Java Programming Language

SL-275
Preface

About This Course
Course Goals

This course provides you with knowledge and skills to:

• Program and run advanced Java™ applications and applets

• Help you prepare for the Sun™ Certified Java Programmer and Developer examinations
Course Overview

This course covers the following areas:

- Syntax of the Java programming language
- Object-oriented concepts as they apply to the Java programming language
- Graphical user interface (GUI) programming
- Applet creation
- Multithreading
- Networking
Module-by-Module Overview

• Module 1 – Getting Started
• Module 2 – Identifiers, Keywords, and Types
• Module 3 – Expressions and Flow Control
• Module 4 – Arrays
• Module 5 – Objects and Classes
• Module 6 – Advanced Language Features
• Module 7 – Exceptions
• Module 8 – Building GUIs
Module-by-Module Overview

- Module 9 – The AWT Event Model
- Module 10 – The AWT Component Library
- Module 11 – Java Foundation Classes
- Module 12 – Introduction to Java Applets
- Module 13 – Threads
- Module 14 – Stream I/O and Files
- Module 15 – Networking
Course Objectives

• Describe key language features
• Compile and run a Java application
• Understand and use the online hypertext Java technology documentation
• Describe language syntactic elements and constructs
• Understand the object-oriented paradigm and use object-oriented features of the language
• Understand and use exceptions
• Develop a graphical user interface
• Describe the Java technology platform’s Abstract Window Toolkit (AWT) used to build GUIs
Course Objectives

• Program to take input from a GUI
• Understand event handling
• Describe the main features of Swing
• Develop Java applets
• Read and write to files and other data sources
• Perform input and output to all sources without the use of a GUI
• Understand the basics of multithreading
• Develop multithreaded Java applications and applets
• Develop Java client and server programs using Transmission Control Protocol/Internet Protocol (TCP/IP) and User Datagram Protocol (UDP)
### Skills Gained by Module

<table>
<thead>
<tr>
<th>Skills Gained</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describe key language features</td>
<td>1</td>
</tr>
<tr>
<td>Compile and run a Java application</td>
<td>2</td>
</tr>
<tr>
<td>Understand and use the online hypertext Java technology documentation</td>
<td>3</td>
</tr>
<tr>
<td>Describe language syntactic elements and constructs</td>
<td>4</td>
</tr>
<tr>
<td>Understand the object-oriented paradigm and use object-oriented features of the language</td>
<td>5</td>
</tr>
<tr>
<td>Understand and use exceptions</td>
<td>6</td>
</tr>
<tr>
<td>Develop a GUI</td>
<td>7</td>
</tr>
<tr>
<td>Describe the Java technology platform’s Abstract Window Toolkit used to build GUIs</td>
<td>8</td>
</tr>
<tr>
<td>Create a program to take input from a graphical user interface</td>
<td>9</td>
</tr>
<tr>
<td>Understand event handling</td>
<td>10</td>
</tr>
<tr>
<td>Describe the main features of Swing</td>
<td>11</td>
</tr>
<tr>
<td>Develop Java applets</td>
<td>12</td>
</tr>
<tr>
<td>Understand the basics of multithreading</td>
<td>13</td>
</tr>
<tr>
<td>Develop multithreaded Java applications and applets</td>
<td>14</td>
</tr>
<tr>
<td>Read and write to files and other data sources</td>
<td>15</td>
</tr>
<tr>
<td>Perform I/O to all sources without the use of a GUI</td>
<td></td>
</tr>
<tr>
<td>Skills Gained</td>
<td>Module</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Develop Java client and server programs using TCP/IP and UDP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>
## Guidelines for Module Pacing

<table>
<thead>
<tr>
<th>Module</th>
<th>Day 1</th>
<th>Day 2</th>
<th>Day 3</th>
<th>Day 4</th>
<th>Day 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>About This Course</td>
<td>A.M.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module 1 – Getting Started</td>
<td>A.M.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module 2 – Identifiers, Keywords, and Types</td>
<td>A.M.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module 3 – Expressions and Flow Control</td>
<td>P.M.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module 4 – Arrays</td>
<td>P.M.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module 5 – Objects and Classes</td>
<td></td>
<td></td>
<td>A.M.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module 6 – Advanced Language Features</td>
<td></td>
<td></td>
<td></td>
<td>P.M.</td>
<td></td>
</tr>
<tr>
<td>Module 7 – Exceptions</td>
<td></td>
<td></td>
<td></td>
<td>A.M.</td>
<td></td>
</tr>
<tr>
<td>Module 8 – Building GUIs</td>
<td></td>
<td></td>
<td></td>
<td>A.M.</td>
<td></td>
</tr>
<tr>
<td>Module 9 – The AWT Event Model</td>
<td></td>
<td></td>
<td></td>
<td>P.M.</td>
<td></td>
</tr>
<tr>
<td>Module 10 – The AWT Component Library</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A.M.</td>
</tr>
<tr>
<td>Module 11 – Java Foundation Classes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A.M.</td>
</tr>
<tr>
<td>Module 12 – Introduction to Java Applets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P.M.</td>
</tr>
<tr>
<td>Module 13 – Threads</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A.M.</td>
</tr>
<tr>
<td>Module 14 – Stream I/O and Files</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P.M.</td>
</tr>
<tr>
<td>Module 15 – Networking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P.M.</td>
</tr>
</tbody>
</table>
Topics Not Covered

- General programming concepts. This is not a course for people who have never programmed before.
- General object-oriented concepts.
How Prepared Are You?

Before attending this course, you should have completed

- SL-110: *Java Programming For Non-Programmers*

or have

- Created compiled programs with C or C++
- Created and edited text files using a text editor
- Used a World Wide Web (WWW) browser, such as Netscape Navigator™
Introductions

• Name

• Company affiliation

• Title, function, and job responsibility

• Programming experience

• Reasons for enrolling in this course

• Expectations for this course
How to Use Course Materials

- Course Map
- Relevance
- Overhead Image
- Lecture
- Exercise
- Check Your Progress
- Think Beyond
Course Icons

- Reference
- Discussion
- Exercise
Typographical Conventions

• **Courier** – Commands, files and directories, and on-screen computer output

• **Courier bold** – Input you type

• **Courier italic** – Variables and command-line placeholders

• *Palatino italics* – Book titles, new words or terms, and words that are emphasized
Module 1

Getting Started
Objectives

• Describe key features of Java programming language
• Describe the Java virtual machine’s (JVM) function
• Describe how garbage collection works
• List the three tasks performed by the Java platform that handle code security
• Define the terms class, packages, applets, and applications
Objectives

- Write, compile, and run a simple Java application

- Use the Java technology application programming interface (API) on-line documentation to identify the methods of the \texttt{java.lang} package
Relevance

• Is the Java programming language a complete language or is it just useful for writing programs for the Web?

• Why is another programming language needed?

• How does the Java technology platform improve on other language platforms?
What Is the Java Programming Language?

- The Java programming language is:
  - A programming language
  - A development environment
  - An application environment
  - A deployment environment

- Similar in syntax to C++; similar in semantics to SmallTalk

- Used for developing both *applets* and *applications*
Primary Goals of the Java Programming Language

• Provides an easy-to-use language by:
  • Avoiding the pitfalls of other languages
  • Being object-oriented
  • Enabling users to create streamlined and clear code
Primary Goals of the Java Programming Language

- Provides an interpreted environment for:
  - Improved speed of development
  - Code portability
- Enables users to run more than one thread of activity
- Supports dynamically changing programs during runtime
- Furnishes better security
Primary Goals of the Java Programming Language

The following features fulfill these goals:

• The Java virtual machine (JVM)

• Garbage collection

• Code security
The Java Virtual Machine

- Provides hardware platform specifications
- Reads compiled byte codes that are platform independent
- Is implemented as software or hardware
- Is implemented in a Java technology development tool or a Web browser
The Java Virtual Machine

- JVM provides definitions for the:
  - Instruction set (central processing unit [CPU])
  - Register set
  - Class file format
  - Stack
  - Garbage-collected heap
  - Memory area
The Java Virtual Machine

• Bytecodes that maintain proper type discipline from the code.

• The majority of type checking is done when the code is compiled.

• Every Sun approved implementation of the JVM must be able to run any compliant class file.
Garbage Collection

- Allocated memory that is no longer needed should be deallocated
- In other languages, deallocation is the programmer’s responsibility
- The Java programming language provides a system-level thread to track memory allocation
- Garbage collection:
  - Checks for and frees memory no longer needed
  - Is done automatically
  - Can vary dramatically across JVM implementations
Code Security

The Java application environment performs as follows:

Compile

```
javac Hello.java
```

Hello.class

Runtime

```
java Hello
```

Network

Class loader

Byte code verifier

Interpreter

Runtime

JIT code generator

Hardware
Java Runtime Environment

• Performs three main tasks:
  • Loads code
  • Verifies code
  • Executes code
Class Loader

- Loads all classes necessary for the execution of a program
- Maintains classes of the local file system in separate "namespaces"
- Prevents spoofing
Bytecode Verifier

Ensures that:

- The code adheres to the JVM specification
- The code does not violate system integrity
- The code causes no operand stack overflows or underflows
- The parameter types for all operational code are correct
- No illegal data conversions (the conversion of integers to pointers) have occurred
A Basic Java Application

HelloWorldApp.java

```java
1 //
2 // Sample HelloWorld application
3 //
4 public class HelloWorldApp{
5   public static void main (String args[]) {
6       System.out.println("Hello World!");
7     }
8 }
```
Compiling and Running
HelloWorldApp

- Compiling HelloWorldApp.java
  javac HelloWorldApp.java

- Running an application
  java HelloWorldApp

- Locating common compile and runtime errors
Compile-Time Errors

- `javac`: Command not found

  System.out.println("Hello World");`

- In class `HelloWorldApp`:
  main must be public and static
Runtime Errors

- Can’t find class HelloWorldApp
- Naming
- One public class per file
The Source File Layout

Contains three "top-level" elements:

• An optional package declaration
• Any number of import statements
• Class and interface declarations
Classes and Packages – An Introduction

- Classes and packages:
  - Prominent packages within the Java class library are:
    java.lang
    java.awt
    java.applet
    java.net
    java.io
    java.util
Using the Java API Documentation

- A set of hypertext markup language (HTML) files provides information about the API
- One package contains hyperlinks to information on all of the classes
- A class document includes the class hierarchy, a description of the class, a list of member variables, a list of constructors, and so on
Exercise: Performing Basic Java Tasks

• Exercise objectives:
  • Identify packages, classes, and methods in the Java API documents
  • Identify standard input and output methods
  • Write, compile, and run two simple applications using these methods

• Tasks:
  • Read the documentation
  • Create a Java application
  • Use standard input and output
Check Your Progress

- Describe key features of the Java programming language
- Describe the Java virtual machine’s (JVM) function
- Describe how garbage collection works
- List the three tasks performed by the Java platform that handle code security
- Define the terms class, packages, applets, and applications
- Write, compile, and run a simple Java application
Check Your Progress

- Use the Java technology API online documentation to identify the methods of the `java.lang` package.
Think Beyond

• How can you benefit from using this programming language in your work environment?
Module 2

Identifiers, Keywords, and Types
Objectives

• Use comments in a source program

• Distinguish between valid and invalid identifiers

• Recognize Java technology keywords

• List the eight primitive types

• Define literal values for numeric and textual types

• Define the terms class, object, member variable, and reference variable
Objectives

- Create a class definition for a simple class containing primitive member variables
- Declare variables of class type
- Construct an object using `new`
- Describe default initialization
- Access the member variables of an object using the dot notation
- Describe the significance of a reference variable
- State the consequences of assigning variables of class type
Relevance

• What is your understanding of a class?
• What is your understanding of an object?
Comments

• Three permissible styles of comment in a Java technology program are:

```java
// comment on one line

/* comment on one or more lines */

/** documenting comment */
```
Semicolons, Blocks, and Whitespace

- A **statement** is a single line of code terminated by a semicolon (\(;\)):
  \[
  \text{totals} = a + b + c + d + e + f;
  \]

- A **block** is a collection of statements bounded by opening and closing braces:
  
  ```java
  
  \{
  \text{x} = \text{y} + 1;
  \text{y} = \text{x} + 1;
  \}
  ```
Semicolons, Blocks, and Whitespace

• You can use a *block* in a *class* definition:

```java
public class Date {
    int day;
    int month;
    int year;
}
```

• You can nest block statements

• Any amount of *whitespace* is allowed in a Java program
Identifiers

- Are names given to a variable, class, or method
- Can start with a letter, underscore(_), or dollar sign($)
- Are case sensitive and have no maximum length

Examples:

```java
identifier
username
user_name
_sys_var1
$change
```
# Java Keywords

<table>
<thead>
<tr>
<th>abstract</th>
<th>do</th>
<th>implements</th>
<th>private</th>
<th>throw</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>double</td>
<td>import</td>
<td>protected</td>
<td>throws</td>
</tr>
<tr>
<td>break</td>
<td>else</td>
<td>instanceof</td>
<td>public</td>
<td>transient</td>
</tr>
<tr>
<td>byte</td>
<td>extends</td>
<td>int</td>
<td>return</td>
<td>true</td>
</tr>
<tr>
<td>case</td>
<td>false</td>
<td>interface</td>
<td>short</td>
<td>try</td>
</tr>
<tr>
<td>catch</td>
<td>final</td>
<td>long</td>
<td>static</td>
<td>void</td>
</tr>
<tr>
<td>char</td>
<td>finally</td>
<td>native</td>
<td>super</td>
<td>volatile</td>
</tr>
<tr>
<td>class</td>
<td>float</td>
<td>new</td>
<td>switch</td>
<td>while</td>
</tr>
<tr>
<td>continue</td>
<td>for</td>
<td>null</td>
<td>synchronized</td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>if</td>
<td>package</td>
<td>this</td>
<td></td>
</tr>
</tbody>
</table>
Primitive Types

• The Java programming language defines eight primitive types:
  • Logical  boolean
  • Textual  char
  • Integral  byte, short, int, and long
  • Floating  double and float
Logical – boolean

- The boolean data type has two literals, true and false.

- For example, the statement:

```java
boolean truth = true;
```

declares the variable truth as boolean type and assigns it a value of true.
Textual – **char and String**

**char**

- Represents a 16-bit Unicode character
- Must have its literal enclosed in single quotes (’ ’)
- Uses the following notations:
  - `'a'` The letter `a`
  - `'\t'` A tab
  - `'\u????'` A specific Unicode character, `????`, is replaced with exactly four hexadecimal digits
Textual – char and String

String

• Is not a primitive data type; it is a class
• Has its literal enclosed in double quotes ("   ")

"The quick brown fox jumps over the lazy dog."

• Can be used as follows:

    String greeting = "Good Morning !! \n";
    String err_msg = "Record Not Found !"
Integral – byte, short, int, and long

- Uses three forms – Decimal, octal, or hexadecimal
  
  2
  077
  0xBAAC
  
  The decimal value is two.
  The leading zero indicates an octal value.
  The leading 0x indicates a hexadecimal value.

- Has a default int

- Defines long by using the letter L or l
Integral—`byte`, `short`, `int`, and `long`

- Each of the integral data types have the following range:

<table>
<thead>
<tr>
<th>Integer Length</th>
<th>Name or Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bits</td>
<td>byte</td>
<td><code>-2^7</code> to <code>-2^7 - 1</code></td>
</tr>
<tr>
<td>16 bits</td>
<td>short</td>
<td><code>-2^{15}</code> to <code>-2^{15} - 1</code></td>
</tr>
<tr>
<td>32 bits</td>
<td>int</td>
<td><code>-2^{31}</code> to <code>-2^{31} - 1</code></td>
</tr>
<tr>
<td>64 bits</td>
<td>long</td>
<td><code>-2^{63}</code> to <code>-2^{63} - 1</code></td>
</tr>
</tbody>
</table>
Floating Point – `float` and `double`

- Default is `double`

- Floating point literal includes either a decimal point or one of the following:
  - `E` or `e` (add exponential value)
  - `F` or `f` (`float`)
  - `D` or `d` (`double`)

3.14     A simple floating-point value (a `double`)
6.02E23   A large floating-point value
2.718F    A simple float size value
123.4E+306D A large double value with redundant `D`
Floating Point – `float` and `double`

- Floating point data types have the following ranges:

<table>
<thead>
<tr>
<th>Float Length</th>
<th>Name or Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 bits</td>
<td><code>float</code></td>
</tr>
<tr>
<td>64 bits</td>
<td><code>double</code></td>
</tr>
</tbody>
</table>
Variables, Declarations, and Assignments

```java
1  public class Assign {
2     public static void main(String args []) {
3
4         int x, y; // declare int variables
5         float z = 3.414f; // declare and assign float
6         double w = 3.1415; // declare and assign double
7         boolean truth = true; // declare and assign boolean
8         char c; // declare character variable
9         String str; // declare String
10        String str1 = "bye"; // declare and assign String variable
11        c = 'A'; // assign value to char variable
12        str = "Hi out there!"; // assign value to String variable
13        x = 6;
14        y = 1000; // assign values to int variables
15        ...
16     }
17  }
```
Java Coding Conventions

• Classes:

    class AccountBook
    class ComplexVariable

• Interfaces:

    interface Account

• Methods:

    balanceAccount()
    addComplex()
Java Coding Conventions

• Variables:

    currentCustomer

• Constants:

    HEAD_COUNT
    MAXIMUM_SIZE
Understanding Objects

• Reviewing the history of objects

• Creating a new type, such as MyDate:

        public class MyDate {
               int day;
               int month;
               int year;
        }

• Declaring a variable:

        MyDate myBirth, yourBirth

• Accessing members:

        myBirth.day = 26;
        myBirth.month = 11;
        yourBirth.year = 1960;
Creating an Object

- Declaration of primitive types allocates memory space
- Declaration of nonprimitive types does *not* allocate memory space
- Declared variables are not the data itself, but references (or pointers) to the data
Creating an Object – Memory Allocation and Layout

• A declaration allocates storage only for a reference:

```java
MyDate today;
today = new MyDate();
```

today ????
Creating an Object – Memory Allocation and Layout

• Use the `new` operator to allocate and initialize storage:

```java
MyDate today;
today = new MyDate();
```

| today | ????
| day   | 0 |
| month | 0 |
| year  | 0 |
Creating an Object – Memory Allocation and Layout

- Assign newly created object to reference variable:

```java
MyDate today;
today = new MyDate();
```

<table>
<thead>
<tr>
<th>today</th>
<th>0x01abcdef</th>
</tr>
</thead>
<tbody>
<tr>
<td>day</td>
<td>0</td>
</tr>
<tr>
<td>month</td>
<td>0</td>
</tr>
<tr>
<td>year</td>
<td>0</td>
</tr>
</tbody>
</table>
Assignment of Reference Variables

• Consider the following code fragment:

```java
int x = 7;
int y = x;
String s = "Hello";
String t = s;
```
Assignment of Reference Variables

```java
int x = 7;
int y = x;
String s = "Hello";
String t = s;
```

- Two variables refer to single object

  ```
  x  7
  y  7
  s  0x01234567
  t  0x01234567
  “Hello”
  ```

  ```java
  t = "World";
  ```

- Reassignment makes two variables point to two objects

  ```
  x  7
  y  7
  s  0x01234567
  t  0x12345678
  “World”
  ```
Terminology Recap

- Class
- Object
- Reference type
- Member
Exercise: Using Identifiers, Keywords, and Types

• Exercise objectives:
  • Using the correct Java keywords, create a class and an object from the class
  • Compile and run the program
  • Verify that the references are assigned and manipulated as described in this module

• Tasks:
  • Create a class and corresponding objects
  • Investigate reference assignments
Check Your Progress

• Use comments in a source program
• Distinguish between valid and invalid identifiers
• Recognize Java technology keywords
• List the eight primitive types
• Define literal values for numeric and textual types
• Define the terms *class, object, member variable*, and *reference variable*
Check Your Progress

• Create a class definition for a simple class containing primitive member variables
• Declare variables of class type
• Construct an object using new
• Describe default initialization
• Access the member variables of an object using the dot notation
• Describe the significance of a reference variable
• State the consequences of assigning variables of class type
Think Beyond

• What classes and objects appear in your existing applications?
Module 3

Expressions and Flow Control
Objectives

- Distinguish between instance and local variables
- Describe how instance variables are initialized
- Identify and correct a Possible reference before assignment compiler error
- Recognize, describe, and use Java operators
- Distinguish between legal and illegal assignments of primitive types
Objectives

• Identify boolean expressions and their requirements in control constructs

• Recognize assignment compatibility and required casts in fundamental types

• Use if, switch, for, while, and do constructions and the labeled forms of break and continue as flow control structures in a program
Relevance

- What types of variables are useful to programmers?
- Can multiple classes have variables with the same name and, if so, what is their scope?
- What types of control structures are used in other languages? What methods do these languages use to control flow?
Variables and Scope

Local variables are:

- Variables that are defined inside a method and are called *local, automatic, temporary, or stack* variables
- Created when the method is executed and destroyed when the method is exited
- Variables that must be initialized before they are used or compile-time errors will occur
Variable Initialization

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>0</td>
</tr>
<tr>
<td>short</td>
<td>0</td>
</tr>
<tr>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>long</td>
<td>0L</td>
</tr>
<tr>
<td>float</td>
<td>0.0f</td>
</tr>
<tr>
<td>double</td>
<td>0.0d</td>
</tr>
<tr>
<td>char</td>
<td>'\u0000' (NULL)</td>
</tr>
<tr>
<td>boolean</td>
<td>false</td>
</tr>
<tr>
<td>All reference types</td>
<td>null</td>
</tr>
</tbody>
</table>
### Operators

<table>
<thead>
<tr>
<th>Separator</th>
<th>.</th>
<th>[ ]</th>
<th>( )</th>
<th>;</th>
<th>,</th>
</tr>
</thead>
<tbody>
<tr>
<td>R to L</td>
<td>++</td>
<td>--</td>
<td>+</td>
<td>~</td>
<td>!</td>
</tr>
<tr>
<td>L to R</td>
<td>*</td>
<td>/</td>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L to R</td>
<td>+</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L to R</td>
<td>&lt;&lt;</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L to R</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;=</td>
<td>instanceof</td>
</tr>
<tr>
<td>L to R</td>
<td>==</td>
<td>!=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L to R</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L to R</td>
<td>^</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L to R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L to R</td>
<td>&amp;&amp;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L to R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R to L</td>
<td>?:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R to L</td>
<td>*=</td>
<td>/=</td>
<td>%=</td>
<td>+=</td>
<td>-=</td>
</tr>
<tr>
<td>R to L</td>
<td>&gt;&gt;=</td>
<td>&gt;&gt;&gt;=</td>
<td>&amp;=</td>
<td>^=</td>
<td></td>
</tr>
</tbody>
</table>
Logical Expressions

• The Boolean operators supported are:

!  fl– NOT    &  fl– AND
|  – OR         ^  – XOR

• The bitwise operators are:

~  – Complement   &  – AND
|  – OR          ^  – XOR

• The bitwise operators can work with two Boolean operands
Short-Circuit Logical Operators

- The operators are `&&` (*AND*) and `||` (*OR*)
- Operators can be used as follows:

```java
MyDate d = null;
if ((d != null) && (d.day() > 31)) {
    // do something with d
}
```
String Concatenation With +

- The + operator:
  - Performs `String` concatenation
  - Produces a new `String`:

```java
String salutation = "Dr.";
String name = "Pete " + "Seymour";
String title = salutation + name;
```

- One argument must be a `String` object
- Non-strings are converted to `String` objects automatically
Right-Shift Operators $>>$ and $>>>$

- *Arithmetic or signed right shift* ($>>$) is used as follows:
  
  $128 >> 1$ returns $128/2^1 = 64$
  $256 >> 4$ returns $256/2^4 = 16$
  $-256 >> 4$ returns $-256/2^4 = -16$

- The sign bit is copied during the shift.

- *A logical or unsigned right shift operator* ($>>>$) is:
  
  - Used for bit patterns
  - Not copied during the shift
Left-Shift Operator (<<)

• Left-shift works as follows:

128 << 1 returns 128 * 2^1 = 256
16  << 2 returns 16 * 2^2 = 64
Casting

• If information is lost in an assignment, the programmer must confirm the assignment with a typecast.

• The assignment between `short` and `char` requires an explicit cast.

```java
long bigValue = 99L;
int squashed = (int)(bigValue);

long bigval = 6;    // 6 is an int type, OK
int smallval = 99L; // 99L is a long, illegal
```
Promotion and Casting of Expressions

- Variables are automatically promoted to a longer form (such as int to long).

- Expression is *assignment compatible* if the variable type is at least as large (the same number of bits) as the expression type.

```java
double z = 12.414F;  // 12.414F is float, OK
float z1 = 12.414;   // 12.414 is double, illegal
```
Branching Statements

The if, else statements:

```java
if (boolean expression) {
    statement or block;
}
```

```java
if (condition is true) {
    statement or block;
} else {
    statement or block;
}
```
Branching Statements

The if, else statements:

```java
int count;
count = getCount(); // a method defined in the program
if (count < 0) {
    System.out.println("Error: count value is negative.");
} else {
    System.out.println("There will be " + count + " people for lunch today.");
}
```
Branching Statements

The `switch` statement:

The `switch` statement syntax is:

```java
switch (expr1) {
    case constant2:
        statements;
        break;
    case constant3:
        statements;
        break;
    default:
        statements;
        break;
}
```
Branching Statements

The `switch` statement:

```java
int colorNum = 0;

switch (colorNum) {
    case 0:
        setBackground(Color.red);
        break;
    case 1:
        setBackground(Color.green);
        break;
    default:
        setBackground(Color.black);
        break;
}
```
Looping Statements

The `for` statement:

```java
for (init_expr; boolean testexpr; alter_expr) {
    statement or block;
}
```

Example:

```java
for (int i = 0; i < 10; i++) {
    System.out.println("Are you finished yet?");
}
System.out.println("Finally!");
```
Looping Statements

The **while** loop:

```java
} while (boolean) {
    statement or block;
}
```

Example:

```java
int i = 0;

while (i < 10) {
    System.out.println("Are you finished yet?");
    i++;
}
System.out.println("Done");
```
Looping Statements

The do/while statement:

```java
do {
    statement or block;
} while (boolean test);
```

Example:

```java
int i = 0;

do {
    System.out.println("Are you finished yet?");
    i++;
} while (i < 10);
System.out.println("Done");
```
Special Loop Flow Control

- `break [label];`

- `continue [label];`

- `label: statement; // Where statement should // be a loop`
Special Loop Flow Control

The `break` statement:

```java
do {
    statement or block;
    if (condition is true)
        break;
} while (boolean expression);
```
Special Loop Flow Control

The `continue` statement:

```java
do {
    statement or block;
    if (boolean expression)
        continue;
} while (boolean expression);
```
**Special Loop Flow Control**

Using `break` with labels:

```java
loop:
    do {
        do {
            statement;
            statement;
            statement;
            if (boolean expression)
                break loop;
        } while (boolean expression)
        statement;
    } while (boolean expression);
```
Special Loop Flow Control

Using `continue` with labels:

test:
    do {
        statement;
        do {
            statement;
            statement;
        } while (condition is true)
        continue test;
    } while (condition is true)
    statement;
} while (condition is true);
Exercise: Using Expressions

• Exercise objective:
  • Write, compile, and run two arithmetic programs that use identifiers, expressions, and control structures

• Tasks:
  • Use factorial application
  • Create a geometry program
Check Your Progress

• Distinguish between instance and local variables
• Describe how instance variables are initialized
• Identify and correct a Possible reference before assignment compiler error
• Recognize, describe, and use Java operators
• Distinguish between legal and illegal assignments of primitive types
Check Your Progress

• Identify boolean expressions and their requirements in control constructs

• Recognize assignment compatibility and required casts in fundamental types

• Use if, switch, for, while, and do constructions and the labeled forms of break and continue as flow control structures in a program
Think Beyond

• What data types do most programming languages use to group similar data elements together?

• How do you perform the same operation on all elements of a group (for example, a matrix)?

• What data types does the Java programming language use?
Module 4

Arrays
Objectives

• Declare and create arrays of primitive, class, or array types

• Explain why elements of an array are initialized

• Given an array definition, initialize the elements of an array

• Determine the number of elements in an array

• Create a multidimensional array

• Write code to copy array values from one array type to another
Relevance

• What is the purpose of an array?
Declaring Arrays

• Group data objects of the same type

• Declare arrays of primitive or class types

```java
char s[];
Point p[];
```

```java
char [] s;
Point [] p;
```

• Create space for a reference

• Remember an array is an object not memory reserved for primitive types
Creating Arrays

Use the `new` keyword to create an array object.

```java
s = new char[20];
p = new Point[100];

p[0] = new Point();
p[1] = new Point();
.
.
.
```
Initializing Arrays

- Initialize an array element
- Create an array with initial values:

```java
String names[];
names = new String[3];
names[0] = "Georgianna";
names[1] = "Jen";
names[2] = "Simon";

Myclass array[] = {
    new Myclass(),
    new Myclass(),
    new Myclass()
};

Color palette[] = {
    Color.blue,
    Color.red,
    Color.white
};
```
Multi-Dimensional Arrays

- Arrays of arrays:

```java
int twoDim [][] = new int [4][];
twoDim[0] = new int[5];
twoDim[1] = new int[5];

int twoDim [][] = new int [][] [4]; illegal
```
Multi-Dimensional Arrays

• Non-rectangular arrays of arrays:

```
twoDim[0] = new int[2];
twoDim[1] = new int[4];
twoDim[2] = new int[6];
twoDim[3] = new int[8];
```

• Array of four arrays of five integers each:

```
int twoDim[][] = new int[4][5];
```
Array Bounds

All array subscripts begin at 0:

```java
int list[] = new int [10];
for (int i = 0; i < list.length; i++) {
    System.out.println(list[i]);
}
```
Array Resizing

• Cannot resize an array

• Can use the same reference variable to refer to an entirely new array:

```java
int elements[] = new int[6];
elements = new int[10];
```
Copying Arrays

The `System.arraycopy()` method:

1 //original array
2 int elements[] = { 1, 2, 3, 4, 5, 6 };
3
4 // new larger array
5 int hold[] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
6
7 // copy all of the elements array to the hold
8 // array, starting with the 0th index
9 System.arraycopy(elements, 0, hold, 0, elements.length);
Exercise: Using Arrays

- Exercise objectives:
  - Define and initialize an array
  - Write a program that defines, initializes, and uses arrays

- Tasks:
  - Use a basic array
  - Create an array of arrays
  - Create an anagram game
Check Your Progress

- Declare and create arrays of primitive, class, or array types
- Explain why elements of an array are initialized
- Given an array definition, initialize the elements of an array
- Determine the number of elements in an array
- Create a multidimensional array
- Write code to copy array values from one array type to another
Think Beyond

• How can you create a three-dimensional array?

• What is one disadvantage of using arrays?
Module 5

Objects and Classes
Objectives

• Define encapsulation, polymorphism, and inheritance

• Use the access modifiers private and public

• Develop a program segment to create and initialize an object

• Invoke a method on a particular object

• Describe constructor and method overloading

• Describe the purpose of the this reference
Objectives

• Discuss why Java application code is reusable.
• In a Java program, identify the following:
  • The `package` statement
  • The `import` statement
  • Classes, member functions, and variables
  • Constructors
  • Overloaded methods
  • Overridden methods
  • Parent class constructors
Relevance

• The elements of the Java programming language covered so far exist in most languages regardless of whether they are object-oriented.

• What features does the Java programming language possess that make it an object-oriented language?

• What does the term *object-oriented* mean?
Object Fundamentals

• Key features:
  • Encapsulation
  • Polymorphism
  • Inheritance
• Abstraction
• Classes and objects
Classes and Objects

• A class is a template or model.

• An object is created based on that model.

• There is one copy of a class per program, but many objects (*instantiate using the `new` keyword*).

• Methods define the operations for a class.

• Methods must belong to a class.
Classes and Objects

class EmpInfo {
    String name;
    String designation;
    String department;
}

// Create instance
EmpInfo employee = new EmpInfo();

// Initializes the three members
employee.name = "Robert Javaman";
employee.designation = "Manager";
employee.department = "Coffee Shop";

System.out.println(employee.name + " is " +
                    employee.designation + " at " +
                    employee.department);
Classes and Objects

public class MyDate {
    int day, month, year;

    public void tomorrow() {
        // code to increment day
    }
}

MyDate d = new MyDate();
d.tomorrow();
int i = d.day;
Defining Methods

The method declaration takes the following form:

```
<modifiers> <return_type> <name> (  
    [<argument_list>]  
    [throws <exception>] } 
< block >
}
```

Example:

```
public int addDays(int days) {
    < block > // Method code here
}
```
Pass-by-Value

• The Java programming language only passes arguments by value

• When an object instance is passed as an argument to a method, the value of the argument is a reference to the object

• The contents of the object can be changed in the called method, but the object reference is never changed
The this Reference

```java
public class MyDate {
    int day, month, year;

    public void tomorrow() {
        this.day = this.day + 1;
        // wrap around code...
    }
}
```
Data Hiding

public class MyDate {
    private int day, month, year;

    public void tomorrow() {
        this.day = this.day + 1;
        // validate day range
    }
}

public class DateUser {
    public static void main(String args[]) {
        MyDate mydate = new MyDate();
        mydate.day = 21; // illegal!
    }
}
Data Hiding

// Part of MyDate class
public void setDay(int targetDay) {
    if (targetDay > this.daysInMonth()) {
        System.err.println("invalid day "+ targetDay);
    }
    else {
        this.day = targetDay;
    }
}
Encapsulation

• Hides the implementation details of a class
• Forces the user to use an interface to access data
• Makes the code more maintainable
Overloading Method Names

• It can be used as follows:

    public void println(int i)
    public void println(float f)
    public void println(String s)

• Argument lists *must* differ.

• Return types *can* be different.
Constructing and Initializing Objects

• Calling `new Xxxx()` to allocate space for the new object results in:
  • Space for the new object is allocated and initialized to 0 or null.
  • Explicit initialization is performed.
  • A constructor is executed.
Explicit Member Initialization

class Initialized {
    private int x = 5;
    private String name = "Fred";
    private MyDate created = new MyDate();

    // Accessor methods go here
    ...
}

Constructors

- The method name must exactly match the classname.
- There must not be a return type declared for the method.
Constructors

```java
public class Xyz {
    // member variables go here

    public Xyz() {
        // set up the object
    }

    public Xyz(int x) {
        // set up the object with a parameter
    }
}
```
Invoking Overloaded Constructors

```java
public class Employee {
    private String name;
    private int salary;

    public Employee(String n, int s) {
        name = n;
        salary = s;
    }

    public Employee(String n) {
        this(n, 0);
    }

    public Employee() {
        this("Unknown");
    }
}
```
The Default Constructor

- Is in every class
- Enables you to create object instances with `new Xxx()`
- Is invalid if you add a constructor declaration with arguments
The **is a Relationship**

The **Employee** class:

```java
public class Employee {
    String name;
    Date hireDate;
    Date dateOfBirth;
}
```
The is a Relationship

• The Manager class:

```java
public class Manager {
    String name;
    Date hireDate;
    Date dateOfBirth;
    String department;
    Employee subordinates[];
}
```

• Subclassing
The `extends` Keyword

```java
public class Employee {
    String name;
    Date hireDate;
    Date dateOfBirth;
}

public class Manager extends Employee {
    String department;
    Employee subordinates[];
}
```
Single Inheritance

• When a class inherits from only one class, it is called *single inheritance*.

• Single inheritance makes code more reliable.

• *Interfaces* provide the benefits of multiple inheritance without drawbacks.
Single Inheritance

Employee

- attributes
  - name
  - address
  - salary
- methods
  - up_salary
  - promote

- Engineer
- Secretary
- Manager
- Director

Inheritance examples

- attributes
  - bonus
- methods
  - up_bonus

- attributes
  - car allowance
- methods
  - up_allowance
  - promote
Constructors Are Not Inherited

- A subclass inherits all methods and variables from the superclass (parent class).
- A subclass does not inherit the constructor from the superclass.
- Two ways to include a constructor are:
  - Use the default constructor
  - Write one or more explicit constructors
Polymorphism

- **Polymorphism** is the ability to have many different forms; for example, the Manager class has access to methods from Employee class.

- An object has only one form.

- A reference variable has many forms; it can refer to objects of different forms.
Polymorphism

Employee e = new Manager()  //legal

// Illegal attempt to assign Manager member
// variable when object is a parent Employee class
e.department = "Finance";
Heterogeneous Collections

• Collections with a common class are called *homogenous* collections.

• Collections with dissimilar objects are *heterogeneous* collections.
Heterogeneous Collections

- Because a Manager is an Employee:

  ```java
  // In the Employee class
  public TaxRate findTaxRate(Employee e) {
  }
  // Meanwhile, elsewhere in the application class
  Manager m = new Manager();
  TaxRate t = findTaxRate(m);
  ```

- An example of heterogeneous collection is:

  ```java
  Employee[] staff = new Employee[1024];
  staff[0] = new Manager();
  staff[1] = new Employee();
  ```
The **instanceof** Operator

```java
public class Employee extends Object
public class Manager extends Employee
public class Contractor extends Employee

public void method(Employee e) {
    if (e instanceof Manager) {
        // Gets benefits and options
        // along with salary
    } else if (e instanceof Contractor) {
        // Gets hourly rates
    } else {
        // regular employee
    }
}
```
Casting Objects

• Use instanceof to test the type of an object.

• Restore full functionality of an object by casting.

• Check for proper casting using the following guidelines:
  • Casts up hierarchy are done implicitly.
  • Downward casts must be to a subclass and checked by the compiler.
  • The object type is checked at runtime when runtime errors can occur.
Overriding Methods

• A subclass can modify behavior inherited from a parent class.

• A subclass can create a method with different functionality than the parent’s method but with the same:
  • Name
  • Return type
  • Argument list
Overriding Methods

public class Employee {
    String name;
    int salary;
    public String getDetails() {
        return "Name: " + name + "\n" + "Salary: " + salary;
    }
}

public class Manager extends Employee {
    String department;

    public String getDetails() {
        return "Name: " + name + "\n" + "Manager of " + department;
    }
}
Overriding Methods

• Virtual method invocation:

```java
Employee e = new Manager();
e.getDetails();
```

• Compile-time type and runtime type
Rules About Overridden Methods

- Must have a return type that is identical to the method it overrides
- Cannot be less accessible than the method it overrides
- Must throw exceptions that are same type as the method being overridden
Rules About Overridden Methods

```java
public class Parent {
    public void method() {}
}

public class Child extends Parent {
    private void method() {}
}

public class UseBoth {
    public void otherMethod() {
        Parent p1 = new Parent();
        Parent p2 = new Child();
        p1.method();
        p2.method();
    }
}
```
The `super` Keyword

- `super` is used in a class to refer to its superclass.
- `super` is used to refer to the member variables of superclass.
- Superclass behavior is invoked as if the object was part of the superclass.
- Behavior invoked does not have to be in the superclass; it can be further up in the hierarchy.
The super Keyword

```java
class Employee {
    private String name;
    private int salary;
    public String getDetails() {
        return "Name: " + name + "\nSalary: " + salary;
    }
}
class Manager extends Employee {
    private String department;
    public String getDetails() {
        // call parent method
        return super.getDetails() + "\nDepartment: " + department;
    }
}
```
Invoking Parent Class Constructors

- Initialization of objects is structured.
- When an object is initialized, the following sequence of events occur:
  - The memory space is allocated and initialized to "zero" values
  - Explicit initialization is performed for each class in the hierarchy
  - A constructor is called for each class in the hierarchy
Invoking Parent Class Constructors

• In many circumstances, the default constructor is used to initialize the parent object.

    ```java
    public class Employee {
        String name;
        public Employee(String n) {
            name = n;
        }
    }
    ```

    ```java
    public class Manager extends Employee {
        String department;
        public Manager(String s, String d) {
            super(s);
            department = d;
        }
    }
    ```

• If used, you must place `super` or `this` in the first line of the constructor.
Packages

- You must specify package declaration at the beginning of the source file.

- You are permitted only one package declaration per source file.

```java
// Class Employee of the Finance department for the ABC company
package abc.financeDept;

public class Employee {
    ...
}
```

- Package names must be hierarchical and separated by dots.
The `import` Statement

- Tells the compiler where to find classes to use
- Precedes all class declarations:

```java
import abc.financeDept.*;

public class Manager extends Employee {
    String department;
    Employee subordinates[];
}
```
Directory Layout and Packages

- Packages are stored in the directory tree containing the package name.

```java
package abc.financedept

public class Employee {
    ...
}

javac -d . Employee.java
```
Exercise: Using Objects and Classes

• Exercise objective:
  • Write, compile, and run three programs that use the object-oriented concepts of inheritance, constructors, and data hiding by modeling a bank account.

• Tasks:
  • Create a bank account
  • Create several account types
  • Create an online account service
Check Your Progress

- Define *encapsulation, polymorphism, and inheritance*
- Use the access modifiers *private* and *public*
- Develop a program segment to create and initialize an object
- Invoke a method on a particular object
- Describe constructor and method overloading
- Describe the purpose of the *this* reference
Check Your Progress

• Discuss why Java application code is reusable

• In a Java program, identify the following:
  • The package statement
  • The import statement
  • Classes, member functions, and variables
  • Constructors
  • Overloaded methods
  • Overriden methods
  • Parent class constructors
Think Beyond

• Now that you understand objects and classes, how could you put this to use on a project you are working on?
Module 6

Advanced Language Features
Objectives

• Describe static variables, methods, and initializers
• Describe final classes, methods, and variables
• List the access control levels
• Identify deprecated classes and explain how to migrate from JDK™ 1.0 to JDK 1.1 to JDK 1.2
• Describe how to apply collections and reflections
Objectives

• In a Java program, identify:
  • static methods and variables
  • public, private, protected, and default variables
• Use abstract classes and methods
• Explain how and when to use inner classes
• Explain how and when to use interfaces
• Describe the difference between == and equals()
Relevance

- How can you keep a class or method from being subclassed or overridden?
- How can you extend the use of array concepts to objects?
Class (static) Variables

• Are shared among all instances of a class

• Can be marked either as public or as private

• Can be accessed from outside the class if marked as public without an instance of the class

```java
public class Count {
    private static int counter = 0;

    public Count() {
        counter++;  
        serialNumber = counter;
    }
}
```
Class (static) Methods

You can invoke static method without any instance of the class to which it belongs.

```java
public class GeneralFunction {
    public static int addUp(int x, int y) {
        return x + y;
    }
}

public class UseGeneral {
    public void method() {
        int a = 9;
        int b = 10;
        int c = GeneralFunction.addUp(a, b);
        System.out.println("addUp() gives " + c);
    }
}
```
Static Initializers

• A class can contain code in a *static block* that does not exist within a method body.

• Static block code executes only once, when the class is loaded.
Static Initializers

```
public class StaticInitDemo {
    static int i = 5;

    static {
        System.out.println("Static code i= " + i++);
    }
}

class Test {
    public static void main(String args[]) {
        System.out.println("Main code: i=" + StaticInitDemo.i);
    }
}
```
public class Car {
    String color;
    String model;

    // Specific to this instance.
    int serialNumber;

    // Accessible by all instances.
    static int nextSerialNumber = 1;

    public Car (String color, String model) {
        this.color = color;
        this.model = model;
        serialNumber = nextSerialNumber++;
    }

    public void whoAmI() {
        System.out.println(
            "I am a " + color + " " + model + ", serial number = " + serialNumber);
    }

    public static void main (String args[]) {
        Car JanesCar = new Car("Red", "Coupe");
        Car JoesCar = new Car("Blue", "Hatchback");

        JanesCar.whoAmI();
        JoesCar.whoAmI();
    }
}

Access class data:

nextSerialNumber or Car.nextSerialNumber
public class Car2 {
    private String color;
    private String model;

    // Specific to this instance.
    private int serialNumber;

    // Accessible by all instances.
    private static int nextSerialNumber = 1;

    public Car2 (String color, String model) {
        this.color = color;
        this.model = model;
        serialNumber = nextSerialNumber++;
    }

    public void whoAmI() {
        System.out.println(
            "I am a " + color + model + 
            ", serial number = " + serialNumber);
    }

    public static void getNextSerialNum() {
        System.out.println(
            "The next available serial number is " +
            nextSerialNumber);
    }

    public static void main (String args[]) {
        Car2 JanesCar = new Car2("Red", "Coupe");
        Car2 JoesCar = new Car2("Blue", "Hatchback");

        // Use nonstatic method to get instance data
        JanesCar.whoAmI();
        JoesCar.whoAmI();

        // Use static method to get class data
        getNextSerialNum();
    }
}


sun educational services
static methods and data

1 public class Car2 {
2    private String color;
3    private String model;
4
5    // Specific to this instance.
6    private int serialNumber;
7
8    // Accessible by all instances.
9    private static int nextSerialNumber = 1;
10
11   public Car2 (String color, String model) {
12     this.color = color;
13     this.model = model;
14     serialNumber = nextSerialNumber++;
15   }
16
17   public void whoAmI() {
18     System.out.println(
19       "I am a " + color + model + 
20       ", serial number = " + serialNumber);
21   }
22
23   public static void getNextSerialNum() {
24     System.out.println(
25         "The next available serial number is " +
26         nextSerialNumber);
27   }
28
29   public static void main (String args[]) {
30     Car2 JanesCar = new Car2("Red", "Coupe");
31     Car2 JoesCar = new Car2("Blue", "Hatchback");
32
33     // Use nonstatic method to get instance data
34     JanesCar.whoAmI();
35     JoesCar.whoAmI();
36
37     // Use static method to get class data
38     getNextSerialNum();
39   }}                                   // just to fit on page
**Static Methods and Data**

### JanesCar

**instance data**

```java
serialNumber = 1;
```

- JanesCar.whoAmI()
- getNextSerialNum() and main()

### Car

**class (static) data**

```java
nextSerialNumber = 3;
```

- getNextSerialNum() and main()

### JoesCar

**instance data**

```java
serialNumber = 2;
```

- JoesCar.whoAmI()
The `final` Keyword

- You cannot subclass a `final` class.
- You cannot override a `final` method.
- A `final` variable is a constant.
Abstract Classes

• A class that declares the existence of methods but not the implementation is called an abstract class.

• You can declare a class as abstract by marking it with the abstract keyword.

```java
public abstract class Drawing {
    public abstract void drawDot(int x, int y);
    public void drawLine(int x1, int y1,
        int x2, int y2) {
        // draw using the drawDot() method repeatedly.
    }
}
```

• An abstract class can contain member variables and non-abstract methods.
Interfaces

- An interface is a variation on the idea of an abstract class.
- In an interface, all the methods are abstract.
- You can simulate multiple inheritance by implementing such interfaces.
- The syntax is:

```java
public interface Transparency {
    public static final int OPAQUE=1;
    public static final int BITMASK=2;
    public static final int TRANSLUCENT=3;

    public int getTransparency();
}
```
Interfaces

```java
public class MyApplet extends Applet
    implements Runnable, MouseListener {
    "..."
}

interface SayHello {
    void printMessage();
}

class SayHelloImpl implements SayHello {
    void printMessage() {
        System.out.println("Hello");
    }
}
```
Interfaces

• Interfaces are useful for:
  • Declaring methods that one or more classes are expected to implement
  • Determining an object’s programming interface without revealing the actual body of the class
  • Capturing similarities between unrelated classes without forcing a class relationship
## Advanced Access Control

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Same Class</th>
<th>Same Package</th>
<th>Subclass</th>
<th>Universe</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>protected</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>default</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>private</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Deprecation

• Deprecation is the obsoletion of class constructors and method calls.

• Obsolete methods and constructors are replaced by methods with a more standardized naming convention.

• When migrating code, compile the code with the -deprecation flag:

  javac -deprecation MyFile.java
Deprecation

JDK 1.1 code, before deprecation is as follows:

```java
package myutilities;

import java.util.*;
import java.text.*;

public final class DateConverter {
    private static String day_of_the_week [] =
        {"Sunday", "Monday", "Tuesday", "Wednesday",
         "Thursday", "Friday", "Saturday"};

    public static String getDayOfWeek (String theDate){
        int month, day, year;

        StringTokenizer st = new StringTokenizer (theDate, "/");
        month = Integer.parseInt(st.nextToken ());
        day = Integer.parseInt(st.nextToken());
        year = Integer.parseInt(st.nextToken());
        Date d = new Date (year, month, day);
        return (day_of_the_week[d.getDay()]);
    }
}
```
Deprecation

Compiling previous code with the -deprecation flag yields:

```bash
% javac -deprecation DateConverter.java
```

*DateConverter.java:16: Note: The constructor java.util.Date(int,int,int) has been deprecated.*

```java
    Date d = new Date (year, month, day);
```

*DateConverter.java:18: Note: The method int getDay() in class java.util.Date has been deprecated.*

```java
    return (day_of_the_week[d.getDay()]);
```

Note: DateConverter.java uses a deprecated API. Please consult the documentation for a better alternative.

3 warnings
Deprecation

A JDK 1.3 version rewritten is:

```java
package myutilities;

import java.util.*;
import java.text.*;

public final class DateConverter2 {
    private static String day_of_the_week[] = {
        "Sunday", "Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "Saturday"};

    public static String getDayOfWeek (String theDate) {
        Date d = null;
        SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yy");
        try {
            d = sdf.parse (theDate);
        } catch (ParseException e) {
            System.out.println (e);
            e.printStackTrace();
        }

        Calendar c =
            new GregorianCalendar( TimeZone.getTimeZone("EST"),Locale.US);
        c.setTime (d);

        return( day_of_the_week[(c.get(Calendar.DAY_OF_WEEK)-1)];
    }
}
```
The == Operator Versus equals() Method

• The equals() and == methods determine if reference values refer to the same object.

• The equals() method is overridden in classes to return true if the contents and type of two separate objects match.
**toString() Method**

- Converts an object to a `String`
- Converts a primitive type to a `String`, but uses wrapper classes that have the method
- Overrides to provide information about the object in readable format
Inner Classes

- Added to JDK 1.1
- Allow a class definition to be placed inside another class definition
- Group classes that logically belong together
- Have access to their enclosing class’s scope
Properties of Inner Classes

• You can use the class name only within the defined scope, except when used in a qualified name.

The name of the inner class must differ from the enclosing class.

• The inner class can be defined inside a method.

Any variable, either a local variable or a formal parameter, can be accessed by methods within an inner class provided the variable is marked as final.
Properties of Inner Classes

- The inner class can use both class and instance variables of enclosing classes and local variables of enclosing blocks.
- The inner class can be defined as abstract.
- Only inner classes can be declared as private or protected.
- An inner class can act as an interface implemented by another inner class.
Properties of Inner Classes

• Inner classes that are declared `static` automatically become top-level classes.

• Inner classes cannot declare any `static` members; only top-level classes can declare `static` members.

An inner class wanting to use a `static` must declare `static` in the top-level class.
## Wrapper Classes

- **Look at primitive data elements as objects**

<table>
<thead>
<tr>
<th>Primitive Data Type</th>
<th>Wrapper Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>byte</td>
<td>Byte</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>short</td>
<td>Short</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
</tbody>
</table>
Wrapper Classes

```java
define {pInt = 500; Integer wInt = new Integer(pInt); int p2 = wInt.intValue();}
```
Collection API

• A collection (or a container) is a single object representing a group of objects known as its elements.

• Collection classes Vector, Bits, BitSet, Stack, Hashtable, LinkedList, and so on are supported.

• The Collection API contains interfaces that maintain objects as a:
  
  • Collection – A group of objects with no specific ordering
  • Set – A group of objects with no duplication
  • List – A group of ordered objects; duplication is permitted
The Vector Class

The Vector class provides methods for working with dynamic arrays of varied element types.

```
java.lang.Object
  
  + java.util.AbstractCollection
    \  java.util.AbstractList
     \  java.util.Vector
```
Synopsis

- Each vector maintains a capacity and capacityIncrement.
- As elements are added, storage for the vector increases in chunks up to the size of the capacityIncrement variable.
Constructors

- public Vector()
- public Vector(int initialCapacity)
- public Vector(int initialCapacity, int capacityIncrement)
Variables

• protected int capacityIncrement
• protected int elementCount
• protected Object elementData[]
Methods

public final int size()

public final boolean contains(Object elem)

public final int indexOf(Object elem)

public final synchronized Object elementAt(int index)

public final synchronized void setElementAt(Object obj, int index)

public final synchronized void removeElementAt(int index)

public final synchronized void addElement(Object obj)

public final synchronized void insertElementAt(Object obj, int index)
The Vector Class

```java
1 import java.util.*;
2
3 public class MyVector extends Vector {
4   public MyVector() {
5     // storage capacity & capacityIncrement
6     super(1,1);
7   }
8
9   public void addInt(int i) {
10      // addElement requires Object arg
11      addElement(new Integer(i));
12   }
13
14   public void addFloat(float f) {
15      addElement(new Float(f));
16   }
17
18   public void addString(String s) {
19      addElement(s);
20   }
21
22   public void addCharArray(char a[]) {
23      addElement(a);
24   }
```
public void printVector() {
    Object o;
    // compare with capacity()
    int length = size();
    System.out.println("Number of vector elements is "+
              length + " and they are:");
    for (int i = 0; i < length; i++) {
        o = elementAt(i);
        if (o instanceof char[]) {
            // An array's toString() method does not print
            // what we want.
            System.out.println(String.copyValueOf((char[]) o));
        } else {
            System.out.println(o.toString());
        }
    }
}

public static void main(String args[]) {
    MyVector v = new MyVector();
    int digit = 5;
    float real = 3.14F;
    char letters[] = { 'a', 'b', 'c', 'd' };
    String s = new String("High there!");
    v.addInt(digit);
    v.addFloat(real);
    v.addString(s);
    v.addCharArray(letters);
    v.printVector();
}
Reflection API

Can be used to:

• Construct new class instances and new arrays
• Access and modify fields of objects and classes
• Invoke methods on objects and classes
• Access and modify elements of arrays
Reflection API Features

java.lang.Class

java.lang.reflect.Field

java.lang.reflect.Method

java.lang.reflect.Array

java.lang.reflect.Constructor
Reflection API Security Model

- The Java Security Manager controls access to the core Reflection API on a class-by-class basis.

- Standard Java programming language access control is enforced when:
  - A Field is used to get or set a field value
  - A Method is used to invoke a method
  - A Constructor is used to create and initialize a new instance of a class
Exercise: Working With Advanced Language Features

- Exercise objective:
  - Rewrite, compile, and run three programs that use the bank account model and employ advanced object-oriented features, such as inner classes, vector classes, and interfaces

- Tasks:
  - Modify the bank account
  - Use inner classes
  - Add find and delete methods to MyVector class
Check Your Progress

- Describe static variables, methods, and initializers
- Describe final classes, methods, and variables
- List the access control levels
- Identify deprecated classes and explain how to migrate from JDK 1.0 to JDK 1.1 to JDK 1.2
- Describe how to apply collections and reflections
Check Your Progress

• In a Java program, identify:
  • static methods and variables
  • public, private, protected, and default variables

• Use abstract classes and methods

• Explain how and when inner classes are used

• Explain how and when interfaces are used

• Describe the difference between == and equals()
Think Beyond

• What features of the Java programming language are used to deal with runtime error conditions?
Module 7

Exceptions
Course Map

The Java Programming Language Basics
- Getting Started
- Identifiers, Keywords, and Types
- Expressions and Flow Control
- Arrays

Object-Oriented Programming
- Objects and Classes
- Advanced Language Features

Exception Handling
- Exceptions

Developing Graphical User Interfaces
- Building GUIs
- The AWT Event Model
- The AWT Component Library
- Java Foundation Classes

Applets
- Introduction to Java Applets

Multithreading
- Threads

Communications
- Stream I/O and Files
- Networking
Objectives

• Define exceptions
• Use try, catch, and finally statements
• Describe exception categories
• Identify common exceptions
• Develop programs to handle your own exceptions
Relevance

• In most programming languages, how do you resolve runtime errors?
Exceptions

• The Exception class defines mild error conditions that your program encounters.

• Exceptions can occur when:
  • The file you try to open does not exist
  • The network connection is disrupted
  • Operands being manipulated are out of prescribed ranges
  • The class file you are interested in loading is missing

• An error class defines serious error conditions
Exception Example

```java
public class HelloWorld {
    public static void main (String args[]) {
        int i = 0;

        String greetings [] = {
            "Hello world!",
            "No, I mean it!",
            "HELLO WORLD!!"
        };

        while (i < 4) {
            System.out.println (greetings[i]);
            i++;
        }
    }
}
```
try and catch Statements

```java
1 try {
2     // code that might throw a particular exception
3     catch (MyExceptionType e) {
4         // code to execute if a MyExceptionType exception is thrown
5     } catch (Exception e) {
6         // code to execute if a general Exception exception is thrown
7     }
```
Call Stack Mechanism

- If an exception is not handled in the current try/catch block, it is thrown to the caller of that method.
- If the exception gets back to the main method and is not handled there, the program is terminated abnormally.
finally Statement

```java
try {
    startFaucet();
    waterLawn();
} finally {
    stopFaucet();
}
```
Exception Example Revisited

```java
public class HelloWorld2 {
    public static void main (String args[]) {
        int i = 0;

        String greetings [] = {
            "Hello world!",
            "No, I mean it!",
            "HELLO WORLD!!"
        };

        while (i < 4) {
            try {
                System.out.println (greetings[i]);
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Re-setting Index Value");
                i = -1;
            } finally {
                System.out.println("This is always printed");
            }
            i++;
        }
    }
}
```
Exception Categories

```
Throwable
    |--- Error
    |     |--- VirtualMachineError
    |     |     |--- OutOfMemoryError
    |     |     |     |--- StackOverflowError
    |     |--- AWTError
    |--- Exception
    |     |--- RuntimeException
    |     |     |--- ArithmeticException
    |     |     |     |--- NullPointerException
    |     |     |     |     |--- IndexOutOfBoundsException
    |     |--- IOException
    |     |     |--- EOFException
    |     |--- ArithmeticException
    |     |     |--- NullPointeException
    |     |     |     |--- IndexOutOfBoundsException
    |     |--- IOException
    |     |     |--- EOFException
    |     |--- FileNotFoundException
```
Common Exceptions

- ArithmeticException
- NullPointerException
- NegativeArraySizeException
- ArrayIndexOutOfBoundsException
- SecurityException
The Handle or Declare Rule

• Handle the exception by using the try-catch-finally block.

• Declare that the code causes an exception by using the throws clause.
Creating Your Own Exceptions

```java
public class ServerTimedOutException extends Exception {
    private int port;

    public ServerTimedOutException(String reason, int port) {
        super(reason);
        this.port = port;
    }

    // Use Exception class's getMessage() to get the
    // reason the exception was made

    public int getPort() {
        return port;
    }
}
```
Handling User-Defined Exceptions

```java
public void connectMe(String serverName) throws ServerTimedOutException {
    int success;
    int portToConnect = 80;
    success = open(serverName, portToConnect);
    if (success == -1) {
        throw new ServerTimedOutException("Could not connect", 80);
    }
}

public void findServer() {
    try {
        connectMe(defaultServer);
    } catch (ServerTimedOutException e) {
        System.out.println(
            "Server timed out, trying alternative");
        try {
            connectMe(alternativeServer);
        } catch (ServerTimedOutException e1) {
            System.out.println(
                "Error: " + e1.getReason() +
                " connecting to port " + e1.getPort());
        }
    }
}
```
Check Your Progress

• Define exceptions
• Use `try`, `catch`, and `finally` statements
• Describe exception categories
• Identify common exceptions
• Develop programs to handle your own exceptions
Think Beyond

• What features does the Java application environment have that support user interface development?
Module 8

Building GUIs
Objectives

• Describe the AWT package and its components

• Define the terms containers, components, and layout managers, and how they work together to build a graphical user interface (GUI)

• Use layout managers

• Use the FlowLayout, BorderLayout, GridLayout, and CardLayout managers to achieve a desired dynamic layout

• Add components to a container

• Use the Frame and Panel containers appropriately
Objectives

• Describe how complex layouts with nested containers work

• In a Java program, identify the following:
  • Containers
  • The associated layout managers
  • The layout hierarchy of all components
Relevance

• As a platform-independent programming language, how is Java technology used to make the GUI platform independent?
The AWT

- Provides basic GUI components that are used in all Java applets and applications
- Contains classes that can be extended and their properties inherited; classes can also be abstract
- Ensures that every GUI component that is displayed on the screen is a subclass of the abstract class Component
- Has Container, which is an abstract subclass of Component and includes two subclasses:
  - Panel
  - Window
The `java.awt` Package

```
java.lang.Object
  \|-- BorderLayout
  \  \|-- CardLayout
  \  \|-- CheckboxGroup
      \|-- Color
      \|-- Dimension
      \|-- Event
          \|-- Font
          \|-- FlowLayout
          \|-- FontMetrics
          \|-- Graphics
          \|-- GridBagConstraints
          \|-- GridBagLayout
          \|-- GridLayout
              \|-- Image
              \|-- Insets
              \|-- Point
              \|-- Polygon
              \|-- Rectangle
              \|-- Toolkit
              \|-- MenuComponent
      \|-- Component
          \|-- MenuBar
              \|-- Menu
                  \|-- MenuItem
                      \|-- Menu-
                          \|-- PopupMenu
                              \|-- CheckboxMenuItem
          \|-- Applet (java.applet package)
              \|-- Panel
                  \|-- Window
                      \|-- ScrollPane
                          \|-- Dialog
                              \|-- - - FileDialog
                                  \|-- Frame
                                      \|-- TextArea
                                          \|-- TextField
```

Exceptions: `AWTException`

Errors: `AWTError`
Containers

- The two main types of containers are Window and Panel.
  - Windows are objects of java.awt.Window.
  - Panels are objects of java.awt.Panel.
Building Graphical User Interfaces

• The position and size of a component in a container is determined by a layout manager.

• You can control the size or position of components by disabling the layout manager.

You must then use `setLocation()`, `setSize()`, or `setBounds()` on components to locate them in the container.
Frame

- Is a subclass of Window
- Has title and resizing corners
- Inherits from Container and adds components with the add() method
- Can be used to create invisible Frame objects with a title specified by a string.
- Has BorderLayout as the default layout manager
- Uses the setLayout method to change the default layout manager
```java
import java.awt.*;

public class MyFrame extends Frame {
 public MyFrame (String str) {
   super(str);
 }

 public static void main (String args[]) {
   MyFrame fr = new MyFrame("Hello Out There!");
   fr.setSize(500,500);
   fr.setBackground(Color.blue);
   fr.setVisible(true);
 }
```
Panel

- Provides a space for components
- Allows subpanels to have their own layout manager
- Adds components with the add() method
FrameWithPanel.java

```java
import java.awt.*;

public class FrameWithPanel extends Frame {

    // Constructor
    public FrameWithPanel (String str) {
        super(str);
    }

    public static void main (String args[]) {
        FrameWithPanel fr =
            new FrameWithPanel("Frame with Panel");
        Panel pan = new Panel();

        fr.setSize(200,200);
        fr.setBackground(Color.blue);
        fr.setLayout(null);  // Override default layout mgr

        pan.setSize(100,100);
        pan.setBackground(Color.yellow);

        fr.add(pan);
        fr.setVisible(true);
    }
}
```
FrameWithPanel.java
Container Layouts

- FlowLayout
- BorderLayout
- GridLayout
- CardLayout
- GridBagLayout
Default Layout Managers

Component

Container

Window

Frame
Dialog

BorderLayout

Panel
Applet

FlowLayout
A Simple FlowLayout Example

```java
import java.awt.*;

public class ExGui {
    private Frame f;
    private Button b1;
    private Button b2;

    public void go() {
        f = new Frame("GUI example");
        f.setLayout(new FlowLayout());
        b1 = new Button("Press Me");
        b2 = new Button("Don't press Me");
        f.add(b1);
        f.add(b2);
        f.pack();
        f.setVisible(true);
    }

    public static void main(String args[]) {
        ExGui guiWindow = new ExGui();
        guiWindow.go();
    }
}
```
FlowLayout Manager

- Default layout for Panels
- Components added from left to right
- Default alignment is centered
- Uses components’ preferred sizes
- Use the constructor to tune behavior
```java
import java.awt.*;

public class MyFlow {
    private Frame f;
    private Button button1, button2, button3;

    public void go() {
        f = new Frame("Flow Layout");
        f.setLayout(new FlowLayout);
        button1 = new Button("Ok");
        button2 = new Button("Open");
        button3 = new Button("Close");
        f.add(button1);
        f.add(button2);
        f.add(button3);
        f.setSize(100,100);
        f.setVisible(true);
    }

    public static void main(String args[]) {
        MyFlow mflow = new MyFlow();
        mflow.go();
    }
}
```
MyFlow.java

After user or program resizes

Flow Layout

After user or program resizes

Flow Layout
```java
1 import java.awt.*;

3 public class ExGui2 {
4   private Frame f;
5   private Button bn, bs, bw, be, bc;
6
7   public void go() {
8     f = new Frame("Border Layout");
9     bn = new Button("B1");
10    bs = new Button("B2");
11    bw = new Button("B3");
12    be = new Button("B4");
13    bc = new Button("B5");
14
15     f.add(bn, BorderLayout.NORTH);
16     f.add(bs, BorderLayout.SOUTH);
17     f.add(bw, BorderLayout.WEST);
18     f.add(be, BorderLayout.EAST);
19     f.add(bc, BorderLayout.CENTER);
20
21     f.setSize(200,200);
22     f.setVisible(true);
23   }
24
25   public static void main(String args[]) {
26     ExGui2 guiWindow2 = new ExGui2();
27     guiWindow2.go();
28   }
29 }
```
ExGui2.java

After window is resized

After window is resized
GridLayout **Manager**

- Components are added left to right, top to bottom.
- All regions are equally sized.
- The constructor specifies the rows and columns.
GridEx.java

```java
import java.awt.*;

public class GridEx {
    private Frame f;
    private Button b1, b2, b3, b4, b5, b6;

    public void go() {
        f = new Frame("Grid Example");
        f.setLayout (new GridLayout(3,2));

        b1 = new Button("1");
        b2 = new Button("2");
        b3 = new Button("3");
        b4 = new Button("4");
        b5 = new Button("5");
        b6 = new Button("6");

        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.add(b4);
        f.add(b5);
        f.add(b6);

        f.pack();
        f.setVisible(true);
    }

    public static void main(String args[]) {
        GridEx grid = new GridEx();
        grid.go();
    }
}```
GridEx.java

After window is resized

Grid example

1 2
3 4
5 6

Grid Example

1 2
3 4
5 6
CardLayout Manager
CardLayout Manager

```java
    import java.awt.*;
    import java.awt.event.*;

    public class CardTest implements MouseListener {
      private Panel p1, p2, p3, p4, p5;
      private Label lb1, lb2, lb3, lb4, lb5;

      // Declare a CardLayout object to call its methods.
      private CardLayout myCard;
      private Frame f;

      public void go() {
        f = new Frame("Card Test");
        myCard = new CardLayout();
        f.setLayout(myCard);

        // Create the panels that I want
        // to use as cards.
        p1 = new Panel();
        p2 = new Panel();
        p3 = new Panel();
        p4 = new Panel();
        p5 = new Panel();

        // Create a label to attach to each panel, and
        // change the color of each panel, so they are
        // easily distinguishable
        lb1 = new Label("This is the first Panel");
        p1.setBackground(Color.yellow);
        p1.add(lb1);

        lb2 = new Label("This is the second Panel");
        p2.setBackground(Color.green);
        p2.add(lb2);

        lb3 = new Label("This is the third Panel");
        p3.setBackground(Color.magenta);
        p3.add(lb3);
    }
```
lb4 = new Label("This is the fourth Panel");
p4.setBackground(Color.white);
p4.add(lb4);

lb5 = new Label("This is the fifth Panel");
p5.setBackground(Color.cyan);
p5.add(lb5);

// Set up the event handling here.
p1.addMouseListener(this);
p2.addMouseListener(this);
p3.addMouseListener(this);
p4.addMouseListener(this);
p5.addMouseListener(this);

// Add each panel to my CardLayout
f.add(p1, "First");
f.add(p2, "Second");
f.add(p3, "Third");
f.add(p4, "Fourth");
f.add(p5, "Fifth");

// Display the first panel.
myCard.show(f, "First");

f.setSize(200,200);
f.setVisible(true);
}

double get double() {
   return 0.0;
}

String get String(String s) {
   return s;
}

int get int(int i) {
   return i;
}

public void mouseClicked(MouseEvent e) { }
public void mouseEntered(MouseEvent e) { }
public void mouseExited(MouseEvent e) { }
public void mousePressed(MouseEvent e) {
   myCard.next(f);
}
public void mouseReleased(MouseEvent e) { }
```java
79    public static void main (String args[]) {
80        CardTest ct = new CardTest();
81        ct.go();
82    }
83 }
```
GridBagLayout Manager

- Complex layout facilities can be placed in a grid.
- A single component can take its preferred size.
- A component can extend over more than one cell.
ExGui3.java

```java
import java.awt.*;

public class ExGui3 {
  private Frame f;
  private Panel p;
  private Button bw, bc;
  private Button bfile, bhelp;

  public void go() {
    f = new Frame("GUI example 3");
    bw = new Button("West");
    bc = new Button("Work space region");
    f.add(bw, BorderLayout.WEST);
    f.add(bc, BorderLayout.CENTER);
    p = new Panel();
    bfile = new Button("File");
    bhelp = new Button("Help");
    p.add(bfile);
    p.add(bhelp);
    f.add(p, BorderLayout.NORTH);
    f.pack();
    f.setVisible(true);
  }

  public static void main(String args[]) {
    ExGui3 gui = new ExGui3();
    gui.go();
  }
}
```
Output of `ExGui3.java`
Exercise: Building GUIs

• Exercise objective:
  • Develop two graphical user interfaces using the AWT

• Tasks:
  • Create a calculator GUI
  • Create an account GUI
Check Your Progress

• Describe the AWT package and its components

• Define the terms *containers, components, and layout managers*, and how they work together to build a graphical user interface (GUI)

• Use layout managers

• Use the `FlowLayout`, `BorderLayout`, `GridLayout`, and `CardLayout` managers to achieve a desired dynamic layout

• Add components to a container

• Use the `Frame` and `Panel` containers appropriately
Check Your Progress

• Describe how complex layouts with nested containers work

• In a Java program, identify the following:
  • Containers
  • The associated layout managers
  • The layout hierarchy of all components
Think Beyond

- You now know how to display a GUI on the computer screen. What do you need to make the GUI useful?
Module 9

The AWT Event Model
Objectives

- Write code to handle events that occur in a GUI
- Describe the concept of adapter classes, including how and when to use them
- Determine the user action that originated the event from the event object details
- Create the appropriate interface and event handler methods for a variety of event types
Relevance

• What parts are required for a GUI to make it useful?

• How does a graphical program handle a mouse click or any other type of user interaction?
What Is an Event?

- Events – Objects that describe what happened
- Event sources – The generator of an event
- Event handlers – A method that receives an event object, deciphers it, and processes the user’s interaction
JDK 1.0 Event Model Versus Java 2
SDK Event Model

• Hierarchical model (JDK 1.0)
• Delegation model (JDK 1.1 and beyond)
Hierarchical Model (JDK 1.0)

• Is based on containment
Hierarchical Model (JDK 1.0)

• Advantages:
  • Uses object-oriented principles

• Disadvantages:
  • An event can be handled only by the component from which it originated or by one of the containers of the originating component
  • To handle events, you must either subclass the component that receives the event or create a `handleEvent()` method at the base container
Delegation Model

```java
Panel and Frame event handlers

Frame

Panel

Button

Action event

Action handler

actionPerformed (ActionEvent e) {
    ...
}
```
Delegation Model

```java
1 import java.awt.*;
2
3 public class TestButton {
4   public static void main(String args[]) {
5     Frame f = new Frame("Test");
6     Button b = new Button("Press Me!");
7     b.addActionListener(new ButtonHandler());
8     f.add(b, BorderLayout.CENTER);
9     f.pack();
10    f.setVisible(true);
11  }
12 }
```

```java
1 import java.awt.event.*;
2
3 public class ButtonHandler implements ActionListener {
4   public void actionPerformed(ActionEvent e) {
5     System.out.println("Action occurred");
6     System.out.println("Button's label is:" + e.getActionCommand());
7  }
8 }
```

Java Programming Language

Module 9, slide 10 of 24
Delegation Model (JDK 1.1 and Beyond)

- Advantages:
  - Events are not accidentally handled
  - You can create and use filter (adapter) classes to classify event actions
  - There is better distribution of work among the classes

- Disadvantage:
  - You should not combine two event models
import java.awt.*;

public class TestButton {
    public static void main(String args[]) {
        Frame f = new Frame("Test");
        Button b = new Button("Press Me!");
        b.addActionListener(new ButtonHandler());
        f.add(b, BorderLayout.CENTER);
        f.pack();
        f.setVisible(true);
    }
}
The ButtonHandler Class

```java
import java.awt.event.*;

public class ButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Action occurred");
        System.out.println("Button's label is:");
        System.out.println(" + e.getActionCommand());
    }
}
```
Event Categories

java.util.EventObject
  |
java.awt.AWTEvent
  |
java.awt.event
  | ActionEvent
  | AdjustmentEvent
  | ComponentEvent
  | ItemEvent
  | TextEvent
  |
java.beans.beancontext
  | BeanContextEvent
  | ...
  |
java.beans.beancontext
  | ContainerEvent
  | FocusEvent
  | InputEvent
  | WindowEvent
  |
java.beans.beancontext
  | KeyEvent
  | MouseEvent
# Java GUI Behavior

<table>
<thead>
<tr>
<th>Category</th>
<th>Interface Name</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>ActionListener</td>
<td>actionPerformed(ActionEvent)</td>
</tr>
<tr>
<td>Item</td>
<td>ItemListener</td>
<td>itemStateChanged(ItemEvent)</td>
</tr>
<tr>
<td>Mouse motion</td>
<td>MouseMotionListener</td>
<td>mouseDragged(MouseEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mouseMoved(MouseEvent)</td>
</tr>
<tr>
<td>Mouse button</td>
<td>MouseListener</td>
<td>mousePressed(MouseEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mouseReleased(MouseEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mouseEntered(MouseEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mouseExited(MouseEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mouseClicked(MouseEvent)</td>
</tr>
<tr>
<td>Key</td>
<td>KeyListener</td>
<td>keyPressed(KeyEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>keyReleased(KeyEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>keyTyped(KeyEvent)</td>
</tr>
<tr>
<td>Focus</td>
<td>FocusListener</td>
<td>focusGained(FocusEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>focusLost(FocusEvent)</td>
</tr>
<tr>
<td>Adjustment</td>
<td>AdjustmentListener</td>
<td>adjustmentValueChanged(AdjustmentEvent)</td>
</tr>
<tr>
<td>Component</td>
<td>ComponentListener</td>
<td>componentMoved(ComponentEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentHidden(ComponentEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentResized(ComponentEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentShown(ComponentEvent)</td>
</tr>
</tbody>
</table>
### Java GUI Behavior

<table>
<thead>
<tr>
<th>Category</th>
<th>Interface Name</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>WindowListener</td>
<td>windowClosing(WindowEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowOpened(WindowEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowIconified(WindowEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowDeiconified(WindowEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowClosed(WindowEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowActivated(WindowEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowDeactivated(WindowEvent)</td>
</tr>
<tr>
<td>Container</td>
<td>ContainerListener</td>
<td>componentAdded(ContainerEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentRemoved(ContainerEvent)</td>
</tr>
<tr>
<td>Text</td>
<td>TextListener</td>
<td>textValueChanged(TextEvent)</td>
</tr>
</tbody>
</table>
```
import java.awt.*;
import java.awt.event.*;

public class TwoListen
   implements MouseMotionListener,
           MouseListener {
   private Frame f;
   private TextField tf;

   public void go() {
      f = new Frame("Two listeners example");
      f.add(new Label("Click and drag the mouse"),
               BorderLayout.NORTH);
      tf = new TextField(30);
      f.add(tf, BorderLayout.SOUTH);

      f.addMouseMotionListener(this);
      f.addMouseListener(this);
      f.setSize(300, 200);
      f.setVisible(true);
   }

   // These are MouseMotionListener events
   public void mouseDragged (MouseEvent e) {
      String s =
         "Mouse dragging:  X = " + e.getX() +
         " Y = " + e.getY();
      tf.setText(s);
   }

   public void mouseEntered (MouseEvent e) {
      String s = "The mouse entered";
      tf.setText(s);
   }

   public void mouseExited (MouseEvent e) {
      String s = "The mouse has left the building";
      tf.setText(s);
   }
```

Complex Example

40
41   // Unused MouseMotionListener method.
42   // All methods of a listener must be present in the
43   // class even if they are not used.
44   public void mouseMoved (MouseEvent e) { }
45
46   // Unused MouseListener methods.
47   public void mousePressed (MouseEvent e) { }
48   public void mouseClicked (MouseEvent e) { }
49   public void mouseReleased (MouseEvent e) { }
50
51   public static void main(String args[]) {
52     TwoListen two = new TwoListen();
53     two.go();
54   }
Multiple Listeners

- Multiple listeners cause unrelated parts of a program to react to the same event
- All registered listeners call their handlers when the event occurs
Event Adapters

• The listener classes that you define can extend adapter classes and override only the methods that you need.

• For example:

```java
import java.awt.*;
import java.awt.event.*;

public class MouseClickHandler extends MouseAdapter {

    // We just need the mouseClicked handler, so we use
    // an adapter to avoid having to write all the
    // event handler methods

    public void mouseClicked (MouseEvent e) {
        // Do stuff with the mouse click...
    }
}
```
Anonymous Classes

```java
import java.awt.*;
import java.awt.event.*;

public class AnonTest {
  private Frame f;
  private TextField tf;

  public void go() {
    f = new Frame("Anonymous classes example");
    f.add(new Label("Click and drag the mouse"),
      BorderLayout.NORTH);
    tf = new TextField (30);
    f.add (tf, BorderLayout.SOUTH);

    f.addMouseMotionListener(
      new MouseMotionAdapter() {
        public void mouseDragged (MouseEvent e) {
          String s = "Mouse dragging:  X = " + e.getX() +
            " Y = " + e.getY();
          tf.setText (s);
        }
      }); // <- note the closing parenthesis
    f.addMouseListener (new MouseClickHandler());
    f.setSize(300, 200);
    f.setVisible(true);
  }

  public static void main(String args[]) {
    AnonTest obj = new AnonTest();
    obj.go();
  }
}
```
Exercise: Working With Events

• Exercise objective:
  • Write, compile, and run the revised Calculator GUI and Account GUI codes to include event handlers.

• Tasks:
  • Re-create the calculator GUI
  • Re-create the account GUI
Check Your Progress

- Write code to handle events that occur in a GUI
- Describe the concept of adapter classes, including how and when to use them
- Determine the user action that originated the event from the event object details
- Create the appropriate interface and event handler methods for a variety of event types
Think Beyond

You now know how to set up a Java GUI for both graphic output and interactive user input. However, only a few of the components from which GUIs can be built have been described. What other components would be useful in a GUI?
Module 10

The AWT Component Library
Objectives

- Identify key AWT components
- Use AWT components to build user interfaces for real programs
- Control the colors and fonts used by an AWT component
- Use the Java printing mechanism
Relevance

- You now know how to set up a Java GUI for both graphic output and interactive user input. However, only a few of the components from which GUIs can be built have been described. What other components would be useful in a GUI?
Features of the AWT

- AWT components provide mechanisms for controlling the interface appearance, including color and font.
- The AWT also supports printing. (It was added in the JDK 1.1 release.)
Creating a Button

```java
public void actionPerformed(ActionEvent ae) {
    System.out.println("Button press received.");
    System.out.println("Button’s action command is: "+
    ae.getActionCommand());
}
```

```java
b = new Button("Sample");
b.setActionCommand("Action Command Was Here!");
b.addActionListener(this);
f.add(b);
```
Creating a Checkbox

```java
f = new Frame("Sample Checkbox");
one = new Checkbox("One", true);
two = new Checkbox("Two", false);
three = new Checkbox("Three", false);

one.addItemListener(this);
two.addItemListener(this);
three.addItemListener(this);

f.setLayout(new FlowLayout());
f.add(one);
f.add(two);
f.add(three);
```
Creating the `ItemListener` Interface

```java
class Handler implements ItemListener {
    public void itemStateChanged(ItemEvent ev) {
        String state = "deselected";
        if (ev.getStateChange() == ItemEvent.SELECTED){
            state = "selected";
        }
        System.out.println(ev.getItem() + " " + state);
    }
}
```
Creating the `CheckboxGroup` – Radio Buttons

```java
1     f = new Frame("CheckBoxGroup");
2     cbg = new CheckboxGroup();
3     one = new Checkbox("One", cbg, false);
4     two = new Checkbox("Two", cbg, false);
5     three = new Checkbox("Three", cbg, true);
6
7     f.setLayout(new FlowLayout());
8
9     one.addItemListener(this);
10    two.addItemListener(this);
11    three.addItemListener(this);
12
13    f.add(one);
14    f.add(two);
15    f.add(three);
```
Creating a Choice

```java
1   f = new Frame("Sample Choice");
2   choice = new Choice();
3   choice.addItem("First");
4   choice.addItem("Second");
5   choice.addItem("Third");
6   choice.addItemListener(this);
7   f.add(choice, BorderLayout.CENTER);
```
Canvas

- Provides a blank space to draw, write text, or receive keyboard or mouse input
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class MyCanvas extends Canvas implements KeyListener{
    private int index;
    Color colors[] = { Color.red, Color.green, Color.blue };

    public void paint(Graphics g) {
        g.setColor(colors[ index ]);
        g.fillRect(0, 0, getSize().width, getSize().height);
    }

    public void keyTyped(KeyEvent ev) {
        index++;
        if ( index == colors.length ) {
            index = 0;
        }
        repaint();
    }

    // Unused KeyListener methods
    public void keyPressed(KeyEvent ev) { }
    public void keyReleased(KeyEvent ev) { }

    public static void main(String args[])
    {
        Frame f = new Frame("Canvas");
        MyCanvas mc = new MyCanvas();
        mc.setSize(150, 150);
        f.add(mc, BorderLayout.CENTER);
        mc.requestFocus();
        mc.addKeyListener(mc);
        f.pack();
        f.setVisible(true);
    }
Creating a Label

1. Frame f = new Frame("Label");
2. Label lb = new Label("Hello");
3. f.add(lb);
Creating a `TextField`

```java
1   Frame f = new Frame("TextField");
2   TextField tf = new TextField("Single line", 30);
3   tf.addActionListener(this);
4   f.add(tf);
```
Creating a `TextArea`

```java
1     f = new Frame("TextArea");
2     ta = new TextArea("Hello!", 4, 30);
3     f.add(ta, BorderLayout.CENTER);
```
Text Components

- `TextArea` and `TextField` are subclasses
- `TextArea` and `TextField` inherit the default behavior for keystrokes from `TextComponent`
Creating a **List** Component

1. `List lst = new List(4, true);`
2. `lst.add("Hello");`
3. `lst.add("there");`
4. `lst.add("how");`
Creating a Dialog

```java
d = new Dialog(f, "Dialog", true);
d.setLayout(new GridLayout(2,1));
dl = new Label("Hello, I'm a Dialog");
dbl = new Button("OK");
d.add(dl);
d.add(db1);
d.pack();
```

![Dialog Window](image1)

![Dialog Window](image2)
Creating a FileDialog

1. FileDialog d = new FileDialog(parentFrame, "FileDialog");
2. d.setVisible(true);  // block here until OK selected
3. String fname = d.getDirectory() + d.getFile();
Creating a ScrollPane

1. Frame f = new Frame("ScrollPane");
2. Panel p = new Panel();
3. ScrollPane sp = new ScrollPane();
4. p.setLayout(new GridLayout(3, 4));
5. sp.add(p);
6. f.add(sp, BorderLayout.CENTER);
7. f.setSize(100, 100);
8. f.setVisible(true);
Menu

- Must be added to a menu container
- Includes a help menu:
  - `setHelpMenu(Menu)`
Creating a `MenuBar`

```java
1  Frame f = new Frame("MenuBar");
2  MenuBar mb = new MenuBar();
3  f.setMenuBar(mb);
```
Creating a Menu

```java
1     f = new Frame("Menu");
2     mb = new MenuBar();
3     m1 = new Menu("File");
4     m2 = new Menu("Edit");
5     m3 = new Menu("Help");
6     mb.add(m1);
7     mb.add(m2);
8     mb.setHelpMenu(m3);
9     f.setMenuBar(mb);
```
Creating a MenuItem

```java
1     mi1 = new MenuItem("New");
2     mi2 = new MenuItem("Save");
3     mi3 = new MenuItem("Load");
4     mi4 = new MenuItem("Quit");
5     mi1.addActionListener(this);
6     mi2.addActionListener(this);
7     mi3.addActionListener(this);
8     mi4.addActionListener(this);
9     ml.add(mi1);
10    ml.add(mi2);
11    ml.add(mi3);
12    ml.addSeparator();
13    ml.add(mi4);
```
Creating a CheckBoxMenuItem

```java
mb = new MenuBar();
m1 = new Menu("File");
m2 = new Menu("Edit");
m3 = new Menu("Help");
mb.add(m1);
mb.add(m2);
mb.setHelpMenu(m3);
f.setMenuBar(mb);

mi2 = new MenuItem("Save");
mi2.addActionListener(this);
m1.add(mi2);

mi5 = new CheckBoxMenuItem("Persistent");
mi5.addItemListener(this);
ml.add(mi5);
```
Creating a PopupMenu

```java
1  Frame f = new Frame("PopupMenu");
2  Button b = new Button("Press Me");
3  PopupMenu p = new PopupMenu("Popup");
4  MenuItem s = new MenuItem("Save");
5  MenuItem ld = new MenuItem("Load");
6  b.addActionListener(this);
7  f.add(b, BorderLayout.CENTER);
8  p.add(s);
9  p.add(ld);
10  f.add(p);

1  public void actionPerformed(ActionEvent ev) {
2     // display popup at (10,10) relative to b
3     p.show(b, 10, 10);
4  }
```
Controlling Visual Aspects

- Colors:
  - setForeground()
  - setBackground()

Example:

```java
int r = 255;
Color c = new Color(r, 0, 0);
```
Controlling Visual Aspects

• Fonts:
  • You can use the `setFont()` method to specify the font used for displaying text.
  • `Dialog`, `DialogInput`, `Serif`, and `SansSerif` are valid font names.
Printing

• Allow the use of local printer conventions:

```java
1. Frame f = new Frame("Print test");
2. Toolkit t = f.getToolkit();
3. PrintJob job = t.getPrintJob(f, "MyPrintJob", null);
4. Graphics g = job.getGraphics();
```

• Draw on the graphics object

• Send the graphics object to printer

• End the print job

• Obtain a new graphic for each page use:

  ```java
  f.printComponents(g);
  ```
Exercise: Creating a Paint Program Layout

- Exercise objective:
  - Practice creating a more sophisticated GUI application that uses many components

- Tasks:
  - Create a Java application to use classes and objects
  - Investigate reference assignments
Check Your Progress

- Identify key AWT components
- Use AWT components to build user interfaces for real programs
- Control the colors and fonts used by an AWT component
- Use the Java printing mechanism
Think Beyond

• What would make the AWT work better?
Module 11

Java Foundation Classes
Course Map

The Java Programming Language Basics
- Getting Started
- Identifiers, Keywords, and Types
- Expressions and Flow Control
- Arrays

Object-Oriented Programming
- Objects and Classes
- Advanced Language Features

Exception Handling
- Exceptions

Developing Graphical User Interfaces
- Building GUIs
- The AWT Event Model
- The AWT Component Library
- Java Foundation Classes

Applets
- Introduction to Java Applets

Multithreading
- Threads

Communications
- Stream I/O and Files
- Networking
Objectives

• Identify the key features of Java Foundation Classes
• Describe the key features of `com.sun.java.swing` package
• Identify Swing components
• Define `containers` and `components`, and explain how they work together to build a Swing GUI
• Write, compile, and run a basic Swing application
• Use top-level containers, such as `JFrame` and `JApplet` effectively
Relevance

• While the AWT by itself is useful, it is a part of a new set of classes, called Java Foundation Classes (JFC), that, as a whole, take GUIs to a new level. What exactly is JFC and, in particular, what is Swing? What can Swing do that AWT cannot?
Introduction

• Java Foundation Classes (JFC) consists of five APIs:
  • AWT
  • Java 2D
  • Accessibility
  • Drag and Drop
  • Swing
Swing Introduction

• Pluggable look and feel:
  • Application appears to be platform specific
  • There are custom Swing components

• Swing architecture:
  • Built around APIs that implement various parts of AWT
  • Most components do not use platform-specific implementations like AWT
Swing Hierarchy

java.awt.Container

com.sun.java.swing.JComponent

Swing Hierarchy:

- JTextComponent
  - JTextField
  - JTextArea
  - JPasswordField
  - HtmlEditorKit
- AbstractButton
- JLayeredPane
- JLabel
- JComboBox
- JScrollPane
- JToolBar
- JMenuBar
- JMenuItem
- JMenu
- JPanel
- JCheckBox
- JRadioButton
- JCheckBoxMenuItem
- JRadioButtonMenuItem
- JMenu
- JTable
- JSlider
- JRootPane

Diagram showing the Swing hierarchy, with various Swing components and their relationships.
Swing Components

JApplet  JComboBox  JList
Swing Components

- JLabel
- JButton
- JToggleButton
- JOptionPane
A Basic Swing Application
HelloSwing

```java
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.accessibility.*;

public class HelloSwing implements ActionListener {
    private JFrame jFrame;
    private JLabel jLabel;
    private JPanel jPanel;
    private JButton jButton;
    private AccessibleContext accContext;

    private String labelPrefix = "Number of button clicks: ";
    private int numClicks = 0;

    public void go() {
        // Here is how you can set up a particular
        // lookAndFeel. Not necessary for default.
        //
        // try {
        //     UIManager.setLookAndFeel(
        //         UIManager.getLookAndFeel());
        // } catch (UnsupportedLookAndFeelException e) {
        //     System.err.println("Couldn't use the " +
        //         "default look and feel " + e);
        // }

        jFrame = new JFrame("HelloSwing");
        jLabel = new JLabel(labelPrefix + "0");
        jButton = new JButton("I am a Swing button!");

        // Create a shortcut: make ALT-A be equivalent
        // to pressing mouse over button.
        jButton.setMnemonic('i');
        jButton.addActionListener(this);
    }
}
```
HelloSwing

// Add support for accessibility.
accContext = jButton.getAccessibleContext();
accContext.setAccessibleDescription(
    "Pressing this button increments " +
    "the number of button clicks");

// Set up pane.
// Give it a border around the edges.
jPanel = new JPanel();
jPanel.setBorder(
    BorderFactory.createEmptyBorder(30,30,10,30));

// Arrange for compts to be in a single column.
jPanel.setLayout(new GridLayout(0, 1));

// Put compts in pane, not in JFrame directly.
jPanel.add(jButton);
jPanel.add(jLabel);
jFrame.setContentPane(jPanel);

// Set up a WindowListener inner class to handle
// window's quit button.
WindowListener wl = new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
};

jFrame.addWindowListener(wl);
jFrame.pack();
jFrame.setVisible(true);

// Button handling.
public void actionPerformed(ActionEvent e) {
    numClicks++;
    jLabel.setText(labelPrefix + numClicks);
}
78
79   public static void main(String[] args) {
80       HelloSwing helloSwing = new HelloSwing();
81       helloSwing.go();
82   }
83 }
Basic Swing Application

- Importing Swing packages
- Choosing the look and feel:
  - `getLookAndFeel()`
- Setting up a Window container
  - `JFrame` is similar to `Frame`
  - You cannot add components directly to `JFrame`
  - *A content pane* contains all of the `Frame`'s visible components except menu bar
Basic Swing Application

• Setting up Swing components:
  • HelloSwing.java example instantiates four Swing components: JFrame, JButton, JLabel, and JPanel

• Supporting assistive technologies:
  • HelloSwing.java example code supports assistive technologies

```java
1   accContext = jButton.getAccessibleContext();
2   accContext.setAccessibleDescription("Pressing this button increments " + "the number of button clicks.");
```
Building a Swing GUI

• Top-level containers (JFrame, JApplet, JDialog, and JWindow)

• Lightweight components (such as JButton, JPanel, and JMenu)

• Swing components are added to a content pane associated with a top-level container
Building a Swing GUI

```java
import java.awt.*;
import com.sun.java.swing.*;

public class SwingGUI {
    private JFrame topLevel;
    private JPanel jPanel;
    private JTextField jTextField;
    private JList jList;

    private JButton b1;
    private JButton b2;
    private Container contentPane;

    private Object listData[] = {
        new String("First selection"),
        new String("Second selection"),
        new String("Third selection")
    };

    public void go() {
        topLevel = new JFrame("Swing GUI");

        jPanel = new JPanel();
        jTextField = new JTextField(20);
        jList = new JList(listData);

        contentPane = topLevel.getContentPane();
        contentPane.setLayout(new BorderLayout);

        b1 = new JButton("1");
        b2 = new JButton("2");
        contentPane.add(b1, BorderLayout.NORTH);
        contentPane.add(b2, BorderLayout.SOUTH);
    }
}
```
Building a Swing GUI

```java
36     jPanel.setLayout(new FlowLayout());
37     jPanel.add(jTextField);
38     jPanel.add(jList);
39     contentPane.add(jPanel, BorderLayout.CENTER);
40
42     topLevel.pack();
43     topLevel.setVisible(true);
44 }
46     public static void main (String args[]) {
47         SwingGUI swingGUI = new SwingGUI();
48         swingGUI.go();
49     }
50 }
```
The JComponent Class

- Swing components that are subclasses of JComponent
- Borders
- Double buffering
- Tool tips
- Keyboard navigation
- Application-wide pluggable look and feel
Exercise: Creating Swing Applications

• Exercise objective:
  • Write, compile, and run a simple and an advanced Swing GUI program using Swing components

• Tasks:
  • Create a basic Swing application
  • Create a text editor using Swing
Check Your Progress

- Identify the key features of Java Foundation Classes
- Describe the key features of `com.sun.java.swing` package
- Identify Swing components
- Define containers and components, and explain how they work together to build a Swing GUI
- Write, compile, and run a basic Swing application
- Use top-level containers, such as `JFrame` and `JApplet` effectively
Think Beyond

• You now know how to program GUI applications. Suppose you want to run a GUI application using a Web browser. How is this done?
Module 12

Introduction to Java Applets
Objectives

- Differentiate between a standalone application and an applet
- Write an HTML tag to call a Java applet
- Describe the class hierarchy of the applet and AWT classes
- Create the HelloWorld.Java applet
- List the major methods of an applet
- Describe and use the painting model of AWT
Objectives

• Use applet methods to read images and files from URLs
• Use \texttt{<param>} tags to configure applets
Relevance

• What advantages do applets provide?
What Is an Applet?

A Java class that can be:

- Embedded within an HTML page and downloaded and executed by a Web browser
- Loaded using the browser as follows:
  - Load URL
  - Load the HTML document
  - Load applet classes
  - Run the applet
Applet Security Restrictions

• Most browsers prevent the following:
  • Runtime execution of another program
  • File I/O (input/output)
  • Calls to any native methods
  • Attempts to open a socket to any system except the host that provided the applet
Applet Class Hierarchy

java.lang.Object
  ↓
java.awt.Component
  ↓
java.awt.Container
  ↓
java.awt.Window  java.awt.Panel
    ↓          ↓
java.awt.Frame  java.applet.Applet
Key Applet Methods

- init()
- start()
- stop()
- destroy()
- paint()
Applet Display

- Applets are graphical in nature
- The browser environment calls the `paint()` method

```java
import java.awt.*;
import java.applet.*;

public class HelloWorld extends Applet {
    public void paint(Graphics g){
        g.drawString("Hello World!", 25, 25);
    }
}
```
Applet Methods and the Applet Life Cycle

- `init()`
  - Called when the applet is created
  - Can be used to initialize data values
- `start()`
  - Called when the applet becomes visible
- `stop()`
  - Called when the applet becomes invisible
AWT Painting

• `paint(Graphics g)`

• `repaint()`

• `update(Graphics g)`
AWT Painting

AWT thread (waiting)

repaint()

update() - clear area and then call paint()

Exposure

paint()
Applet Display Strategies

• Maintain a model of the display

• Use `paint()` to render the display based only on the model

• Update the model and call `repaint()` to change the display
What Is the appletviewer?

A Java application that:

• Enables you to run applets without using a Web browser

• Loads the HTML file supplied as an argument

  appletviewer HelloWorld.html

• Needs at least the following HTML code:

```html
9  <applet code=HelloWorld.class width=100 height=100>
10  </applet>
```
The applet Tag

<applet
  [archive=archiveList]
  code=appletFile.class
  width=pixels height=pixels
  [codebase=codebaseURL]
  [alt=alternateText]
  [name=appletInstanceName]
  [align=alignment]
  [vspace=pixels] [hspace=pixels]
>
  [<param name=appletAttribute1 value=value>]
  [<param name=appletAttribute2 value=value>]
    ...
  [alternateHTML]
</applet>
Additional Applet Facilities

- `getDocumentBase()` – Returns a URL object that describes the directory of the current browser page
- `getCodeBase()` – Returns a URL object that describes the source directory of the applet class
- `getImage(URL base, String target)` and `getAudioClip(URL base, String target)` – Use the URL object as a starting point
A Simple Image Test

```
// Applet which shows an image of Duke in surfing mode

import java.awt.*;
import java.applet.Applet;

public class HwImage extends Applet {
    Image duke;

    public void init() {
        duke = getImage(getDocumentBase(),
                        "graphics/surferDuke.gif");
    }

    public void paint(Graphics g) {
        g.drawImage(duke, 25, 25, this);
    }
}
```
AudioClip

• Playing a clip

play(URL soundDirectory, String soundFile);

play(URL soundURL);
A Simple Audio Test

```java
// Applet which plays a sound once

import java.awt.Graphics;
import java.applet.Applet;

public class HwAudio extends Applet {
   public void paint(Graphics g) {
      g.drawString("Audio Test", 25, 25);
      play(getCodeBase(), "sounds/cuckoo.au");
   }
}
```
Looping an AudioClip

• Loading an AudioClip
• Playing an AudioClip
• Stopping an AudioClip
A Simple Looping Test

```java
// Applet which continuously repeats a sound
import java.awt.Graphics;
import java.applet.*;

public class HwLoop extends Applet {
    AudioClip sound;
    
    public void init() {
        sound = getAudioClip(getCodeBase(), "sounds/cuckoo.au");
    }

    public void paint(Graphics g) {
        g.drawString("Audio Test", 25, 25);
    }

    public void start() {
        sound.loop();
    }

    public void stop() {
        sound.stop();
    }
}
```
Mouse Input

- mouseClicked – The mouse has been clicked (mouse button pressed and then released in one motion)
- mouseEntered – The mouse cursor enters a component
- mouseExited – The mouse cursor leaves a component
- mousePressed – The mouse button is pressed down
- mouseReleased – The mouse button is later released
A Simple Mouse Test

1 // This applet is HelloWorld extended to watch for mouse
2 // input. "Hello World!" is reprinted at the location of
3 // the mouse press.
4
5 import java.awt.Graphics;
6 import java.awt.event.*;
7 import java.applet.Applet;
8
9 public class HwMouse
10     extends Applet
11     implements MouseListener {
12
13   private int mouseX = 25;
14   private int mouseY = 25;
15
16   // Register this applet instance to catch
17   // MouseListener events.
18   public void init() {
19     addMouseListener(this);
20   }
21
22   public void paint(Graphics g) {
23     g.drawString("Hello World!", mouseX, mouseY);
24   }
25
26   // Process the mousePressed MouseListener event
27   public void mousePressed(MouseEvent evt) {
28     mouseX = evt.getX();
29     mouseY = evt.getY();
30     repaint();
31   }
32
33   // We are not using the other mouse events.
34   public void mouseClicked(MouseEvent e) { }
35   public void mouseEntered(MouseEvent e) { }
36   public void mouseExited(MouseEvent e) { }
37   public void mouseReleased(MouseEvent e) { }
38 }
Reading Parameters

• Applet code

```html
<html>
<applet code=DrawAny.class width=200 height=200>
<param name=image value="graphics/duke.gif">
</applet>
</html>
```

• Program code

```java
import java.awt.*;
import java.applet.*;

public class DrawAny extends Applet {
  Image im;

  public void init() {
    String imageName = getParameter("image");
    if (imageName == null) {
      System.out.println("Error: Cannot find image");
      System.exit(0);
    }
    im = getImage(getDocumentBase(), imageName);
  }

  public void paint(Graphics g) {
    g.drawImage(im, 0, 0, this);
  }
}
```
Dual Purpose Code Sample

```java
// Applet/Application which shows an image of
// Duke in surfing mode

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class AppletApp extends Applet {
    Date date;

    public void init() {
        date = new Date();
    }

    public void paint (Graphics g) {
        g.drawString("This Java program started at", 25, 25);
        g.drawString(date.toString(), 25, 60);
    }

    public static void main (String args[]) {
        // Create a Frame to house the applet
        Frame frame = new Frame("Application");

        // Create an instance of the class (applet)
        AppletApp app = new AppletApp();

        // Add it to the center of the frame
        frame.add(app, BorderLayout.CENTER);
        frame.setSize (250, 150);
    }
}
```
34     // Register the AppletApp class as the
35     // listener for a Window Destroy event
36     frame.addWindowListener (new WindowAdapter() {
37       public void windowClosing (WindowEvent e) {
38         System.exit(0);
39       }
40     });
41
42     // Call the applet methods
43     app.init();
44     app.start();
45     frame.setVisible(true);  // Invokes paint()
Exercise: Creating Applets

- Exercise objective:
  - Become familiar with programming Java applets

- Tasks:
  - Write an applet
  - Create concentric squares
  - Create a rollover applet
Check Your Progress

- Differentiate between a standalone application and an applet
- Write an HTML tag to call a Java applet
- Describe the class hierarchy of the applet and AWT classes
- Create the `HelloWorld.Java` applet
- List the major methods of an applet
- Describe and use the painting model of AWT
- Use applet methods to read images and files from URLs
Check Your Progress

- Use `<param>` tags to configure applets
- Use the URL object to fetch sounds and images into your applet
- Handle various mouse events within the applet
- Pass parameters to an applet from an HTML file using the `<param>` tags
Think Beyond

• How can you use applets on your company’s Web page to improve the overall presentation?
Module 13

Threads
Objectives

• Define a thread

• Create separate threads in a Java program, controlling the code and data that are used by that thread

• Control the execution of a thread and write platform-independent code with threads

• Describe the difficulties that might arise when multiple threads share data

• Use `wait()` and `notify()` to communicate between threads
Objectives

• Use synchronized to protect data from corruption

• Explain why suspend(), resume(), and stop() methods have been deprecated in JDK 1.2
Relevance

- How do you get programs to perform multiple tasks?
Threads

- What are threads?
- Virtual CPU
Three Parts of a Thread

- CPU
- Code
- Data
Creating the Thread

```java
1  public class ThreadTest {
2      public static void main(String args[]) {
3          Xyz r = new Xyz();
4          Thread t = new Thread(r);
5          t.start();
6      }
7  }
8
9  class Xyz implements Runnable {
10     int i;
11
12     public void run() {
13         i = 0;
14         while (true) {
15             System.out.println("Hello " + i++);
16             if (i == 50) {
17                 break;
18             }
19         }
20     }
21  }
22 }
```
Creating the Thread

• Multithreaded programming:
  • Multiple threads from the same `Runnable` instance
  • Threads share the same data and code
Starting the Thread

- Using the `start()` method
- Placing the thread in runnable state
Thread Scheduling

- **New**
  - start()

- **Runnable**
  - Scheduler
  - sleep() timeout
  - thread join()s
  - interrupt()

- **Running**
  - run() completes
  - wait()
  - Lock available
  - notify()
  - interrupt()

- **Dead**
  - sleep() or
  - join()

- **Blocked in object's wait() pool**

- **Blocked in object's lock pool**
Thread Scheduling

```java
public class Xyz implements Runnable {
    public void run() {
        while (true) {
            // do lots of interesting stuff
            // Give other threads a chance
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                // This thread's sleep was interrupted
                // by another thread
            }
        }
    }
}
```
public class Xyz implements Runnable {
    private boolean timeToQuit=false;

    public void run() {
        while(! timeToQuit) {
            ...
        }
        // clean up before run() ends
    }

    public void stopRunning() {
        timeToQuit=true;
    }
}

public class ControlThread {
    private Runnable r = new Xyz();
    private Thread t = new Thread(r);

    public void startThread() {
        t.start();
    }

    public void stopThread() {
        // use specific instance of Xyz
        r.stopRunning();
    }
}
Basic Control of Threads

• Testing threads:
  • isAlive()

• Putting threads on hold:
  • sleep()
  • join()
Putting Threads on Hold

```java
1  public class Xyz implements Runnable {
2    ...
3    public void run() {
4        while (running) {
5            // do your task
6             try {
7                Thread.sleep((int)(Math.rando() * 100));
8            } catch (InterruptedException e) {
9                // somebody woke me up
10            }
11            ...
12        }
13    }
14  }
15
16  public class TTest {
17      public static void main(String args[]) {
18          Runnable r = new Xyz();
19          Thread t1 = new Thread(r);
20          t1.start();
21      }
22  }
```
Putting Threads on Hold

```java
public void doTask() {
    TimerThread tt = new TimerThread(100);
    tt.start();
    ...
    // Do stuff in parallel with the other thread for
    // a while
    ...
    // Wait here for the timer thread to finish
    try {
        tt.join();
    } catch (InterruptedException e) {
        // tt came back early
    }
    // Now continue in this thread
    ...
}
```
Extending the Thread Class

```java
public class MyThread extends Thread {
    public void run() {
        while (running) {
            // do lots of interesting stuff
            try {
                sleep(100);
            } catch (InterruptedException e) {
                // sleep interrupted
            }
        }
    }

    public static void main(String args[]) {
        Thread t = new MyThread();
        t.start();
    }
}
```
Selecting a Way to Create Threads

• Implementing Runnable:
  • Better object-oriented design
  • Single inheritance
  • Consistency

• Extending Thread:
  • Simpler code
Using the `synchronized` Keyword

```java
public class MyStack {
    int idx = 0;
    char [] data = new char[6];

    public void push(char c) {
        data[idx] = c;
        idx++;
    }

    public char pop() {
        idx--;  
        return data[idx];
    }

    public char pop() {
        idx--;
        return data[idx];
    }
}
```
The Object Lock Flag

- Every object has a flag that can be thought of as a "lock flag"
- `synchronized` allows interaction with the lock flag

```java
public void push(char c) {
    synchronized (this) {
        data[idx] = c;
        idx++;
    }
}
```
The Object Lock Flag

Object `this`

- Lock flag missing

Thread, trying to execute

```java
public char pop() {
    synchronized (this) {
        idx--;
        return data[idx];
    }
}
```

Waiting for

- object lock

Code or behavior

Data or state
Releasing the Lock Flag

• Released when the thread passes the end of the `synchronized()` code block

• Automatically released when a break or exception is thrown by the `synchronized()` code block
synchronized – Putting It Together

• *All* access to delicate data should be synchronized.

• Delicate data protected by `synchronized` should be private.
synchronized – Putting It Together

• The following two code segments are equivalent:

```java
public void push(char c) {
    synchronized(this) {
        :
        :
        :}
}
```

```java
public synchronized void push(char c) {
    :
    :
    :}
```
Deadlock

- Is two threads, each waiting for a lock from the other
- Is not detected or avoided
- Can be avoided by:
  - Deciding on the order to obtain locks
  - Adhering to this order throughout
  - Releasing locks in reverse order
Thread Interaction – \texttt{wait()} and \texttt{notify()}

- Scenario:
  - Consider yourself and a cab driver as two threads
- The problem:
  - How to determine when you are at your destination
- The solution:
  - You notify the cabbie of your destination and relax
  - Cabbie drives and notifies you upon arrival at your destination
Thread Interaction

- `wait()` and `notify()`

- The pools:
  - Wait pool
  - Lock pool
Monitor Model for Synchronization

- Leave shared data in a consistent state
- Ensure programs cannot deadlock
- Do not put threads expecting different notifications in the same wait pool
Producer

```java
public void run() {
    char c;

    for (int i = 0; i < 200; i++) {
        c = (char)(Math.random() * 26 + 'A');
        theStack.push(c);
        System.out.println("Producer" + num + ": " + c);
        try {
            Thread.sleep((int)(Math.random() * 300));
        } catch (InterruptedException e) {
            // ignore it
        }
    }
}
```
Consumer

```java
public void run() {
    char c;
    for (int i = 0; i < 200; i++) {
        c = theStack.pop();
        System.out.println("Consumer" + num + ": " + c);
        try {
            Thread.sleep((int)(Math.random() * 300));
        } catch (InterruptedException e) { }
    }
}
```
SyncStack Class

13  public class SyncStack {
14   private Vector buffer = new Vector(400, 200);
15  
16   public synchronized char pop() {
17   }
18  
19   public synchronized void push(char c) {
20   }
pop() Method

```java
1   public synchronized char pop() {
2       char c;
3       while (buffer.size() == 0) {
4           try {
5               this.wait();
6           } catch (InterruptedException e) {
7               // ignore it...
8           }
9       }
10      c = ((Character)buffer.remove(buffer.size()-1)).
11         charValue();
12      return c;
13   }
```
push() Method

```
1   public synchronized void push(char c) {
2       this.notify();
3       Character charObj = new Character(c);
4       buffer.addElement(charObj);
5   }
```
SyncTest.java

```java
1  package mod13;
2  
3  public class SyncTest {
4  
5     public static void main(String[] args) {
6     
7         SyncStack stack = new SyncStack();
8     
9         Producer p1 = new Producer(stack);
10        Thread prodT1 = new Thread (p1);
11        prodT1.start();
12     
13         Producer p2 = new Producer(stack);
14        Thread prodT2 = new Thread (p2);
15        prodT2.start();
16     
17         Consumer c1 = new Consumer(stack);
18        Thread consT1 = new Thread (c1);
19        consT1.start();
20     
21         Consumer c2 = new Consumer(stack);
22        Thread consT2 = new Thread (c2);
23        consT2.start();
24     }
25 }
```
1 package mod13;

2
3 public class Producer implements Runnable {
4   private SyncStack theStack;
5   private int num;
6   private static int counter = 1;
7
8   public Producer (SyncStack s) {
9     theStack = s;
10     num = counter++;
11   }
12
13   public void run() {
14     char c;
15     for (int i = 0; i < 200; i++) {
16       c = (char)(Math.random() * 26 + 'A');
17       theStack.push(c);
18       System.out.println("Producer" + num + ": " + c);
19       try {
20         Thread.sleep((int)(Math.random() * 300));
21       } catch (InterruptedException e) {
22         // ignore it
23       }
24     }
25   }
26 }
package mod13;

public class Consumer implements Runnable {
    private SyncStack theStack;
    private int num;
    private static int counter = 1;

    public Consumer (SyncStack s) {
        theStack = s;
        num = counter++;
    }

    public void run() {
        char c;
        for (int i = 0; i < 200; i++) {
            c = theStack.pop();
            System.out.println("Consumer" + num + ": " + c);
            try {
                Thread.sleep((int)(Math.random() * 300));
            } catch (InterruptedException e) { }
        }
    }
}
package mod13;

import java.util.Vector;

public class SyncStack {
    private Vector buffer = new Vector(400, 200);

    public synchronized char pop() {
        char c;
        while (buffer.size() == 0) {
            try {
                this.wait();
            } catch (InterruptedException e) {
                // ignore it...
            }
            // ignore it...
        }
        c = ((Character) buffer.remove(buffer.size()-1)).
            charValue();
        return c;
    }

    public synchronized void push(char c) {
        this.notify();
        Character charObj = new Character(c);
        buffer.addElement(charObj);
    }
}
The `suspend()` and `resume()` Methods

- Have been deprecated in JDK 1.2
- Should be replaced with `wait()` and `notify()`
The `stop()` Method

- Releases the lock before it terminates
- Can leave shared data in an inconsistent state
- Should be replaced with `wait()` and `notify()`
- Should create long-lived threads
Proper Thread Control

public class ControlledThread extends Thread {
    static final int SUSP = 1;
    static final int STOP = 2;
    static final int RUN = 0;
    private int state = RUN;

    public synchronized void setState(int s) {
        state = s;
        if ( s == RUN ) {
            notify();
        }
    }

    public synchronized boolean checkState() {
        while ( state == SUSP ) {
            try {
                wait();
            } catch (InterruptedException e) {
                // ignore
            }
        }
        if ( state == STOP ) {
            return false;
        }
        return true;
    }

    public void run() {
        while ( true ) {
            // doSomething();
            // Be sure shared data is in consistent state in
            // case the thread is waited or marked for exiting
            // from run()
            if ( !checkState() ) {
                break;
            }
        }
    }
}
Exercise: Using Multithreaded Programming

• Exercise objectives:
  • Become familiar with the concepts of multithreading by writing some multithreaded programs
  • Create a multithreaded applet

• Tasks:
  • Create three threads
  • Incorporate animation
Check Your Progress

• Define a thread

• Create separate threads in a Java program, controlling the code and data that are used by that thread

• Control the execution of a thread and write platform-independent code with threads

• Describe the difficulties that might arise when multiple threads share data

• Use keyword synchronized to protect data from corruption

• Use wait() and notify() to communicate between threads
Check Your Progress

• Use synchronized to protect data from corruption

• Explain why suspend(), resume(), and stop() methods have been deprecated in JDK 1.2
Think Beyond

- Do you have applications that could benefit from being multithreaded?
Module 14

Stream I/O and Files
Objectives

- Describe and use the streams philosophy of the java.io package
- Construct file and filter streams, and use them appropriately
- Distinguish readers and writers from streams, and select appropriately between them
- Examine and manipulate files and directories
- Read, write, and update text and data files
- Use the Serialization interface to persist the state of objects
Relevance

• What mechanisms are in place within the Java programming language to read and write from files?
Stream I/O

• A stream is either a source of bytes or a destination for bytes.

• The two basic types of streams are:
  • Input stream
  • Output stream

• Node streams read from or write to a specific place.

• Filter streams use node streams as input or output.
Stream Fundamentals

Node InputStream (for example, from file)  \rightarrow \text{FilterInputStream}

"Flow" of bytes downstream

read()
InputStream Methods

• The three basic read() methods:
  
  • int read()
  • int read(byte[])
  • int read(byte[], int, int)

• The other methods:
  
  • void close()
  • int available()
  • skip(long)
  • boolean markSupported()
  • void mark(int)
  • void reset()
OutputStream Methods

• The three basic write() methods:
  - void write(int)
  - void write(byte[])
  - void write(byte[], int, int)

• The other methods:
  - void close()
  - void flush()
Basic Stream Classes

InputStream
  SequenceInputStream
  PipedInputStream
  FilterInputStream
  ByteArrayInputStream
  DataInputStream
  ObjectInputStream
  PushbackInputStream
  BufferedInputStream
  FileInputStream
  ByteArrayOutputStream
Basic Stream Classes

- FileInputStream and FileOutputStream
- BufferedInputStream and BufferOutputStream
- DataInputStream and DataOutputStream
- PipedInputStream and PipedOutputStream
URL Input Streams

```java
java.net.URL imageSource;

try {
    imageSource = new URL("http://mysite.com/~info");
} catch (MalformedURLException e) {
    // ignore
}

images[0] = getImage(imageSource, "Duke/T1.gif");
```
Opening an Input Stream

1 InputStream is = null;
2 String datafile = new String("Data/data.1-96");
3 byte buffer[] = new byte[24];
4 try {
5   // new URL throws a MalformedURLException
6   // URL.openStream() throws an IOException
7   is = (new URL(getDocumentBase(), datafile)).openStream();
8 } catch (Exception e) {
9    // ignore
10 }

Now you can use it to read information, just as with a FileInputStream object:

11 try {
12   is.read(buffer, 0, buffer.length);
13 } catch (IOException e1) {
14    // ignore
15 }
Readers and Writers

- The Java programming language uses *Unicode* to represent strings and characters.

- `InputStreamReader` and `OutputStreamWriter` convert Unicode to platform-specific code.

- Chain `BufferedReader` and `BufferedWriter` to `InputStreamReader` and `OutputStreamWriter` for efficiency.
Reading String Input

```java
import java.io.*;

public class CharInput {

    public static void main (String args[]) throws java.io.IOException {
        String s;
        InputStreamReader ir;
        BufferedReader in;

        ir = new InputStreamReader(System.in);
        in = new BufferedReader(ir);

        while ((s = in.readLine()) != null) {
            System.out.println("Read: "+ s);
        }
    }
}
```
Creating a New File Object

- File myFile;

- myFile = new File("mymotd");

- myFile = new File("/", "mymotd");

- // more useful if the directory or filename // is a variable
  File myDir = new File("/");
  myFile = new File(myDir, "mymotd");
File Tests and Utilities

• File names:

  String getName()
  String getPath()
  String getAbsolutePath()
  String getParent()
  boolean renameTo(File newName)

• File tests:

  boolean exists()
  boolean canWrite()
  boolean canRead()
  boolean isFile()
  boolean isDirectory()
  boolean isAbsolute();
File Tests and Utilities

- General file information and utilities:

  ```java
  long lastModified()
  long length()
  boolean delete()
  ```

- Directory utilities:

  ```java
  boolean mkdir()
  String[] list()
  ```
Creating a Random Access File

• With the file name:

```java
myRAFile = new RandomAccessFile(
    String name, String mode);
```

• With a File object:

```java
myRAFile = new RandomAccessFile(
    File file, String mode);
```
Random Access Files

- long getFilePointer()
- void seek(long pos)
- long length()
Serialization

• Saving an object to permanent storage is called persistence.

• Only the object’s data are serialized.

• Data marked with the transient keyword are not serialized.

```java
1 public class MyClass implements Serializable {
2     public transient Thread myThread;
3     private String customerID;
4     private int total;
5 }
```

```java
1 public class MyClass implements Serializable {
2     public transient Thread myThread;
3     private transient String customerID;
4     private int total;
5 }
```
import java.io.*;
import java.util.Date;

public class SerializeDate {

    SerializeDate() {
        Date d = new Date();

        try {
            FileOutputStream f =
            new FileOutputStream("date.ser");
            ObjectOutputStream s =
                new ObjectOutputStream(f);
            s.writeObject(d);
            s.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String args[]) {
        new SerializeDate();
    }
}
Reading an Object From a File Stream

```java
import java.io.*;
import java.util.Date;

public class UnSerializeDate {

    UnSerializeDate () {
        Date d = null;

        try {
            FileInputStream f =
                new FileInputStream("date.ser");
            ObjectInputStream s =
                new ObjectInputStream (f);
            d = (Date) s.readObject () ;
            s.close ();
        } catch (Exception e) {
            e.printStackTrace ;
        }

        System.out.println(
                "Unserialized Date object from date.ser");
        System.out.println("Date: "+d);
    }

    public static void main (String args[]) {
        new UnSerializeDate();
    }
}
```
Exercise: Getting Acquainted With I/O

• Exercise objective:
  • Become familiar with stream I/O by writing programs that perform I/O to files

• Tasks:
  • Open a file
  • Create a simple database program
  • Use persistence
Check Your Progress

- Describe and use streams philosophy of the java.io package
- Construct file and filter streams, and use them appropriately
- Distinguish readers and writers from streams, and select appropriately between them
- Examine and manipulate files and directories
- Read, write, and update text and data files
- Use the Serialization interface to persist the state of objects
Think Beyond

- Do you have applications that require file I/O?
Module 15

Networking
Objectives

• Create a minimal Transmission Control Protocol/Internet Protocol (TCP/IP) server and a minimal TCP/IP client:
  • ServerSocket
  • Socket

• Create a minimal User Datagram Protocol (UDP) server and a minimal UDP client:
  • DatagramSocket
  • DatagramPacket
Relevance

• How can a communication link between a client machine and a server on the network be established?
Networking

- Sockets:
  - Sockets hold two streams

- Setting up the connection:
  - Set up is similar to a telephone system
Networking With Java Technology

• Addressing the connection:
  • Address or name of remote machine
  • Port number to identify purpose

• Port numbers:
  • Range from 0–65535
Java Networking Model

Server

ServerSocket (port #)
ServerSocket.accept ()

Socket (host, port#)
(Socket ()

OutputStream

InputStream

Socket.close ()

Register with this service
Wait for a connection

Client

Socket (host, port#)
(Attempt to connect)

OutputStream

InputStream

Socket.close ()
import java.net.*;
import java.io.*;

public class SimpleServer {
    public static void main(String args[]) {
        ServerSocket s = null;
        Socket s1;
        String sendString = "Hello Net World!";
        int slength = sendString.length();
        OutputStream s1out;
        DataOutputStream dos;

        // Register your service on port 5432
        try {
            s = new ServerSocket(5432);
        } catch (IOException e) {
            // ignore
        } // ignore
    }
}

20     // Run the listen/accept loop forever
21     while (true) {
22       try {
23         // Wait here and listen for a connection
24         s1=s.accept();
25
26         // Get a communication stream associated with
27         // the socket
28         slout = s1.getOutputStream();
29         dos = new DataOutputStream (slout);
30
31         // Send your string!
32         // (UTF provides machine independence)
33         dos.writeUTF(sendString);
34
35         // Close the connection, but not the server socket
36         dos.close();
37         slout.close();
38         s1.close();
39       } catch (IOException e) { 
40         // ignore
41       }
42     }
43   }
44 }

import java.net.*;
import java.io.*;

public class SimpleClient {
    public static void main(String args[]) throws IOException {
        int c;
        Socket s1;
        InputStream s1In;
        DataInputStream dis;

        // Open your connection to a server, at port 5432
        // localhost used here
        s1 = new Socket("127.0.0.1",5432);

        // Get an input file handle from the socket and
        // read the input
        s1In = s1.getInputStream();
        dis = new DataInputStream(s1In);

        String st = new String (dis.readUTF());
        System.out.println(st);

        // When done, just close the connection and exit
        dis.close();
        s1In.close();
        s1.close();
    }
}
UDP Sockets

• Are used for connection-less protocol

• Messages are not guaranteed

• Are supported in Java technology through the DatagramSocket and DatagramPacket classes
The DatagramPacket

DatagramPacket has two constructors: one for receiving data and one for sending data.

- DatagramPacket(
    byte [] recvBuf, int readLength)

- DatagramPacket(
    byte [] sendBuf, int sendLength,
    InetAddress iaddr, int iport)
The **DatagramSocket**

DatagramSocket has three constructors:

- DatagramSocket()
- DatagramSocket(int port)
- DatagramSocket(int port, InetAddress iaddr)
import java.io.*;
import java.net.*;
import java.util.*;

public class UdpServer{

    //This method retrieves the current time on the server
    public byte[] getTime(){
        Date d= new Date();
        return d.toString().getBytes();
    }

    // Main server loop.
    public void go() throws IOException {
        DatagramSocket datagramSocket;
        // Datagram packet from the client
        DatagramPacket inDataPacket;
        // Datagram packet to the client
        DatagramPacket outDataPacket;
        // Client return address
        InetAddress clientAddress;
        // Client return port
        int clientPort;
        // Incoming data buffer. Ignored.
        byte[] msg= new byte[10];
        // Stores retrieved time
        byte[] time;

        // Allocate a socket to man port 8000 for requests.
        datagramSocket = new DatagramSocket(8000);
        System.out.println("UDP server active on port 8000");
    }
}
34     // Loop forever
35     while(true) {
36
37     // Set up receiver packet. Data will be ignored.
38     inDataPacket = new DatagramPacket(msg, msg.length);
39
40     // Get the message.
41     datagramSocket.receive(inDataPacket);
42
43     // Retrieve return address information, including
44     // InetAddress and port from the datagram packet
45     // just received.
46
47     clientAddress = inDataPacket.getAddress();
48     clientPort = inDataPacket.getPort();
49
50     // Get the current time.
51     time = getTime();
52
53     // set up a datagram to be sent to the client using the
54     // current time, the client address and port
55     outDataPacket =
56           new DatagramPacket(
57               time, time.length, clientAddress, clientPort);
58
59     // finally send the packet
60     datagramSocket.send(outDataPacket);
61   }
62 }
public static void main(String args[]) {
    UdpServer udpServer = new UdpServer();

    try {
        udpServer.go();
    } catch (IOException e) {
        System.out.println(
            "IOException occurred with socket.");
        System.out.println(e);
        System.exit(1);
    }
}

import java.io.*;
import java.net.*;

public class UdpClient {

    public void go() throws IOException, UnknownHostException {

        DatagramSocket datagramSocket;
        // Datagram packet to the server
        DatagramPacket outDataPacket;
        // Datagram packet from the server
        DatagramPacket inDataPacket;
        // Server host address
        InetAddress serverAddress;
        // Buffer space.
        byte[] msg = new byte[100];
        // Received message in String form.
        String receivedMsg;

        // Allocate a socket by which messages are sent
        // and received.
        datagramSocket = new DatagramSocket();

        // Server is running on this same machine for this
        // example.
        // This method can throw an UnknownHostException.
        serverAddress = InetAddress.getLocalHost();

        // Set up a datagram request to be sent to the server.
        // Send to port 8000.
        outDataPacket =
            new DatagramPacket(msg, 1, serverAddress, 8000);

        // Make the request to the server.
        datagramSocket.send(outDataPacket);
Minimal UDP Client

```java
// Set up a datagram packet to receive
// server's response.
inDataPacket = new DatagramPacket(msg, msg.length);

// Receive the time data from the server
datagramSocket.receive(inDataPacket);

// Print the data received from the server
receivedMsg = new String(
    inDataPacket.getData(), 0, inDataPacket.getLength());
System.out.println(receivedMsg);

// close the socket
datagramSocket.close();

public static void main(String args[]) {
    UdpClient udpClient = new UdpClient();

    try {
        udpClient.go();
    } catch (Exception e) {
        System.out.println("Exception occurred with socket.");
        System.out.println(e);
        System.exit(1);
    }
}
```
Exercise: Using Socket Programming

• Exercise objective:
  • Gain experience using sockets by implementing a client and server which communicate using sockets

• Tasks:
  • Create sockets
  • Use a multithreaded server
Check Your Progress

- Develop code to set up network connection
- Understand TCP/IP and UDP protocol
- Use `ServerSocket` and `Socket` classes for implementing TCP/IP client and servers
- Use `DatagramPacket` and `DatagramSocket` for effecting a UDP-based network communication
Think Beyond

• There are several advanced Java platform topics, many of which are addressed in other SunEd courses. See Appendix A for a brief discourse on some of them. Be sure and check out the JavaSoft web site (www.javasoft.com) as well.
Appendix B

Using the GridBagLayout
Layout Managers

• Position and size components in a Container
• Adhere to a policy
• Make absolute coordinates platform dependent
• Determine limitations of:
  • FlowLayout
  • GridLayout
  • BorderLayout
The GridBagLayout

- Divides the region into rows and columns

- Sizes components to fit width, height, both, or neither of their regions (one or more contiguous rows and one or more contiguous columns)
The GridBagLayout

- Row/column count determined by cell usage
- Row/column basic size determined by contents
The GridBagLayout

- Use of "spare" space is determined by weight.
- Components can fit width, height, or both of the region.
The GridBagLayout

- Components are located within a region by an anchor.
- Fill can make the anchor ineffective.
The GridBagConstraints Class

• For each component, specify:
  • Top left corner of cell with gridx and gridy
  • Cell size with gridwidth and gridheight
  • Capacity with fill
  • anchor

• For each row and column, specify:
  • Capacity with weightx and weighty
Designing with GridBagLayout

• Sketch all components
• Sketch all components on resized container
• Identify all gridlines and rowhence/column counts
• Identify stretchy rows/columns and allocate weights
• Identify starting row/column for each component
• Identify width/height for each component
• Identify fill for each component
• Identify anchor for each component
• Define row/column weights for each component
Example

Basic, unexpanded layout proposal

Basic, expanded layout proposal
Example

Extra column

Loose component
Example

The image shows a grid with components numbered from 1 to 12, each with specified coordinates and sizes.

<table>
<thead>
<tr>
<th>Component</th>
<th>gridx</th>
<th>gridy</th>
<th>gridwidth</th>
<th>gridheight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
Example
Example
import java.awt.*;
import com.sun.java.swing.*;

public class ExampleGB {
    public static void main(String args[]) {
        JFrame f = new JFrame(“GridBag Example”);
        Container c = f.getContentPane();
        c.setLayout(new GridBagLayout());
        GridBagAdder.add(c, new Canvas(), 3, 2, 1, 1, 1, 0,
            GridBagConstraints.NONE, GridBagConstraints.CENTER);
        GridBagAdder.add(c, new JButton(“1”), 0, 0, 5, 1, 0, 0,
            GridBagConstraints.HORIZONTAL, GridBagConstraints.CENTER);
        GridBagAdder.add(c, new JButton(“2”), 0, 1, 1, 1, 0, 0,
            GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        GridBagAdder.add(c, new JButton(“3”), 1, 1, 1, 1, 0,
            GridBagConstraints.HORIZONTAL, GridBagConstraints.CENTER);
        GridBagAdder.add(c, new JButton(“4”), 2, 1, 1, 1, 0, 0,
            GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        GridBagAdder.add(c, new JButton(“5”), 3, 1, 2, 1, 0, 0,
            GridBagConstraints.HORIZONTAL, GridBagConstraints.CENTER);
        GridBagAdder.add(c, new JButton(“6”), 0, 2, 1, 4, 0, 0,
            GridBagConstraints.HORIZONTAL, GridBagConstraints.CENTER);
        GridBagAdder.add(c, new JButton(“7”), 1, 2, 3, 4, 0, 0,
            GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        GridBagAdder.add(c, new JButton(“8”), 4, 2, 1, 1, 0, 1,
            GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        GridBagAdder.add(c, new JButton(“9”), 4, 3, 1, 1, 0, 1,
            GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        GridBagAdder.add(c, new JButton(“10”), 4, 4, 1, 1, 0, 1,
            GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        GridBagAdder.add(c, new JButton(“11”), 4, 5, 1, 1, 0, 1,
            GridBagConstraints.BOTH, GridBagConstraints.CENTER);
        GridBagAdder.add(c, new JButton(“12”), 0, 6, 5, 1, 0, 0,
            GridBagConstraints.HORIZONTAL, GridBagConstraints.CENTER);
        f.pack();
        f.setVisible(true);
    }
}
Example

```java
38 static class GridBagAdder {
39     // OK to reuse this as we overwrite all elements every time
40     // Note that this is not threadsafe however!
41     static GridBagConstraints cons = new GridBagConstraints();
42     public static void add(Container cont, Component comp, int x, int y,
43         int width, int height, int weightx, int weighty, int fill, int anchor) {
44         cons.gridx = x;
45         cons.gridy = y;
46         cons.gridwidth = width;
47         cons.gridheight = height;
48         cons.weightx = weightx;
49         cons.weighty = weighty;
50         cons.fill = fill;
51         cons.anchor = anchor;
52         cont.add(comp, cons);
53     }
54 }
55 }
```
RELATIVE and REMAINDER

• Shorthand for position, size, or both

• For gridx/gridy:
  RELATIVE => extends to the next position

• For gridwidth/gridheight:
  RELATIVE => extends to last one

• For gridwidth/gridheight:
  REMAINDER => extends to last one

• Careful use of these helps maintenance, but it:
  • Makes adding order significant
  • Might decrease readability of code
Think Beyond

- Are there any layout effects that you cannot handle using the layout managers you now understand?
About This Course .................................................................Preface-1
Course Goals ........................................................................Preface-2
Course Overview ................................................................Preface-3
Course Map ...........................................................................Preface-4
Module-by-Module Overview ................................................Preface-5
Course Objectives ................................................................Preface-7
Skills Gained by Module ......................................................Preface-9
Guidelines for Module Pacing ................................................Preface-10
Topics Not Covered ...............................................................Preface-11
How Prepared Are You? ......................................................Preface-12
Introductions ........................................................................Preface-13
How to Use Course Materials ..............................................Preface-14
Course Icons ........................................................................Preface-15
Typographical Conventions ................................................Preface-16

Getting Started .................................................................1-1
Course Map ............................................................................1-2
Objectives .............................................................................1-3
Relevance ..............................................................................1-5
What Is the Java Programming Language? ..........................1-6
Primary Goals of the Java Programming Language ..............1-7
The Java Virtual Machine ....................................................1-10
Garbage Collection ..............................................................1-13
Code Security .......................................................................1-14
Java Runtime Environment ................................................1-15
Class Loader .........................................................................1-16
Bytecode Verifier ...............................................................1-17
Compiling and Running HelloWorldApp ..............................1-19
Compile-Time Errors .............................................................1-20
Runtime Errors .....................................................................1-21
Identifiers, Keywords, and Types .............................................................................................................. 2-1
Course Map ................................................................................................................................................ 2-2
Objectives .................................................................................................................................................. 2-3
Relevance .................................................................................................................................................. 2-5
Comments ............................................................................................................................................... 2-6
Semicolons, Blocks, and Whitespace ......................................................................................................... 2-7
Identifiers ............................................................................................................................................... 2-9
Java Keywords ...................................................................................................................................... 2-10
Primitive Types .................................................................................................................................... 2-11
Logical – boolean ................................................................................................................................. 2-12
Textual – char and String ...................................................................................................................... 2-13
Integral – byte, short, int, and long ....................................................................................................... 2-15
Floating Point – float and double .......................................................................................................... 2-17
Variables, Declarations, and Assignments ............................................................................................ 2-19
Java Coding Conventions .................................................................................................................... 2-20
Understanding Objects .......................................................................................................................... 2-22
Creating an Object .............................................................................................................................. 2-23
Creating an Object – Memory Allocation and Layout ........................................................................... 2-24
Assignment of Reference Variables ....................................................................................................... 2-27
Assignment of Reference Variables ....................................................................................................... 2-28
Terminology Recap .................................................................................................................................. 2-29
Exercise: Using Identifiers, Keywords, and Types .................................................................................. 2-30
Check Your Progress ............................................................................................................................... 2-31
Objects and Classes ...............................................................5-1

Course Map .............................................................................. 5-2
Objectives .................................................................................. 5-3
Relevance .................................................................................. 5-5
Object Fundamentals ............................................................... 5-6
Classes and Objects .................................................................. 5-7
Defining Methods ....................................................................... 5-10
Pass-by-Value ............................................................................. 5-11
The this Reference ...................................................................... 5-12
Data Hiding ............................................................................... 5-13
Encapsulation .............................................................................. 5-15
Overloading Method Names ....................................................... 5-16
Constructing and Initializing Objects ......................................... 5-17
Explicit Member Initialization ..................................................... 5-18
Constructors ............................................................................. 5-19
Invoking Overloaded Constructors ............................................ 5-21
The Default Constructor ............................................................ 5-22
The is a Relationship .................................................................. 5-23
The extends Keyword ............................................................... 5-25
Single Inheritance ..................................................................... 5-26
Constructors Are Not Inherited ................................................ 5-28
Polymorphism ........................................................................... 5-29
Heterogeneous Collections ....................................................... 5-31
The instanceof Operator ................................................................. 5-33
Casting Objects ................................................................................ 5-34
Overriding Methods ........................................................................ 5-35
Rules About Overridden Methods .................................................. 5-38
The super Keyword .......................................................................... 5-40
Invoking Parent Class Constructors ................................................ 5-42
Packages ........................................................................................... 5-44
The import Statement ....................................................................... 5-45
Directory Layout and Packages ......................................................... 5-46
Exercise: Using Objects and Classes ............................................... 5-47
Check Your Progress ....................................................................... 5-48
Think Beyond ................................................................................... 5-50

**Advanced Language Features** ........................................................................... 6-1
Course Map .......................................................................................... 6-2
Objectives ............................................................................................. 6-3
Relevance .............................................................................................. 6-5
Class (static) Variables ....................................................................... 6-6
Static Initializers .................................................................................. 6-8
Static Methods and Data ..................................................................... 6-10
The final Keyword ............................................................................... 6-13
Abstract Classes .................................................................................. 6-14
Interfaces ............................................................................................... 6-15
Interfaces ............................................................................................... 6-16
Advanced Access Control ................................................................. 6-18
Deprecation .......................................................................................... 6-19
The == Operator Versus equals() Method ......................................... 6-23
toString() Method .............................................................................. 6-24
Inner Classes ....................................................................................... 6-25
Properties of Inner Classes ................................................................. 6-26
# Building GUIs

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check Your Progress</td>
<td>7-16</td>
</tr>
<tr>
<td>Think Beyond</td>
<td>7-17</td>
</tr>
<tr>
<td>Course Map</td>
<td>8-2</td>
</tr>
<tr>
<td>Objectives</td>
<td>8-3</td>
</tr>
<tr>
<td>Relevance</td>
<td>8-5</td>
</tr>
<tr>
<td>The AWT</td>
<td>8-6</td>
</tr>
<tr>
<td>The <code>java.awt</code> Package</td>
<td>8-7</td>
</tr>
<tr>
<td>Containers</td>
<td>8-8</td>
</tr>
<tr>
<td>Building Graphical User Interfaces</td>
<td>8-9</td>
</tr>
<tr>
<td>Frame</td>
<td>8-10</td>
</tr>
<tr>
<td>MyFrame.java</td>
<td>8-11</td>
</tr>
<tr>
<td>Panel</td>
<td>8-13</td>
</tr>
<tr>
<td>FrameWithPanel.java</td>
<td>8-14</td>
</tr>
<tr>
<td>Container Layouts</td>
<td>8-16</td>
</tr>
<tr>
<td>Default Layout Managers</td>
<td>8-17</td>
</tr>
<tr>
<td>A Simple FlowLayout Example</td>
<td>8-18</td>
</tr>
<tr>
<td>FlowLayout Manager</td>
<td>8-19</td>
</tr>
<tr>
<td>MyFlow.java</td>
<td>8-20</td>
</tr>
<tr>
<td>ExGui2.java</td>
<td>8-22</td>
</tr>
<tr>
<td>GridLayout Manager</td>
<td>8-24</td>
</tr>
<tr>
<td>GridEx.java</td>
<td>8-25</td>
</tr>
<tr>
<td>GridBagLayout Manager</td>
<td>8-31</td>
</tr>
<tr>
<td>ExGui3.java</td>
<td>8-32</td>
</tr>
<tr>
<td>Output of ExGui3.java</td>
<td>8-33</td>
</tr>
<tr>
<td>Exercise: Building GUIs</td>
<td>8-34</td>
</tr>
<tr>
<td>Check Your Progress</td>
<td>8-35</td>
</tr>
<tr>
<td>Think Beyond</td>
<td>8-37</td>
</tr>
</tbody>
</table>
### The AWT Event Model

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course Map</td>
<td>9-2</td>
</tr>
<tr>
<td>Objectives</td>
<td>9-3</td>
</tr>
<tr>
<td>Relevance</td>
<td>9-4</td>
</tr>
<tr>
<td>JDK 1.0 Event Model Versus Java 2 SDK Event Model</td>
<td>9-6</td>
</tr>
<tr>
<td>Hierarchical Model (JDK 1.0)</td>
<td>9-7</td>
</tr>
<tr>
<td>Delegation Model</td>
<td>9-9</td>
</tr>
<tr>
<td>Delegation Model</td>
<td>9-10</td>
</tr>
<tr>
<td>Frame With a Single Button</td>
<td>9-12</td>
</tr>
<tr>
<td>The ButtonHandler Class</td>
<td>9-13</td>
</tr>
<tr>
<td>Event Categories</td>
<td>9-14</td>
</tr>
<tr>
<td>Java GUI Behavior</td>
<td>9-15</td>
</tr>
<tr>
<td>Complex Example</td>
<td>9-17</td>
</tr>
<tr>
<td>Complex Example</td>
<td>9-18</td>
</tr>
<tr>
<td>Multiple Listeners</td>
<td>9-19</td>
</tr>
<tr>
<td>Event Adapters</td>
<td>9-20</td>
</tr>
<tr>
<td>Anonymous Classes</td>
<td>9-21</td>
</tr>
<tr>
<td>Exercise: Working With Events</td>
<td>9-22</td>
</tr>
<tr>
<td>Check Your Progress</td>
<td>9-23</td>
</tr>
<tr>
<td>Think Beyond</td>
<td>9-24</td>
</tr>
</tbody>
</table>

### The AWT Component Library

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course Map</td>
<td>10-2</td>
</tr>
<tr>
<td>Objectives</td>
<td>10-3</td>
</tr>
<tr>
<td>Relevance</td>
<td>10-4</td>
</tr>
<tr>
<td>Features of the AWT</td>
<td>10-5</td>
</tr>
<tr>
<td>Creating a Button</td>
<td>10-6</td>
</tr>
<tr>
<td>Creating a Checkbox</td>
<td>10-7</td>
</tr>
<tr>
<td>Creating the ItemListener Interface</td>
<td>10-8</td>
</tr>
</tbody>
</table>
Creating a Choice ................................................................. 10-10
Canvas .................................................................................. 10-11
Creating a Label .................................................................... 10-13
Creating a TextField .............................................................. 10-14
Creating a TextArea ................................................................. 10-15
Text Components .................................................................. 10-16
Creating a List Component ....................................................... 10-17
Creating a Dialog .................................................................. 10-18
Creating a FileDialog .............................................................. 10-19
Creating a ScrollPane ............................................................. 10-20
Menu ..................................................................................... 10-21
Creating a MenuBar ............................................................... 10-22
Creating a Menu .................................................................. 10-23
Creating a MenuItem ............................................................... 10-24
Creating a CheckBoxMenuItem ................................................ 10-25
Creating a PopupMenu ........................................................... 10-26
Controlling Visual Aspects ......................................................... 10-27
Printing .................................................................................... 10-29
Exercise: Creating a Paint Program Layout .................................. 10-30
Check Your Progress ................................................................. 10-31
Think Beyond ........................................................................ 10-32

Java Foundation Classes ............................................................ 11-1
Course Map ........................................................................... 11-2
Objectives ............................................................................. 11-3
Relevance .............................................................................. 11-4
Introduction .......................................................................... 11-5
Swing Introduction ................................................................. 11-6
Swing Hierarchy .................................................................... 11-7
Swing Components ................................................................. 11-8
### A Basic Swing Application ................................................................. 11-10
HelloSwing ......................................................................................... 11-11
HelloSwing ......................................................................................... 11-12
HelloSwing ......................................................................................... 11-13
Basic Swing Application ........................................................................ 11-14
Building a Swing GUI ........................................................................ 11-16
The JComponent Class .......................................................................... 11-19
Exercise: Creating Swing Applications ................................................ 11-20
Check Your Progress ........................................................................ 11-21
Think Beyond ..................................................................................... 11-22

### Introduction to Java Applets ......................................................... 12-1
Course Map ......................................................................................... 12-2
Objectives ......................................................................................... 12-3
Relevance ......................................................................................... 12-5
What Is an Applet? ........................................................................... 12-6
Applet Security Restrictions ............................................................. 12-7
Applet Class Hierarchy ...................................................................... 12-8
Key Applet Methods .......................................................................... 12-9
Applet Display .................................................................................... 12-10
Applet Methods and the Applet Life Cycle ....................................... 12-11
AWT Painting ..................................................................................... 12-12
Applet Display Strategies ................................................................. 12-14
What Is the appletviewer? ................................................................. 12-15
The applet Tag .................................................................................. 12-16
Additional Applet Facilities ............................................................... 12-17
A Simple Image Test .......................................................................... 12-18
AudioClip ......................................................................................... 12-19
A Simple Audio Test .......................................................................... 12-20
Looping an AudioClip ....................................................................... 12-21
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Simple Looping Test</td>
<td>12-22</td>
</tr>
<tr>
<td>Mouse Input</td>
<td>12-23</td>
</tr>
<tr>
<td>A Simple Mouse Test</td>
<td>12-24</td>
</tr>
<tr>
<td>Reading Parameters</td>
<td>12-25</td>
</tr>
<tr>
<td>Dual Purpose Code Sample</td>
<td>12-26</td>
</tr>
<tr>
<td>Exercise: Creating Applets</td>
<td>12-28</td>
</tr>
<tr>
<td>Check Your Progress</td>
<td>12-29</td>
</tr>
<tr>
<td>Think Beyond</td>
<td>12-31</td>
</tr>
</tbody>
</table>

### Threads  

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course Map</td>
<td>13-2</td>
</tr>
<tr>
<td>Objectives</td>
<td>13-3</td>
</tr>
<tr>
<td>Relevance</td>
<td>13-5</td>
</tr>
<tr>
<td>Threads</td>
<td>13-6</td>
</tr>
<tr>
<td>Three Parts of a Thread</td>
<td>13-7</td>
</tr>
<tr>
<td>Creating the Thread</td>
<td>13-8</td>
</tr>
<tr>
<td>Starting the Thread</td>
<td>13-10</td>
</tr>
<tr>
<td>Thread Scheduling</td>
<td>13-11</td>
</tr>
<tr>
<td>Terminating a Thread</td>
<td>13-13</td>
</tr>
<tr>
<td>Basic Control of Threads</td>
<td>13-14</td>
</tr>
<tr>
<td>Putting Threads on Hold</td>
<td>13-15</td>
</tr>
<tr>
<td>Extending the Thread Class</td>
<td>13-17</td>
</tr>
<tr>
<td>Selecting a Way to Create Threads</td>
<td>13-18</td>
</tr>
<tr>
<td>Using the synchronized Keyword</td>
<td>13-19</td>
</tr>
<tr>
<td>The Object Lock Flag</td>
<td>13-20</td>
</tr>
<tr>
<td>The Object Lock Flag</td>
<td>13-21</td>
</tr>
<tr>
<td>Releasing the Lock Flag</td>
<td>13-22</td>
</tr>
<tr>
<td>synchronized – Putting It Together</td>
<td>13-23</td>
</tr>
<tr>
<td>Deadlock</td>
<td>13-25</td>
</tr>
<tr>
<td>Thread Interaction – wait() and notify()</td>
<td>13-26</td>
</tr>
</tbody>
</table>

Java Programming Language  
Copyright 1999 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services September 1999, Revision C.1
### Thread Interaction

- Monitor Model for Synchronization .................................................................................. 13-28
- Producer ............................................................................................................................. 13-29
- Consumer ............................................................................................................................ 13-30
- pop() Method ..................................................................................................................... 13-32
- push() Method .................................................................................................................... 13-33
- SyncTest.java .................................................................................................................... 13-34
- Producer.java ..................................................................................................................... 13-35
- Consumer.java ................................................................................................................... 13-36
- SyncStack.java ..................................................................................................................... 13-37
- The suspend() and resume() Methods ........................................................................... 13-38
- The stop() Method ............................................................................................................. 13-39
- Proper Thread Control ....................................................................................................... 13-40
- Exercise: Using Multithreaded Programming ................................................................. 13-41
- Check Your Progress .......................................................................................................... 13-42
- Think Beyond ...................................................................................................................... 13-44

### Stream I/O and Files

- Course Map ......................................................................................................................... 14-2
- Objectives ............................................................................................................................ 14-3
- Relevance ............................................................................................................................. 14-4
- Stream I/O ........................................................................................................................... 14-5
- Stream Fundamentals ......................................................................................................... 14-6
- InputStream Methods ........................................................................................................ 14-7
- OutputStream Methods .................................................................................................... 14-8
- Basic Stream Classes ......................................................................................................... 14-9
- URL Input Streams ............................................................................................................ 14-11
- Opening an Input Stream .................................................................................................. 14-12
- Readers and Writers ........................................................................................................... 14-13
- Reading String Input ........................................................................................................... 14-14
Creating a New File Object ................................................................. 14-15
File Tests and Utilities ........................................................................ 14-16
Creating a Random Access File .......................................................... 14-18
Random Access Files .......................................................................... 14-19
Serialization ........................................................................................ 14-20
Writing an Object to a File Stream ...................................................... 14-21
Reading an Object From a File Stream ............................................... 14-22
Exercise: Getting Acquainted With I/O ............................................ 14-23
Check Your Progress ........................................................................... 14-24
Think Beyond ....................................................................................... 14-25

Networking ......................................................................................... 15-1
Course Map ......................................................................................... 15-2
Objectives ............................................................................................ 15-3
Relevance .............................................................................................. 15-4
Networking ......................................................................................... 15-5
Networking With Java Technology .................................................... 15-6
Java Networking Model ....................................................................... 15-7
Minimal TCP/IP Server ....................................................................... 15-8
  Minimal TCP/IP Server .................................................................... 15-9
  Minimal TCP/IP Client .................................................................... 15-10
UDP Sockets ........................................................................................ 15-11
The DatagramPacket ......................................................................... 15-12
The DatagramSocket ......................................................................... 15-13
Minimal UDP Server .......................................................................... 15-14
Minimal UDP Client .......................................................................... 15-17
Exercise: Using Socket Programming .............................................. 15-19
Check Your Progress ......................................................................... 15-20
Think Beyond ...................................................................................... 15-21
Using the GridBagLayout .................................................................................................................. B-1
  Layout Managers ............................................................................................................................... B-2
  The GridBagLayout ......................................................................................................................... B-3
  The GridBagConstraints Class .......................................................................................................... B-7
  Designing with GridBagLayout ........................................................................................................ B-8
  Example ............................................................................................................................................... B-9
  RELATIVE and REMAINDER .......................................................................................................... B-16
  Think Beyond .................................................................................................................................... B-17
Copyright 1999 Sun Microsystems Inc., 901 San Antonio Road, Palo Alto, California 94303, Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l’utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l’autorisation préalable et écrite de Sun et de ses bailleurs de licence, s’il y en a.

Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du systèmes Berkeley 4.3 BSD licenciés par l’Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d’autres pays et licenciée exclusivement par X/Open Company Ltd.

Sun, Sun Microsystems, le logo Sun, sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d’autres pays.

Toutes les marques SPARC sont utilisées sous licence sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d’autres pays.

Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

UNIX est une marque déposée aux Etats-Unis et dans d’autres pays et licenciée exclusivement par X/Open Company, Ltd.

L’interfaces d’utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d’utilisation visuelle ou graphique pour l’industrie de l’infomaticque. Sun détient une licence non exclusive de Xerox sur l’interface d’utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l’interface d’utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

L’accord du gouvernement américain est requis avant l’exportation du produit.

Le système X Window est un produit de X Consortium, Inc.

LA DOCUMENTATION EST FOURNIE “EN L’ETAT” ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUS, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L’APTITUDE A UNE UTILISATION PARTICULIERE OU A L’ABSENCE DE CONTREFAÇON.