why?

import

- Java uses a naming shortcut called import to abbreviate references to packaged classes
- java.lang is automatically imported by all programs
- However, to use the class named DataInputStream that is part of the java.io package there are three choices:
  - Fully qualify each time java.io.DataInputStream;
  - Import class specifically and use without qualification import java.io.DataInputStream;
  - Import all classes in the package using a wildcard so that none of them require qualification (most common) import java.io.*;
- Full qualification necessary is for duplicate class names

this

- Sometimes it would be less confusing (or just more convenient) to use the same name for width everywhere it occurred?
- Java offers the "this" qualifier allowing specific reference to class variables to differentiate them from a local variable with the same name

```
public void setWidth(double width) {
        this.width = width;
   }
```

- Using **this** allows the use of an otherwise duplicate name by explicitly referencing the class member
- C++ offers a similar construct, but, in Java this is merely a naming aid

Constructor Methods, 1

- Upon instantiation, a class executes a method defined within it that has the same name as the class

```
public class Room {
   protected double length;
   protected double width;
   protected double footage;
   protected String rname;
  public Room () {       // constructor method
    length = 12;      // default room length
    width  = 12;      // default room width
    rname   = "Empty"; // default room name
    calcFootage();
   }
 // *** Rest of class definition goes here ***
}
```

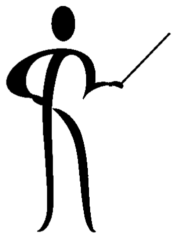
Constructor Methods, 2

- The Room method is the constructor method since it has the same name as the class to which it belongs (this is a requirement)
- Constructor methods may not return a value
- Constructor methods must be public
- This constructor assumes that a Room object will be instantiated without any argument values being passed

```
myRoom = new Room();
```

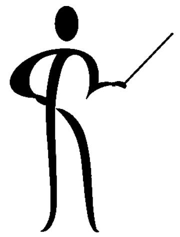
  – In the statement above, the myRoom object is instantiated and automatically calls the constructor
  – The constructor usually (as above) sets initial values for class variables

Overloading Method



- Overloading means having two methods with the same names and different signatures
- Java automatically chooses the correct method by matching the argument list used to the method signatures
- If no signature matches, an error will result
- Overloaded methods are frequently used when:
  - Some methods have generic titles and might be used for a variety of data types
  - Sometimes an instance is created using an argument list, and sometimes the arguments are not available
- This illustrates the object-oriented concept of polymorphism (object decides what method to call, based upon values in signature)

Overloading a Constructor



```
public class Room {
  public Room (double width, double length) {
   this.length = length;   // default room length
   this.width  = width;    // default room width
   calcFootage();                 }
  public Room () {
   length = 12;      // default room length
   width  = 12;      // default room width
   calcFootage();                 }// ** rest of class **
```

- Code below defines two Room variables, instantiated differently:

```
Room myRoom;
myRoom = new Room();
Room otherRoom;
otherRoom = new Room(8.1f,18.5f);
```

  – myRoom   Constructor with no arguments
  – otherRoom   Constructor with two double arguments

Get/Set (Accessor/Mutator) Methods

- Well-behaved classes allow access to their variables only through methods that are part of the class:
  - Make all variables private
  - Define methods to Get (Access) variable values from an object
  - Define methods to Set/Put (Mutate) variable values in an object

```
public class Room {
  protected  double length;
// ** more code here **
  public double getWidth() {
    return width;
   }
  public void setWidth(double width) {
    this.width = width;
   }
// ** rest of class here **
```

Comments

- Use comments liberally to explain what you are doing, after all, the next person to see the code might be you!
- Comments may be coded anywhere a blank is valid
- Multi-line comments begin with a slash followed by an asterisk (/*) and end with an asterisk followed by a slash (*/)
- /* */ Comments may not be nested within one another
- Multi-line comments that begin /** are used by the javadoc utility as part the documentation javadoc generates for a class
- Single-line comments use two slashes (//), they can be nested within /* */ comments

Strongly-typed Data

- Java is a strongly-typed language
- Strongly-typed means that all variables must have a declared type, there is nothing similar to the "variant" data type found in VB and some other languages
- Java has eight simple (or primitive) types of data:
  - Six are numeric data types
  - Two are character data types
- Java allows complex data to be defined using classes, these are also called "reference" data types
- Perhaps the most common reference data type is the String class used to hold character string data

Simple Datatypes

- The simple (primitive) data types in Java are:

– Integer	byte, short, int, long
– Floating-point	float, double
– Character	char
– Boolean	boolean

- Java simple data types are the same size across all platforms, so, there is no reason to have code that changed depending upon the local size limitations
- Numeric values in Java are always signed

Unicode

- Java character data uses the internationally standard 16-bit Unicode encoding
- Traditional European languages use a limited single-byte character set that fits quite nicely into the ASCII/ISCII
- Many world languages (Japanese for instance) use a more complex set of characters and a single byte (256 possible values) is simply is not large enough to encode the variety of characters needed
- The Unicode Worldwide Character Standard is designed to accommodate all of the world's primary languages by supporting a multi-byte encoding scheme that contains 38,885 distinct characters from 25 languages
- For complete information about the Unicode standard contact the Unicode Consortium (http://unicode.org)

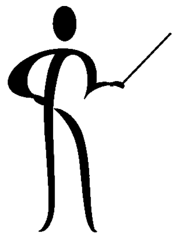
Reference Datatypes

- Objects and arrays provide complex (non-primitive) data types in Java
- Primitive datatypes are often called "non-reference" types since data is passed "by value" to methods
- Objects and arrays are often called "reference types" because they are handled "by reference"
- Passing arguments behaves differently for primitive arguments than for arguments that are objects or arrays

Objects

- Non-reference datatypes include objects and arrays
- Objects are also called "abstract data types" (ADTs) or "user-defined data types"
- Classes are used to create objects (instances of class)
- One of the greatest strengths of Java is the set of built-in classes included in the Java API, to see a list: 1) Look in the Java SDK Documentation, 2) Look under API & Language, 3) Select Java 2 Platform API Specification, 4) Look at the list of "All Classes"
- Graphical User Interface (GUI) programming would be difficult if all code needed to program windowed applications had to be done over and over, fortunately class libraries have evolved to perform these tasks

new Object

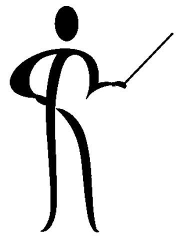


- New objects are created with the "new" keyword

```
Button helloButton;                          // define button
helloButton  = new Button("Hello");  //  instantiate button
```

  - The first line defines a variable of the Button class, however, the button is not useable until instantiation
  - The second line creates an object (sets aside space) of the Button class, allocates memory, and assigns the address to the variable ("instantiation”)
- While Java allocates memory for objects dynamically, programmers are not responsible for tracking the memory
- Unlike some other languages, programmers do not have the option of manipulating the address of variables (this is a safety feature)

Garbage Collection



- In most languages, programmers who allocate memory dynamically are responsible for tracking and releasing memore when no longer used
- Failure to properly release memory causes the "memory leaks" that so often plague certain environments
- Java is specifically designed to avoid memory leaks by cleaning up after itself, this mechanism is called "Garbage Collection" (GC):
  - The Garbage Collector periodically "wakes up" and begins looking for unused memory to free
  - Objects are considered for Garbage Collection when they are no longer referenced by any variables, object variables, nor arrays
  - Java's Garbage Collection may cause a brief performance slow-down when executing but the advantages outweigh this

Strings

- Strings are a most frequently used Java built-in class
- Quoted strings are instances of the String class

```
String abc;                  // uninitialized string
String def = "Cookie";       // string initialized
String ghi = "";             // empty string
abc = "Monster";             // assign abc
```

- Like all classes, String has many methods:
  - Use equals to compare the value of one string to another
  - Use equalsIgnoreCase to compare the value of one string to another without regard to case
  - Use length to determine the length of a string
- Strings are not delimited as they are in C/C++

Assignment Basics

- Java assignment operator places values into variables
- Assignment uses the equal (=) sign and places a copy of the value to the right of the equal sign into the variable to the left of the equal sign:

```
abc = 1;       // assigns the value of 1 to the variable abc
def = abc;     // assigns the contents of abc to def
firstName = "Moon Unit";   // assigns "Moon Unit" variable
```

- Assignment statements must be consistent with datatype
- The assignment operator may be used in declaration

```
int xyz = 1;           // Sets value to 1
char letterA = 'a'; // Assigns letter a to char variable
String lastName = "Zappa";
```

Arithmetic Operators

- Java provides five mathematical operators:

+	Addition	a = b + c;
-	Subtraction	a = b - c;
*	Multiplication	a = b * c;
/	Division	a = b / c;
%	Remainder	a = b % c; (the result of a = 10 % 3; is 1)

- Calculations are performed with a specific set of rules:
  - Things inside parentheses happen first
  - Multiplication, Division, and Remainder take precedence over Addition and Subtraction
  - When equal precedence, work proceeds from left-to-right

Automatic Increment/Decrement

ctrVar = ctrVar + 1; // familiar to most programmers

– add one to ctrVar and places the new value into ctrVar

- Java has special operators specifically for this activity:

++ctrVar	Increments ctrVar by one before using it in the current statement
ctrVar++	Increments ctrVar by one after using it in the current statement
--ctrVar	Decrements ctrVar by one before using it in the current statement
ctrVar--	Decrements ctrVar by one after using it in the current statement

– These operators have **side effects**, the variable being incremented or decremented is changed even though it might not normally be the intended result of the operation

Assignment Shorthand

- Java provides other ways to reduce coding
- If a statement sets an assignment's lvalue to the result of a simple calculation, another shorthand syntax is used
- For instance:

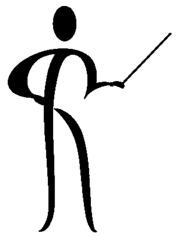
```
varabc += 10;       /* varabc = varabc + 10;     */

vardef -= varghi;   /* vardef = vardef - varghi; */

varjkl *= 0.5;      /* varjkl = varjkl * 0.5;    */

varmno /= 2;        /* varmno = varmno / 2;      */
```

Relational Operators



- Only mathematical symbols are used (no mnemonics)
- Relational operators return a boolean true when the comparison is True, and false when the comparison is False (true and false are reserved words)
- Relational operators are processed AFTER arithmetic

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal
!=	Not equal
!	Unary NOT (used for negation)
||	Logical OR
&&	Logical AND

if Statement, 1

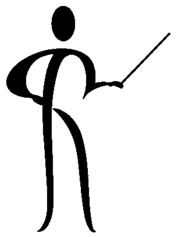
- Parentheses are required around the expression to be tested.

```
if (comparison expression)
        statement ;
if (comparison expression)
{
        statement1;
        statement2;
}
if (comparison expression)
        statement1;
else
        statement2;
```

if Statement, 2

```
if (comparison expression)
{   statement1;
    statement2;
}
else
{   statement3;
    statement4;
}
if (comparison expression)
;           /* could also use empty code block { } here */
else
    statement1;
```

switch and break Statements



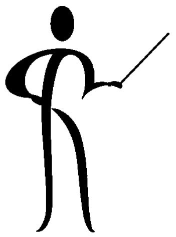
```
switch
   {  case 'm':
        marriedCt++;
        System.out.println("Married = " + marriedCt);
        break;
      case 's':
      case 'd':
        singleCt++;
        System.out.println("Single or Divorced = " + singleCt);
        break;
      case 'w':
        widowedCt++;
        System.out.println("Widowed = " + widowedCt);
        break;
      default:
        System.out.println("Status must be m, s, d, or w");
   } /* end of switch */
```

Looping in Java

- Java provides three methods to cause program loops:

– while	Loop while condition is true (might never execute)
– do-while	Loop while condition is true (always executes at least once)
– for	Loop using automatic increment/decrement until limit is reached (might never execute)

while Statement



- while executes a statement or a group of statements as long as a stated condition is true

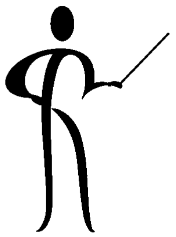
```
/* Blastoff.java */
public class Blastoff {
  public static void main ( String[] args ) {
       int ctr = 10;
       System.out.println("Counting down to blastoff!");
       while ( ctr > 0 )
        {
          System.out.println( ctr-- );
       } /* end of while */
     System.out.println();
     System.out.println("BLAST OFF!");
  } /* End of Main */
} // end Blastoff
```

break and continue

- The break command may also be used to exit loops
- The continue statement causes you to skip remaining statements in a loop and return to the loop control test

```
int a = 0;
int b;
while ( a < 6 )
{
    b = a++;
    if ( b == 3 )
          continue;     // iterate loop
    if ( b > 4 )
          break; // exit loop
    System.out.println("Value in b = " + b);
}
```

do...while Looping



- The do...while construct is similar to the while, it just puts the test last rather than first
- This is significant since the loop is an exit-condition or post-test loop, allowing the structured programming construct DO UNTIL to be performed
- Since the loop condition is checked last, the loop will always execute at least once

```
do {
       // . . .
       // loop code goes here
       // . . .
     } while (y_or_n != 'Y');
```

for Loop

- for loops offer automatic incrementing (or decrementing) of a control value while processing the loop.

```
for (initial_value; true_test; incrementor)
{
/* Java statements go here */
}
```

  - The initial value is set before the loop begins
  - The true_test is a conditional expression, like those in if or switch (without parentheses); the test is performed each time the loop iterates
  - If the test is true the loop is executed, if the test is false the loop is not executed
  - The incrementor “bumps” after the loop body is executed

Arrays

- An array is a group of field values of the same type stored one after another in the computer
- Java allows the processing of an array as a unit, or the processing of individual items in the array
- Arrays are often called tables
- Data values in an array may be individually addressed using an index or subscript
- The first item in an array uses the index/subscript zero [0], the largest useable subscript is the array size - 1
- Indexes/subscripts may be literal values, expressions, or integer variables
- Index/subscript values/variables must be enclosed in brackets “ [ ] ” when used

Vectors

- Vectors are in standard library and provide more flexibility
- Unlike conventional arrays, vectors do not have a fixed size, vector size may shrink and grow as needed
- Entries may be easily inserted into the middle of a vector
- Check the Java SDK API documentation for more information about the Vector class, you'll find that it includes various options for handling arrays such as:

– add	Add or insert items to the vector
– addElement	Add element to end of vector and
– elementAt / get	Return specific vector element
– equals	Compare vector to named object
– firstElement	Return element 0 from vector
– and more!	

Array Exceptions

- Java treats all arrays as objects and will throw an exception if an array is used improperly
- This saves the problems often created in other languages by index (subscript) values that go beyond the limits
- ArrayIndexOutofBoundsException
  - Array index less than zero or greater than or equal to array size
- ArrayStoreException
  - Attempt to store wrong type of object into array
- NegativeArraySizeException
  - Attempt to allocate array with fewer than zero elements

Exception Handling

- One of Java's greatest strengths exception handling
- Java’s mechanism easy, comprehensive, and complete
- Applications and Applets that are properly designed use exception handling to reduce errors and ease debugging
- Exception handling includes statements and code blocks:
  - try Identifies the block of code to be "checked"
  - throw Causes an exception, many methods throw exceptions as part of their design
  - catch Catch block(s) contain code to handle (or at least react to) thrown exceptions
  - finally Contains code so important that it should execute even if a try block does not complete normally due to throwing an exception

Exceptions

- All exceptions in Java inherit from the Throwable class
- Throwable is extended by two other important classes:
  - Error signals problems inside the JVM
  - Exception signals problems probably in your code
- The Exception class is extended to two subclasses:
  - IOExceptions usually when a program did something illogical with a file (tried to read past the end)
  - RuntimeExceptions happen when some programming error occurs such as using an invalid array
- A method's header can indicate exceptions expected to be thrown (not handled) by the method
- A method must catch any exceptions that occur or advertise those exceptions not handled and passed on

  public String getRecord() **throws IOException** { ... }

Java Packages

- Java was designed with reusability and teamwork in mind
- Source code and classes can be grouped together into packages that share a common directory path
- All classes in a package are considered to be the "friend" of others in the package -- classes, methods, and variables that are not marked as private will be available to all classes in the same package
- (contrary to the notion of encapsulation)
- All of Java's built-in classes are accessed via a package structure (e.g. java.awt, or java.io)
- By placing code into packages, the likelihood of ambiguous names is reduced

Import Statement(s)

- The import statement makes classes within a package available without requiring fully-qualified names
- Any Java program could reference java.io.inputStreamReader whenever the class is needed
- By using the statement import java.io.*; all classes in the java directory and io subdirectory are available without fully-qualified names
- Note to C/C++ programmers: NOT same as #INCLUDE

  import java.io.inputStreamReader;

  – Makes inputStreamReader available without full qualification

  import java.io.*;

  – Makes all java.io classes available without full qualification (most frequently used method)

CLASSPATH

- In order to find classes at execution time, you can help Java by creating or modifying the environment variable named "CLASSPATH"
- CLASSPATH includes the directory and subdirectory names where all of your normal Java classes are stored
- It is also a good idea to create a CLASSPATH entry for a single dot "." to make sure that the current directory is available
- The CLASSPATH is an environment variable and must be set appropriately
- Here is an example from a Windows NT system:

```
.;d:\;d:\Java1;
```

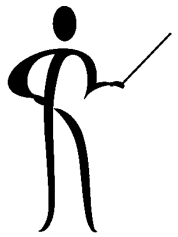
Don't Re-Invent the Wheel!

- Java's classes provide a varied set of built-in methods for use in verifying values, so you don't have to code them!
- Part of the promise of Java and other object-oriented tools is that you can reuse large quantities of code, testing for values is a good example of where exisiting code can be leveraged
- Take a look at the Java SDK API documentation and:
  - Look up the Integer class's parseInt method
  - Look up the Double class's parseDouble method
  - Look to see if similar methods are available for Float, Boolean, or Char data
  - Look up the Date class's after and before methods

Introduction to Derived Classes

- Java allows you to use one class declaration, known as a superclass or base class, as the basis for the declaration of a second class, known as a subclass or derived class
- For example:
  - You might declare an Employee class that describes your company's employees
  - Next, you might declare a Sales class, based on the Employee class, that describes employees in the sales department
- This section of the course covers that part of Object Programming known as Class Derivation or Inheritance

Access



- Four rules apply to the availability of instance variables and methods outside of the class they are defined in:

– Default	All classes within a package have full access to anything not marked private
– private	Not available outside of its own class
– protected	Available within its own class and available to subclasses
– public	Available to all classes

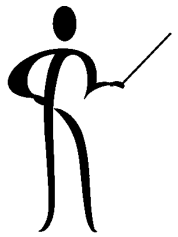
Inheritance

- Just like the cherished heirlooms you hope to inherit, a subclass/derived class inherits all of the non-private data members and member methods (except constructors) from its superclass or base class

```
public class Base {
   public  short baseMember;
   public  void  SetBaseMember( short baseValue ) {. . .};
};
```

- Java uses the terms Superclass for the base class, and Subclass for the derived class

Protected Access Code (revisited)



- A derived class inherits all nonprivate (public & protected) data members and member functions from its base class
  - Public data and methods are available to all
  - Protected data and methods can be accessed: by member methods, friends of its class, member methods, or friends classes derived from its class
- When declaring classes without derived classes:
  - Declare data members as private
  - Declare member functions as public
- When using derived classes:
  - Declare data members as protected
  - Declare member functions as public

Derivation Chain

- Declare three classes, A, B, and C, where class B extends A and C extends B:

– Class A	Variables and methods
– Class B extends A	Variables and methods Inherited members from Class A
– Class C extends B	Variables and methods Inherited members from Class B Inherited members from Class A

  - When you create an object from class B, it will inherit the nonprivate members from class A
  - When you create an object of class C, it will inherit the nonprivate members from class B, which may include some previously inherited members from class A
  - When an object from class C is created, the compiler follows the chain and the constructor from class A, then B, then C are called

Interfaces, 1

- Interfaces are collections of method names and parameter lists, no method code is stored in the interface
- The interface is a "promise" that those who derive from the class will provide a minimum number of methods

```
public interface ECustomer {
   void public TakeAnOrder (Prospect eCust);
   void public SendConfirmation(Prospect eCust);
}
public class EStore implements ECustomer {
   void public TakeAnOrder (Prospect eCust) {
       // order taking code goes here
   }
   void public SendConfirmation(Prospect eCust) {
       // code to send memo to customer
   }
}
```

Interfaces, 2

- Each class that implements an interface is responsible for creating its own methods
- There is no guarantee that the methods in different implementations will be similar, but, it seems likely
- Perhaps one of the most-used interfaces is Runnable, this interface provides a series of methods useful when "animating" a page

Interfaces vs Multiple Inheritance

- One of the areas gurus of Object-Orientedness like to argue over is whether or not Multiple Inheritance is good
- Creators of Java decided that Multiple Inheritance was too complex, so, classes may only extend one superclass
- This is sufficient for most applications and many applets
- Often, Multiple Inheritance is desired due to a need to select the appropriate overloaded methods based upon the data being presented (one style of polymorphism)
- Java allows a class to implement multiple interfaces, providing much of what is called Multiple Inheritance
- All methods of an interface are public implicitly, it is incorrect to define them otherwise
- All methods of an interface must be overridden in order to use it

GUI Applications

- One reason Java caught on so quickly is that the Graphical User Interface (GUI) has the same "look and feel" regardless of what platform you use it on
- Originally, Java offered the Abstract Window Toolkit (AWT) that supports basic GUI programming
- Unfortunately, in order to be as portable as possible (Write Once Run Everywhere) AWT applications are not as rich as those that are native to the platform
- Java 1.1 and Java 2 support a Java Foundation Class (JFC) library subset usually called Swing
- Swing applications are less intertwined with the host GUI and are smaller, faster, more-portable, and richer
- Unfortunately, Swing may not be available in down-level browsers so most applets today still use AWT

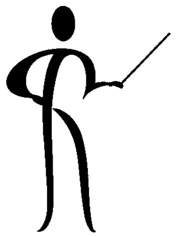
AWT and Swing

- The Abstract Window Toolkit (AWT offers a functional, if somewhat limited set of capabilities
- Java Foundation Classes (JFC) or Swing does not replace AWT, but, provides additional functionality
- Reasons to use JFC or Swing include:
  - Consistent "look and feel" across platforms
  - Richer, more functional user interface components
  - Fewer, implementation-specific bugs
- Reasons to use AWT instead include:
  - Java Swing "look and feel" not the same as native interface
  - Simpler to learn
  - Somewhat faster on initial load
  - Supported by all Java-enabled browsers

Windows

- Window is the basic building block for AWT applications, a Window object has no borders or menu bar and BorderLayout is the default layout manager
- Until now, most of programs have been command-line driven, and perhaps, a bit dull in appearance
- Two major methods for creating GUI applications are:
  - Frames Stand-alone window applications
  - Applets Window designed to open up a browser page
- Objects placed on windows (often called widgets) include:
  - Frames
  - Buttons
  - Lists
  - Checkboxes

Frames



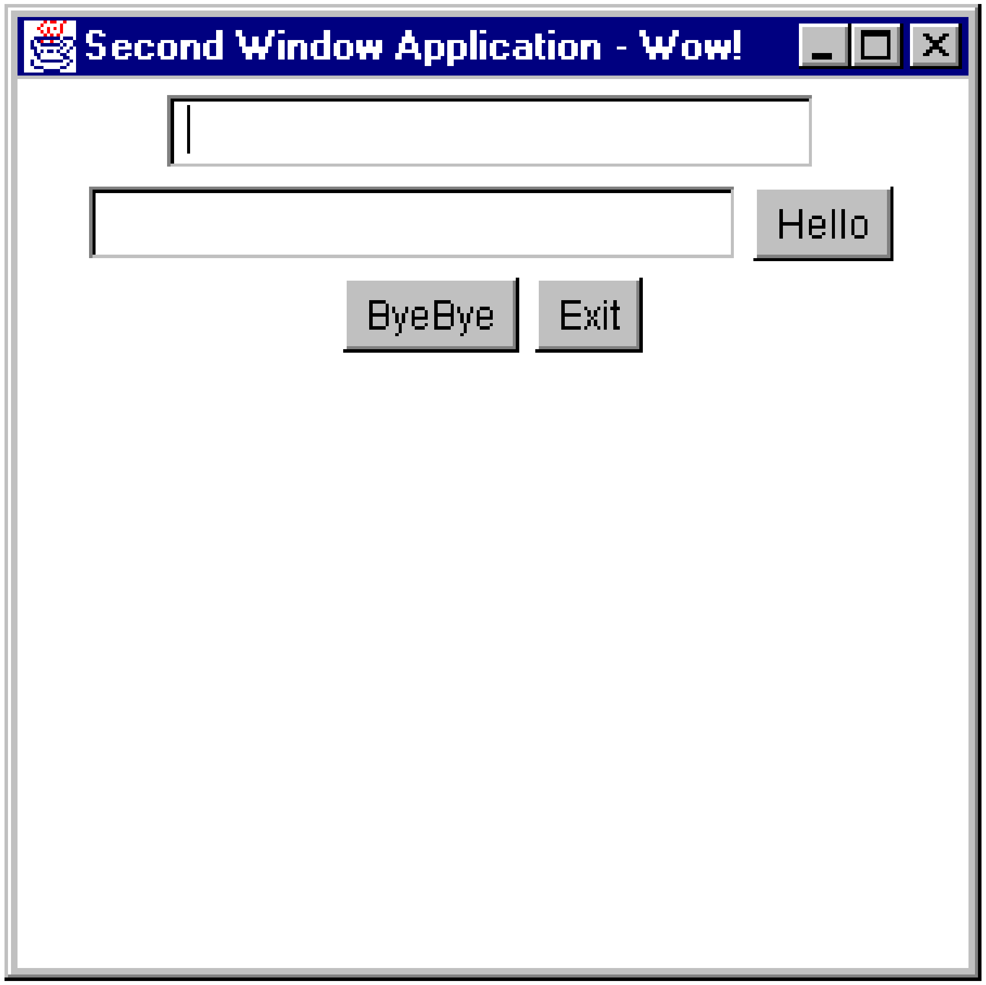
- A Frame is a stand-alone windowed application created using the following steps:
  - Declare a Frame variable
  - Create an instance of Frame
  - Change the size of the frame
  - Display the frame using show

```
// Frame1.java - Sample AWT Application
import java.awt.*;
class Frame1 {
    public static void main(String[] args)  {
       Frame myFrame = new Frame("First Window Application - Wow!");
       myFrame.setSize(400,200);  // set width,height
       myFrame.show();
    } // end main
} // end Frame1
```

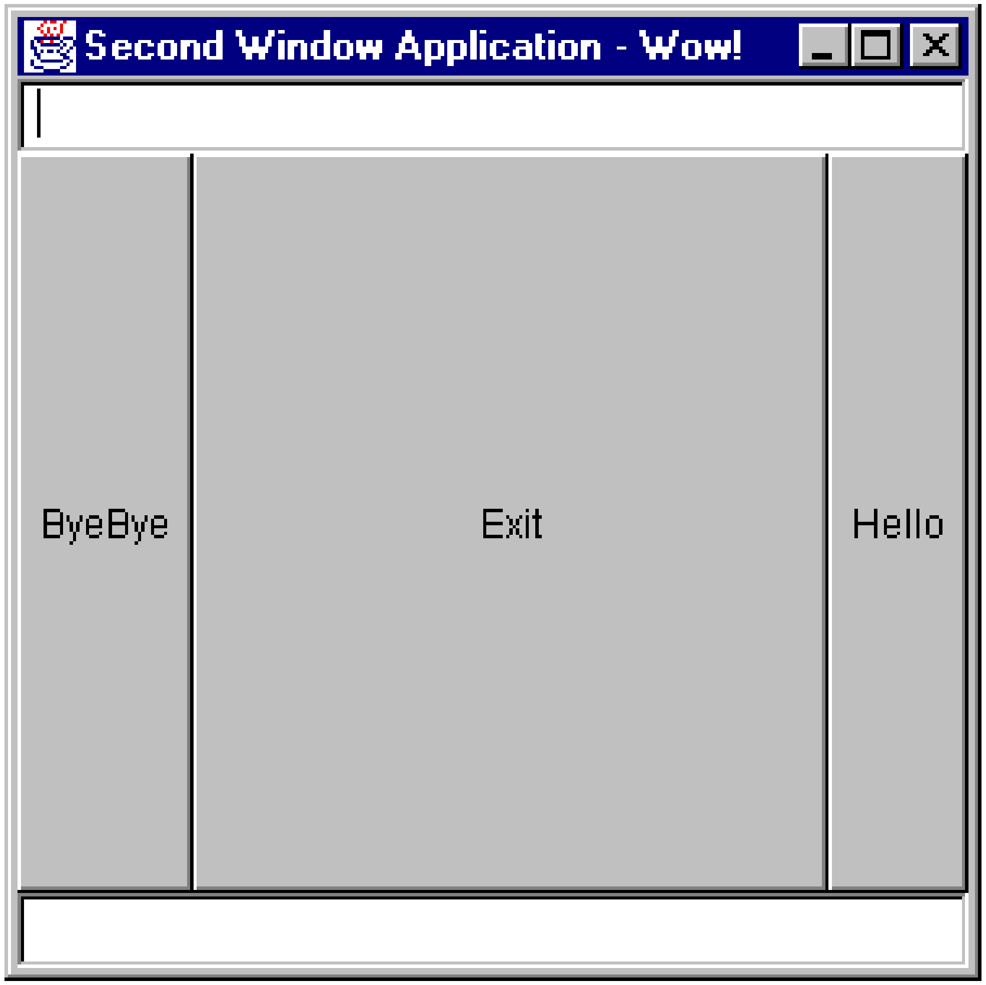
Layout Managers

- Each container uses a layout manager, an object that implements the LayoutManager interface
- The most frequently used layout managers are: BorderLayout, FlowLayout, GridLayout, and GridBagLayout
- GridBagLayout is by far the most flexible and useful, unfortunately, GridBagLayout is the most complex to use
- On the next several pages are illustrations of the four major layouts
- In all four cases the page has five widgets added:
  - Two text fields
  - Three buttons

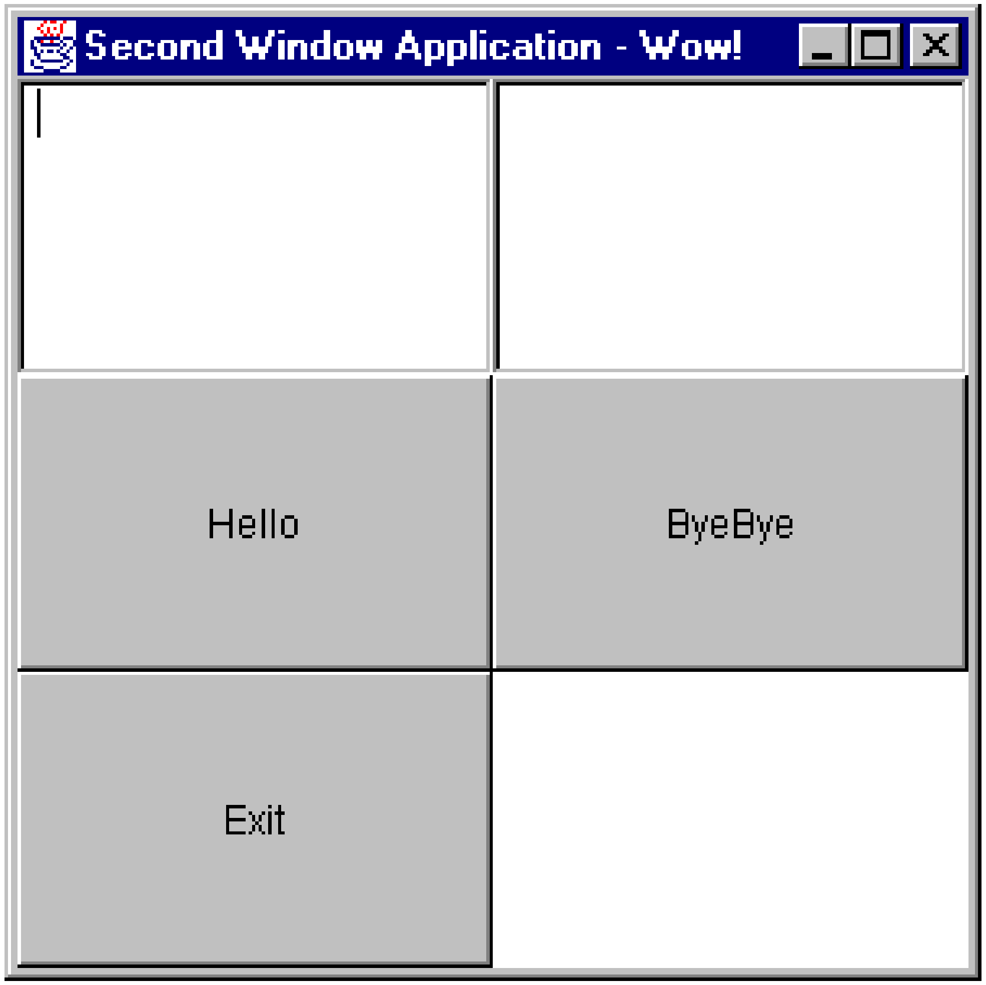
FlowLayout



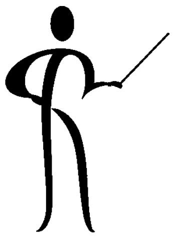
BorderLayout

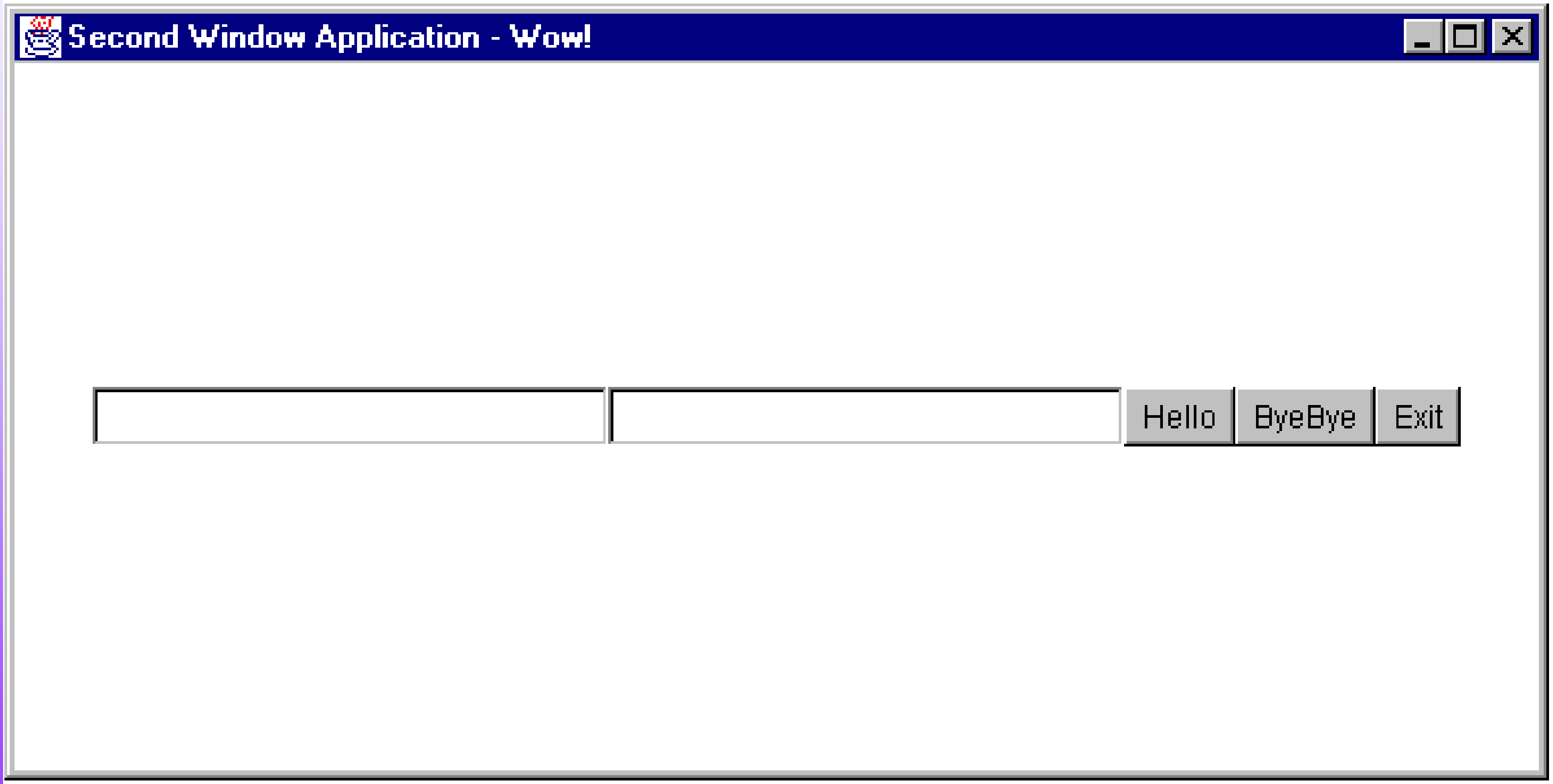


GridLayout

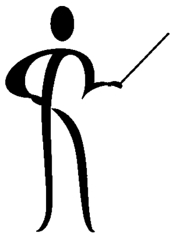


GridBagLayout



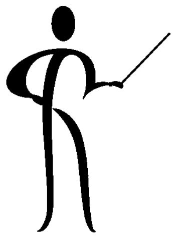


AWT Widgets



- Check the Java documentation for the java.awt class to see more information on the widgets supported including:
  - Container (Panel, ScrollPane, Window (Frame, Dialog, FileDialog))
  - TextComponent (TextField, TextArea)
  - Button
  - Canvas
  - Checkbox
  - Choice
  - Label
  - List
  - Scrollbar

Using Widget Classes



- Use the AWT widget that best suits application needs
- Container classes describe windows and sub-windows:

  Panel, ScrollPane, Window, Frame, Dialog, FileDialog, Text
- Components hold text data:

  TextField, TextArea
- GUI widgets allow user interaction with less typing:

  Button, Canvas, Checkbox, Choice, Label, List

Containers

- Panel is not a separate window, it is used to hold large areas inside a Frame, Dialog, or another Panel (applet is a subclass of panel)
- ScrollPane Defines horizontal and vertical scrollbars
- Window, top-level window without borders or menu, typically not used, Frame and Dialog extend window and are usually used instead
- Frame, resizable window with a title, menu, and whatever belongs on a top-level window in the environment
- Dialog, window typically displayed within another window, might have title or menu, might be resizable, might be modal (requiring response until dismissed)
- FileDialog, extends dialog and displays a file open that is typical for the environment

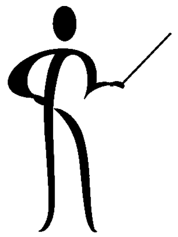
TextComponents

- TextComponent is an abstract class and cannot be instantiated, use TextField or TextArea instead
- TextField displays a single line of text that may be edited (or not)
- TextArea displays multi-line text that may be edited (or not), one of the constructors allows definition of scrollbars in the TextArea

GUI Widgets

- Button defines a push button and its text label
- Canvas used to create an area for drawing graphics or placing images
- Checkbox creates a GUI checkbox with its associated label
- Checkboxes in a CheckboxGroup create a GUI radio-style button with its associated label
- Label is a line of read-only text
- Choice creates a pull-down/pop-up list of choices
- List defines a scrollable list of strings

Applet Methods



- Applets use specific methods at particular times:

– init()	Executes when applet is opened, init(), same as main in application
– start()	Executes when web page is reloaded
– paint()	Executes when the applet is redrawn
– stop()	Executes when a user leaves web page
– destroy()	Executes when applet is unloaded

- Applets also frequently use three methods from awt.Component:

– paint()	Paint component
– repaint()	Schedule a call of update() soon
– update()	Redraws applet, background and calls paint()

Applet Class, JApplet Class and Package

- Packages are collections of related Java classes sharing a common directory path
- The Applet Class may be found in java.awt
- The JApplet Class may be found in javax.swing

java.awt Containers

- java.awt defines several containers:
  - Applet
  - Container
  - Dialog
  - Frame
  - Panel
  - ScrollPane
  - Window

javax.swing Containers

- javax.swing contains a set of Swing-specific containers:
  - Box
  - JApplet
  - JDesktopPane
  - JDialog
  - JFrame
  - JInternalFrame
  - JLayeredPane
  - JPanel
  - JRootPane
  - JScrollPane
  - JSplitPane
  - JTabbedPane
  - JViewport
  - JWindow

Panels

- Panels are containers, that is other GUI widgets (including Panel, Button, Choice, List, etc...) are contained within them
- A Frame or Applet may contain multiple Panels
- Layouts might apply for all objects in an entire window, not always yielding the most attractive results
- Panels use a method named setLayout that allow the layout to change for each Panel

Event Handling

- User interfaces in Java are driven by processing events
- Events correspond to some user action, including:
  - Mouse clicks
  - Selection of buttons
  - Selection of Checkbox, Choice, or List values
  - Window closing
- Events are processed by Event Handling methods
- Event Handling methods return a boolean value
- If the event handler returns true, the event handler has intercepted an event successfully
- If the event handler returns false, the event is forwarded and the programmer may override the default behavior
- Event handling has changed beginning with Java 1.1; the older event handling still works, but is deprecated

Event Listener Interface Summary

Action	Receiving action events
AdjustmentListener	Receiving adjustment events.
AWTEventListener	Receiving notification of dispatch event
ComponentListener	Receiving component events
ContainerListener	Receiving container events
FocusListener	Receiving keyboard focus events
InputMethodListener	Receiving input method events
ItemListener	Receiving item events
KeyListener	Receiving keyboard events
MouseListener	Receiving "interesting" mouse events (press, release, click, enter, and exit)
MouseMotionListener	Receiving mouse motion events
TextListener	Receiving text events
WindowListener	Receiving window events

Drawing Graphics

- Graphics has useful features including text and fonts:
  - java.awt.font
  - java.awt.fontMetrics
  - java.awt.Graphics
  - java.awt.Color
  - java.awt.component
- Look up java.awt.graphics in the API most of the methods use arguments regarding position, width, and height
- The java.awt.Graphics class also includes several options for colors, the standard colors supported are: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow -- these are referenced using the Color constructor: Color(red,green,blue);

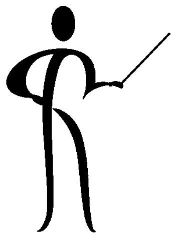
Introduction to Database Connectivity with JDBC

- Sun released the first version of JDBC (Java DataBase Connectivity) in the summer of 1996
- JDBC allows programmers to use SQL to:
- Connect to a database
- Query a database
- Update a database
- JDBC programs are database vendor and platform independent (mostly)
- JDBC2 was released in 1998 and is not fully supported in all environments yet

java Database Support and JDBC

- JDBC allows connections to all of the major database vendors: Oracle, IBM DB2, Microsoft, and others
- Some vendors (Oracle and IBM) now build a JVM into the database engine to support Java Stored Procedures and Java-based access
- JDBC is generic
- Even though JDBC is generic, you may find it is best to obtain JDBC drivers from the database vendor rather than using the ones from Sun
- Your application uses the JDBC driver, the JDBC driver communicates with the database

SQL Support in JDBC



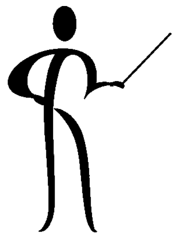
- Invented by IBM, first brought to market by Oracle
- ISO/ANSI standard

```
select empno,lastname,firstname,job,sal
   from emp
   where deptno in (10,20)
   order by lastname,firstname
insert into emp
   (empno,lastname,firstname,job,sal)
    values
          (123,'CODD','TED','GURU',123.45)
update emp
   set sal = sal * 1.07
   where job = 'HERO'
delete from emp
   where empno = 123
```

Connecting to Database

- To use a JDBC database in a Java program:
  - Import java.sql.*
  - Load the JDBC driver
  - Connect to the database using JDBC
  - Execute SQL statements and process results
  - Disconnect from the database

Java Accessing the Database



- The methods used for most database access are:
  - jdbc.odbc.JdbcOdbcDriver

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

  - java.sql.DriverManager and java.sql.Connection

```
Connection c  =
  DriverManager.getConnection("jdbc:odbc:EMPLOYEE");
```

  - java.sql.Statement

```
Statement s = c.createStatement();
```

  - java.sql.ResultSet

```
ResultSet rs = s.executeQuery(sql);  // run sql stmt
```

Result Sets

- Immediately after a query, the result set pointer points one record before the first row returned, use the next method to get the first record

```
String jellyBeanName = rs.getString("JBEANNAME");
// get using col name
```

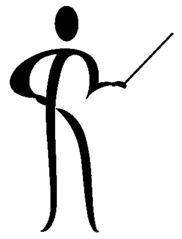
Oracle8i support for Java, 1

- Oracle8i includes a Java Virtual Machine specifically engineered by Oracle to provide multi-threaded support of Java applications instead of having separate JVMs for each bit of Java. This means greatly improved performance for Java applications running with database code.
- Oracle also supports the creation of stored procedures using Java in addition to those created with PL/SQL.
- Oracle8i Release 2 (8.1.6) dramatically improves Java support including: Java 2 (JDK 1.2) support, JDBC 2.0, multi-byte character support, debugging support, performance improvements, and support for Advanced Queueing. The new release also supports an XML parser implemented with Java.

Oracle8i support for Java, 2

- Java support for programming includes:
  - Java stored procedures
  - Enterprise Java Bean 1.0 support
  - Support for CORBA 2.0 standards.
  - Java support for SQL includes:
  - JDBC
  - SQLJ
- SQLJ statements are translated by an SQLJ Preprocessor before Java code is submitted to JDBC.
- Direct JDBC support is more complex, but, yields more control.

Oracle JDBC



- Oracle provides two basic JDBC access methods:
  - “thin” driver
  - “OCI” driver
- The “thin” driver can be downloaded along with an applet and is how an applet must access Oracle
- The “OCI” driver requires an Oracle client environment complete with Net8/SQL*Net connectivity
- The “OCI” driver provides faster database access, the “thin” driver provides the most flexibility

Conclusion

- Java is “the thing” in the IT industry today
- Oracle, IBM, Sun, and many others have spent vast sums to make Java the emerging “standard”

  An ISO/ANSI standard is still needed before Java can truly claim to be a standard
- Java is a third-generation language, only by using an advanced development environment like Oracle JDeveloper or IBM Visual Age for Java can manually writing large amounts of code be avoided
- Java includes a wealth of code in the standard libraries that should be leveraged to reduce the amount of code written
- Oracle and IBM are both allowing stored database procedures using Java in their latest releases
- Java is a good investment of your time and energy!

King
Training Resources

To contact the author:

John Jay 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
http://www.kingtraining.com