Java™ Programming Language

SL-275

Sun microsystems

Copyright 2006 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Sun, Sun Microsystems, the Sun Logo, Java, JavaBeans, JavaOS, JDBC, JDK, JVM, J2SE, Jini, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Netscape Navigator is a trademark of Netscape Communications Corporation.

The OPEN LOOK and Sun Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

U.S. Government approval might be required when exporting the product.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015 (b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS, AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Export Commodity Classification Number Assigned: 2 February 2002

Copyright 2006 Sun Microsystems Inc., 4150 Network Circle, Santa Clara, California 95054, 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.

Sun, Sun Microsystems, the Sun Logo, Java, JavaBeans, JavaOS, JDBC, JDK, JVM, J2SE, Jini, et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

UNIX est une marques déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Netscape Navigator est une marque de Netscape Communications Corporation aux Etats-Unis et dans d'autres pays.

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 larecherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. 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.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, 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.

Course Contents

2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Preface

About This Course

Course Goals

This course provides you with knowledge and skills to:

- Create Java™ technology applications that leverage the object-oriented features of the Java language, such as encapsulation, inheritance, and polymorphism
- Execute a Java technology application from the command-line
- Use Java technology data types and expressions
- Use Java technology flow control constructs
- Use arrays and other data collections
- Implement error-handling techniques using exception handling

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Course Goals

- Create an event-driven graphical user interface (GUI) by using Java technology GUI components: panels, buttons, labels, text fields, and text areas
- Implement input/output (I/O) functionality to read from and write to data and text files
- Create multithreaded programs
- Create a simple Transmission Control Protocol/Internet Protocol (TCP/IP) client that communicates through sockets

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Course Overview

This course describes the following areas:

- The syntax of the Java programming language
- Object-oriented concepts as they apply to the Java programming language
- GUI programming
- Multithreading
- Networking

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Course Map

The Java Programming Language Basics

- Getting Started
- Object-Oriented Programming
- Identifiers, Keywords, and Types
- Expressions and Flow Control
- Arrays

More Object-Oriented Programming

- Class Design
- Advanced Class Features

Building Applications

- Exceptions and Assertions
- Text-Based Applications

Developing Graphical User Interfaces

- Building Java GUIs
- GUI Event Handling
- GUI-Based Applications

Advanced Java Programming

- Threads
- Advanced I/O Streams
- Networking

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Topics Not Covered

- Object-oriented analysis and design – Covered in OO-226: *Object-Oriented Application Analysis and Design Using UML*
- General programming concepts – Covered in SL-110: *Fundamentals of the Java™ Programming Language*

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

How Prepared Are You?

Before attending this course, you should have completed SL-110: *Fundamentals of the Java™ Programming Language*, or have:

- Created and 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™

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Introductions

- Name
- Company affiliation
- Title, function, and job responsibility
- Experience related to topics presented in this course
- Reasons for enrolling in this course
- Expectations for this course

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

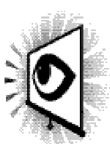
How to Use the Icons

Additional resources

Discussion

Note

Caution



Visual Aid

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Typographical Conventions and Symbols

- `Courier` is used for the names of commands, files, directories, programming code, programming constructs, and on-screen computer output.
- **`Courier bold`** is used for characters and numbers that you type, and for each line of programming code that is referenced in a textual description.
- *`Courier italics`* is used for variables and command-line placeholders that are replaced with a real name or value.
- ***`Courier italics bold`*** is used to represent variables whose values are to be entered by the student as part of an activity.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Typographical Conventions and Symbols

- *Palatino italics* is used for book titles, new words or terms, or words that are emphasized.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Additional Conventions

Java programming language examples use the following additional conventions:

- `Courier` is used for the class names, methods, and keywords.
- Methods are not followed by parentheses unless a formal or actual parameter list is shown.
- Line breaks occur where there are separations, conjunctions, or white space in the code.
- If a command on the Solaris™ Operating System (Solaris OS) is different from the Microsoft Windows platform, both commands are shown.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 1

Getting Started

Objectives

- Describe the key features of Java technology
- Write, compile, and run a simple Java technology application
- Describe the function of the Java Virtual Machine (JVM™)
- Define garbage collection
- List the three tasks performed by the Java platform that handle code security

NOTE: The terms “Java Virtual Machine” and “JVM” mean a Virtual Machine for the Java™ platform.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Relevance

- Is the Java programming language a complete language or is it useful only for writing programs for the Web?
- Why do you need another programming language?
- How does the Java technology platform improve on other language platforms?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

What Is the Java™ Technology?

- Java technology is:
  - A programming language
  - A development environment
  - An application environment
  - A deployment environment
- It is similar in syntax to C++.
- It is used for developing both *applets* and *applications*.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Primary Goals of the Java Technology

- Provides an easy-to-use language by:
  - Avoiding many pitfalls of other languages
  - Being object-oriented
  - Enabling users to create streamlined and clear code
- Provides an interpreted environment for:
  - Improved speed of development
  - Code portability

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Primary Goals of the Java Technology

- Enables users to run more than one thread of activity
- Loads classes dynamically; that is, at the time they are actually needed
- Supports changing programs dynamically during runtime by loading classes from disparate sources
- Furnishes better security

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Primary Goals of the Java Technology

The following features fulfill these goals:

- The Java Virtual Machine (JVM™)[1]
- Garbage collection
- The Java Runtime Environment (JRE)
- JVM tool interface

1. The terms "Java Virtual Machine" and "JVM" mean a Virtual Machine for the Java platform

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

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

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

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
- Fatal error reporting
- High-precision timing support

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Java Virtual Machine

- The majority of type checking is done when the code is compiled.
- Implementation of the JVM approved by Sun Microsystems must be able to run any compliant class file.
- The JVM executes on multiple operating environments.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

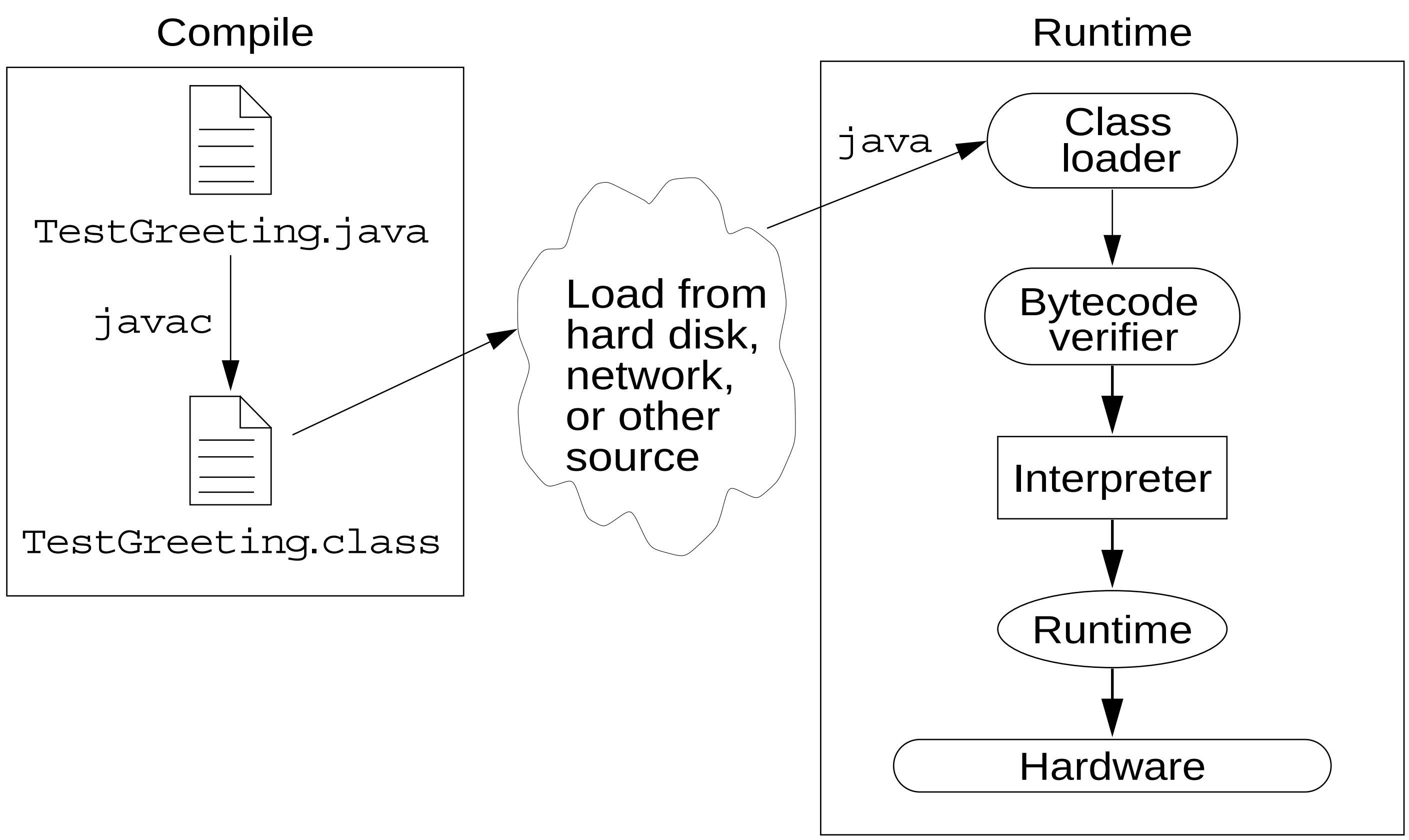
Garbage Collection

- 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 has the following characteristics:
  - Checks for and frees memory no longer needed
  - Is done automatically
  - Can vary dramatically across JVM implementations

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

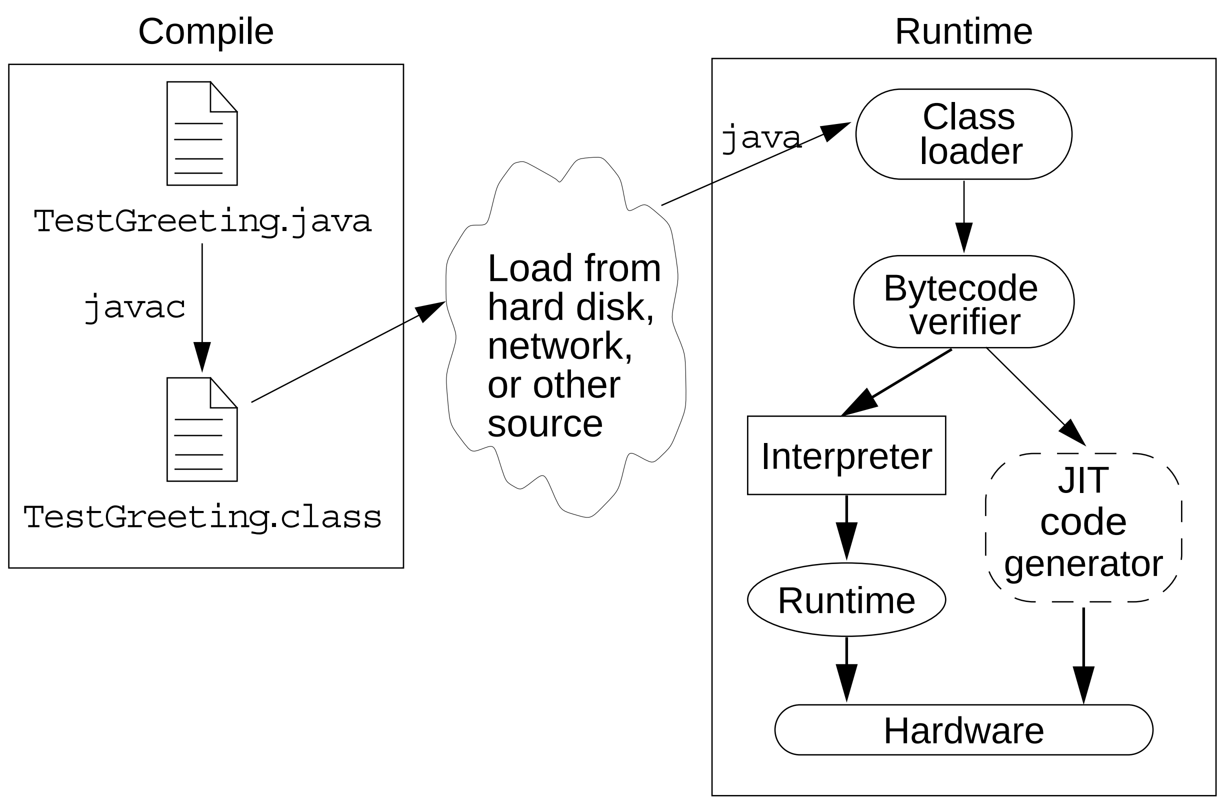
The Java Runtime Environment

The Java application environment performs as follows:



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Operation of the JRE With a Just-In-Time (JIT) Compiler



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

JVM™ Tasks

The JVM performs three main tasks:

- Loads code
- Verifies code
- Executes code

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Class Loader

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

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The 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.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

A Simple Java Application

The `TestGreeting.java` Application

```
//
// Sample "Hello World" application
//
public class TestGreeting{
  public static void main (String[] args) {
    Greeting hello = new Greeting();
    hello.greet();
  }
}
```

The `Greeting.java` Class

```
public class Greeting {
  public void greet() {
    System.out.println(“hi”);
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `TestGreeting` Application

- Comment lines
- Class declaration
- The `main` method
- Method body

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `Greeting` Class

- Class declaration
- The `greet` method

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Compiling and Running the `TestGreeting` Program

- Compile `TestGreeting.java`:

  `javac TestGreeting.java`
- The `Greeting.java` is compiled automatically.
- Run the application by using the following command:

  `java TestGreeting`
- Locate common compile and runtime errors.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Compile-Time Errors

- `javac: Command not found`
- ```
  Greeting.java:4: cannot resolve symbol
  symbol  : method printl  (java.lang.String)
  location: class java.io.PrintStream
  System.out.printl("hi");
            ^
  ```
- ```
  TestGreet.java:4: Public class TestGreeting
  must be defined in a file called
  "TestGreeting.java".
  ```

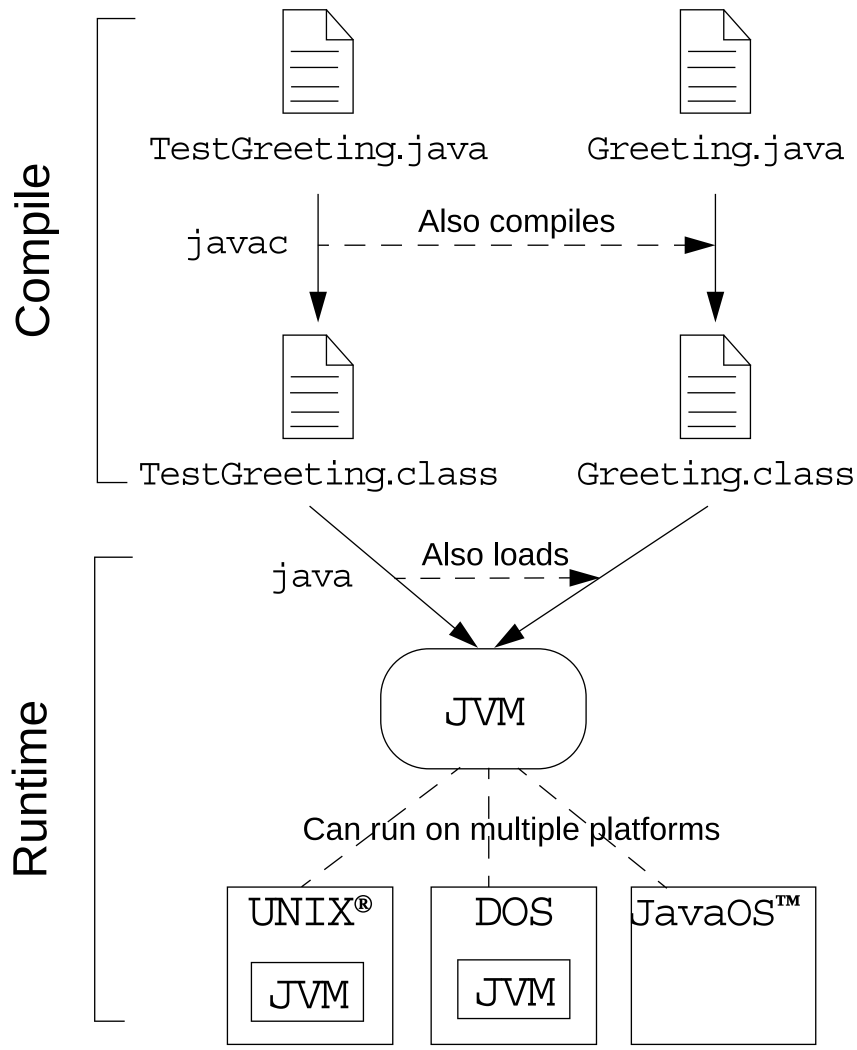

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Runtime Errors

- `Can’t find class TestGreeting`
- `Exception in thread "main" java.lang.NoSuchMethodError: main`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Java Technology Runtime Environment



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 2

Object-Oriented Programming

Objectives

- Define modeling concepts: *abstraction*, *encapsulation*, and *packages*
- Discuss why you can reuse Java technology application code
- Define *class*, *member*, *attribute*, *method*, *constructor*, and *package*
- Use the access modifiers `private` and `public` as appropriate for the guidelines of encapsulation
- Invoke a method on a particular object
- Use the Java technology application programming interface (API) online documentation

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Relevance

- What is your understanding of software analysis and design?
- What is your understanding of design and code reuse?
- What features does the Java programming language possess that make it an object-oriented language?
- Define the term *object-oriented*.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Software Engineering

Toolkits / Frameworks / Object APIs (1990s–Up)

Java 2 SDK | AWT / J.F.C./Swing | Jini™ | JavaBeans™ | JDBC™

Object-Oriented Languages (1980s–Up)

SELF | Smalltalk | Common Lisp Object System | Eiffel | C++ | Java

Libraries / Functional APIs (1960s–Early 1980s)

NASTRAN | TCP/IP | ISAM | X-Windows | OpenLook

High-Level Languages (1950s–Up)

Fortran | LISP | C | COBOL

Operating Systems (1960s–Up)

OS/360 | UNIX | MacOS | Microsoft Windows

Machine Code (Late 1940s–Up)

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Analysis and Design Phase

- Analysis describes *what* the system needs to do:

  Modeling the real-world, including actors and activities, objects, and behaviors
- Design describes *how* the system does it:
  - Modeling the relationships and interactions between objects and actors in the system
  - Finding useful abstractions to help simplify the problem or solution

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Abstraction

- Functions – Write an algorithm once to be used in many situations
- Objects – Group a related set of attributes and behaviors into a class
- Frameworks and APIs – Large groups of objects that support a complex activity; Frameworks can be used *as is* or be modified to extend the basic behavior

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Classes as Blueprints for Objects

- In manufacturing, a blueprint describes a device from which many physical devices are constructed.
- In software, a class is a description of an object:
  - A class describes the data that each object includes.
  - A class describes the behaviors that each object exhibits.
- In Java technology, classes support three key features of object-oriented programming (OOP):
  - Encapsulation
  - Inheritance
  - Polymorphism

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Declaring Java Technology Classes

- Basic syntax of a Java class:

```
<modifier>* class <class_name> {
  <attribute_declaration>*
  <constructor_declaration>*
  <method_declaration>*
}
```

- Example:

```
public class Vehicle {
  private double maxLoad;
  public void setMaxLoad(double value) {
    maxLoad = value;
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Declaring Attributes

- Basic syntax of an attribute:

```
<modifier>* <type> <name> [ = <initial_value>];
```

- Examples:

```
public class Foo {
  private int x;
  private float y = 10000.0F;
  private String name = "Bates Motel";
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Declaring Methods

- Basic syntax of a method:

```
<modifier>* <return_type> <name> ( <argument>* ) {
  <statement>*
}
```

- Examples:

```
public class Dog {
  private int weight;
  public int getWeight() {
    return weight;
  }
  public void setWeight(int newWeight) {
    if ( newWeight > 0 ) {
      weight = newWeight;
    }
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Accessing Object Members

- The *dot* notation is: ***<object>.<member>***
- This is used to access object members, including attributes and methods.
- Examples of dot notation are:

```
d.setWeight(42);
d.weight = 42;  // only permissible if weight is public
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Information Hiding

The problem:

MyDate
+day : int +month : int +year : int

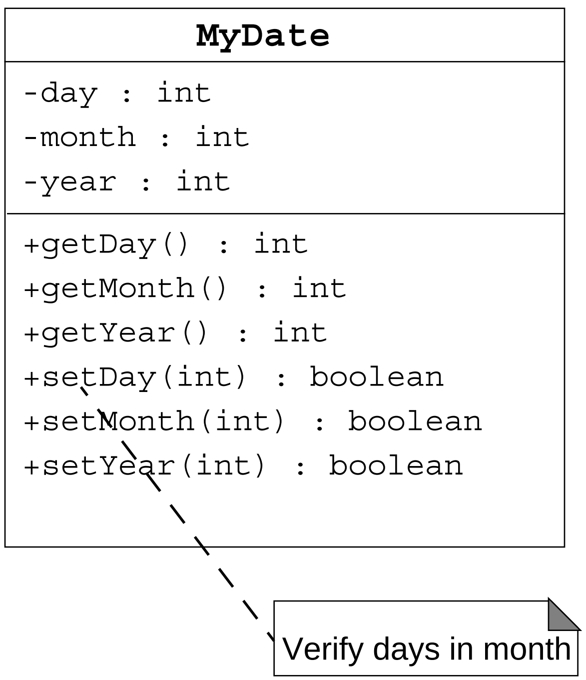
Client code has direct access to internal data (`d` refers to a `MyDate` object):

```
d.day = 32;
// invalid day

d.month = 2; d.day = 30;
// plausible but wrong

d.day = d.day + 1;
// no check for wrap around
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Information Hiding

The solution:



Client code must use setters and getters to access internal data:

```
MyDate d = new MyDate();

d.setDay(32);
// invalid day, returns false

d.setMonth(2);
d.setDay(30);
// plausible but wrong,
// setDay returns false

d.setDay(d.getDay() + 1);
// this will return false if wrap around
// needs to occur
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Encapsulation

- Hides the implementation details of a class
- Forces the user to use an interface to access data
- Makes the code more maintainable

MyDate
`-date : long`

`+getDay() : int`
`+getMonth() : int`
`+getYear() : int`
`+setDay(int) : boolean`
`+setMonth(int) : boolean`
`+setYear(int) : boolean`
`-isDayValid(int) : boolean`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Declaring Constructors

- Basic syntax of a constructor:

```
[<modifier>] <class_name> ( <argument>* ) {
  <statement>*
}
```

- Example:

```
public class Dog {

  private int weight;

  public Dog() {
     weight = 42;
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Default Constructor

- There is always at least one constructor in every class.
- If the writer does not supply any constructors, the default constructor is present automatically:
  - The default constructor takes no arguments
  - The default constructor body is empty
- The default enables you to create object instances with `new` *`Xxx`*`()` without having to write a constructor.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Source File Layout

- Basic syntax of a Java source file is:

```
[<package_declaration>]
<import_declaration>*
<class_declaration>+
```

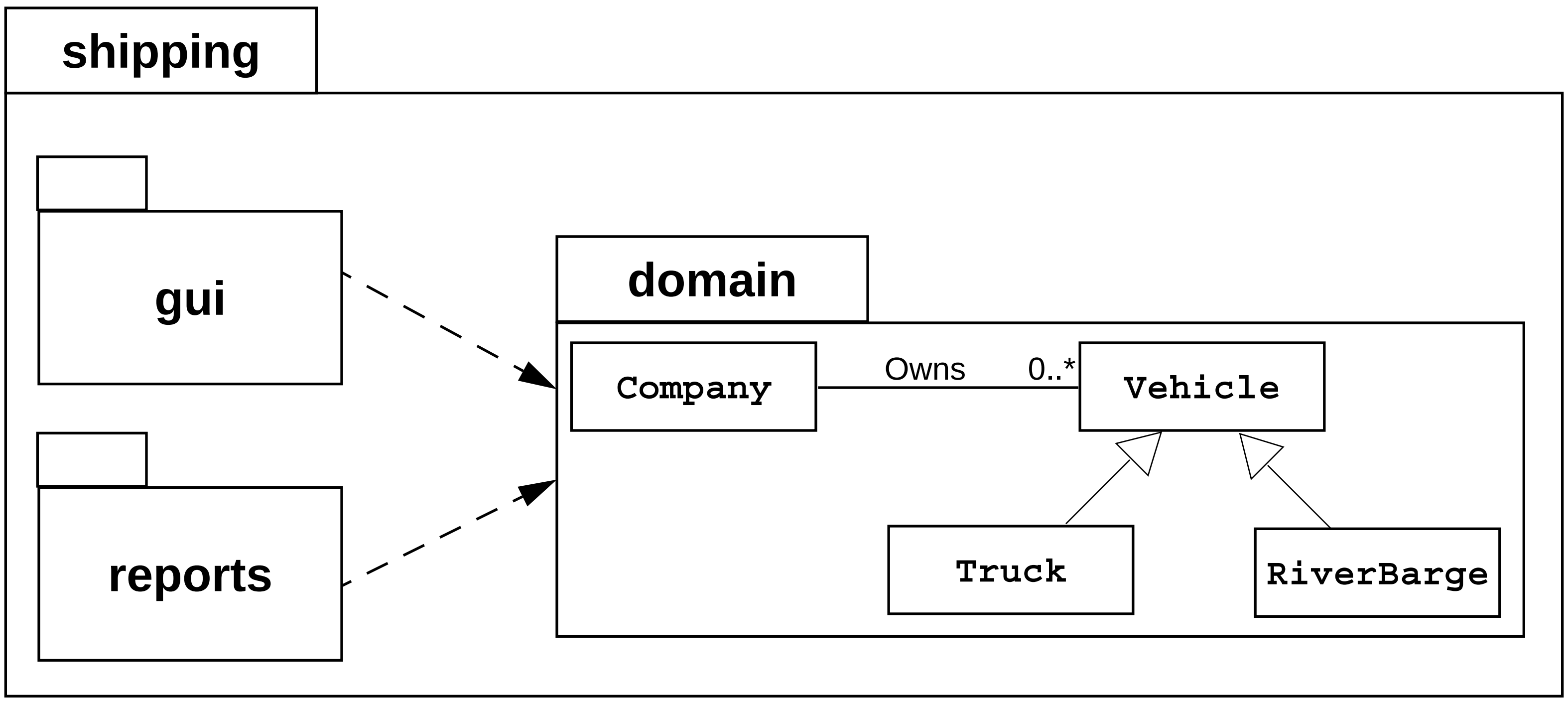
- For example, the `VehicleCapacityReport.java` file is:

```
package shipping.reports;

import shipping.domain.*;
import java.util.List;
import java.io.*;

public class VehicleCapacityReport {
  private List vehicles;
  public void generateReport(Writer output) {...}
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Software Packages

- Packages help manage large software systems.
- Packages can contain classes and sub-packages.



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `package` Statement

- Basic syntax of the `package` statement is:

```
package <top_pkg_name>[.<sub_pkg_name>]*;
```

- Examples of the statement are:

```
package shipping.gui.reportscreens;
```

- Specify the package declaration at the beginning of the source file.
- Only one package declaration per source file.
- If no package is declared, then the class is placed into the default package.
- Package names must be hierarchical and separated by dots.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `import` Statement

- Basic syntax of the `import` statement is:

```
import <pkg_name>[.<sub_pkg_name>]*.<class_name>;
OR
import <pkg_name>[.<sub_pkg_name>]*.*;
```

- Examples of the statement are:

```
import java.util.List;
import java.io.*;
import shipping.gui.reportscreens.*;
```

- The import statement does the following:
  - Precedes all class declarations
  - Tells the compiler where to find classes

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Directory Layout and Packages

- Packages are stored in the directory tree containing the package name.
- An example is the shipping application packages.

```
shipping/
 |
 |------ domain/
 |        |
 |        |------ Company.class
 |        |------ Vehicle.class
 |        |------ RiverBarge.class
 |        |------ Truck.class
 |------ gui/
 |------ reports/
          |
          |------ VehicleCapacityReport.class
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Development

```
JavaProjects/
 └── ShippingPrj/
      ├── src/
      │    └── shipping/
      │         ├── domain/
      │         ├── gui/
      │         └── reports/
      ├── docs/
      └── classes/
           └── shipping/
                ├── domain/
                ├── gui/
                └── reports/
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Compiling Using the `-d` Option

```
cd JavaProjects/ShippingPrj/src
javac -d ../classes shipping/domain/*.java
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Terminology Recap

- Class – The source-code blueprint for a run-time object
- Object – An instance of a class;
  also known as *instance*
- Attribute – A data element of an object;
  also known as *data member*, *instance variable*, and *data field*
- Method – A behavioral element of an object;
  also known as *algorithm*, *function*, and *procedure*
- Constructor – A *method-like* construct used to initialize a new object
- Package – A grouping of classes and sub-packages

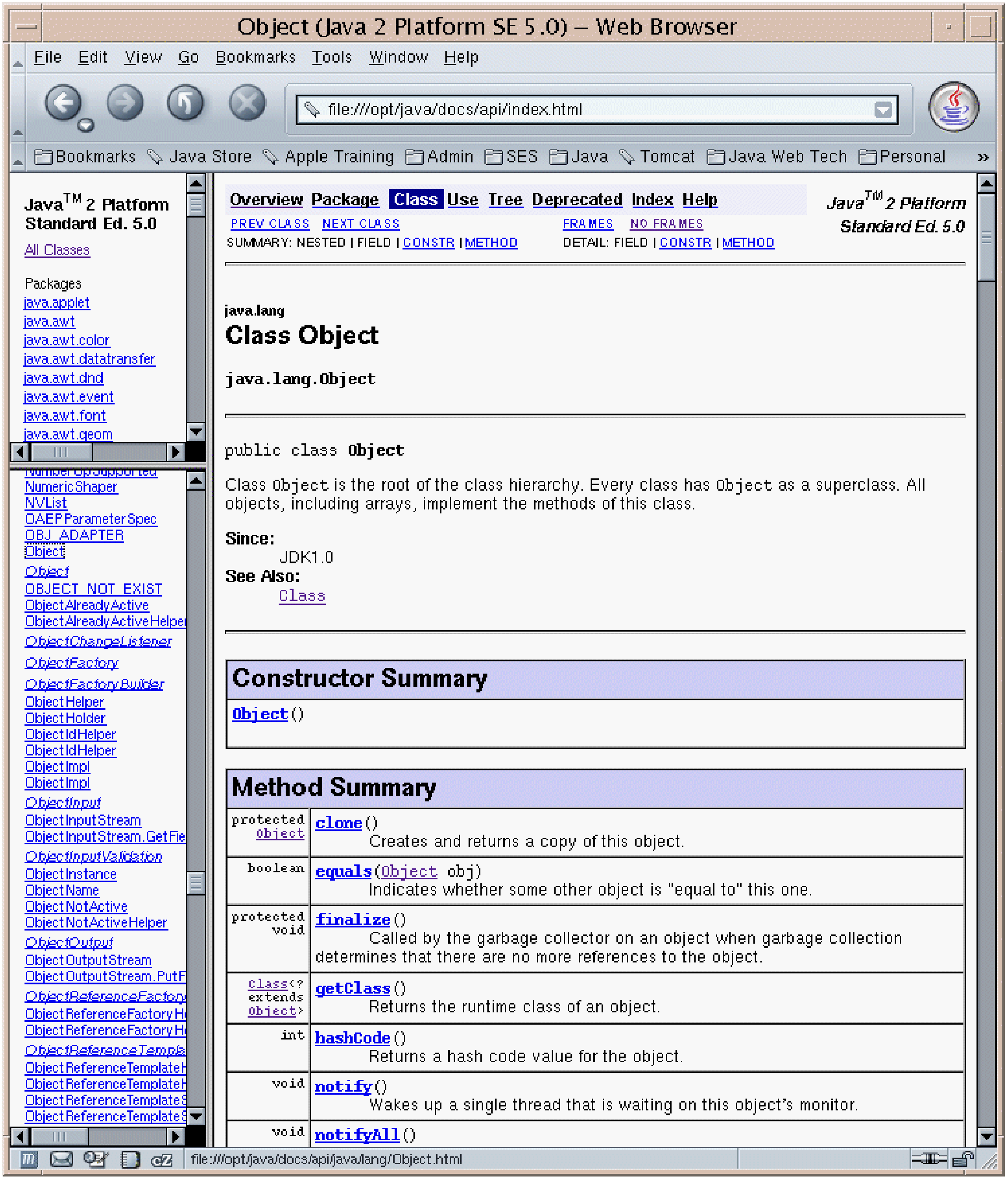
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Using the Java Technology API Documentation

- A set of Hypertext Markup Language (HTML) files provides information about the API.
- A frame describes a package and contains hyperlinks to information describing each class in that package.
- A class document includes the class hierarchy, a description of the class, a list of member variables, a list of constructors, and so on.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Java Technology API Documentation With HTML3



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 3

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 *primitive variable* and *reference variable*

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Objectives

- Declare variables of class type
- Construct an object using `new`
- Describe default initialization
- Describe the significance of a reference variable
- State the consequences of assigning variables of class type

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Relevance

- Do you know the primitive Java types?
- Can you describe the difference between variables holding primitive values as compared with object references?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Comments

The three permissible styles of comment in a Java technology program are:

```
// comment on one line

/* comment on one
 * or more lines
 */

/** documentation comment
 *  can also span one or more lines
 */
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Semicolons, Blocks, and White Space

- A *statement* is one or more lines of code terminated by a semicolon (`;`):

```
totals = a + b + c
         + d + e + f;
```

- A *block* is a collection of statements bound by opening and closing braces:

```
{
  x = y + 1;
  y = x + 1;
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Semicolons, Blocks, and White Space

- A *class* definition uses a special block:

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

- You can nest block statements.

```
while ( i < large ) {
  a = a + i;
  // nested block
  if ( a == max ) {
    b = b + a;
    a = 0;
  }
  i = i + 1;
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Semicolons, Blocks, and White Space

- Any amount of *white space* is permitted in a Java program.

  For example:

  ```
  {int x;x=23*54;}
  ```

  is equivalent to:

  ```
  {
    int x;

    x = 23 * 54;
  }
  ```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Identifiers

Identifiers have the following characteristics:

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

```
identifier
userName
user_name
_sys_var1
$change
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Java Programming Language Keywords

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Reserved literal words: `null`, `true`, and `false`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Primitive Types

The Java programming language defines eight primitive types:

- Logical – `boolean`
- Textual – `char`
- Integral – `byte`, `short`, `int`, and `long`
- Floating – `double` and `float`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Logical – `boolean`

The `boolean` primitive has the following characteristics:

- The `boolean` data type has two literals, `true` and `false`.
- For example, the statement:

```
boolean truth = true;
```

  declares the variable `truth` as `boolean` type and assigns it a value of `true`.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Textual – `char`

The textual `char` primitive has the following characteristics:

- Represents a 16-bit Unicode character
- Must have its literal enclosed in single quotes (’ ’)
- Uses the following notations:

`'a'`	The letter `a`
`'\t'`	The tab character
`'\u????'`	A specific Unicode character, `????`, is replaced with exactly four hexadecimal digits . For example, `’\u03A6’` is the Greek letter phi [Φ].

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Textual – `String`

The textual `String` type has the following characteristics:

- 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 errorMessage = "Record Not Found !";
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Integral – `byte`, `short`, `int`, and `long`

The integral primitives have the following characteristics:

- Integral primates use three forms: Decimal, octal, or hexadecimal

`2`	The decimal form for the integer `2`.
`077`	The leading `0` indicates an octal value.
`0xBAAC`	The leading `0x` indicates a hexadecimal value.

- Literals have a default type of `int`.
- Literals with the suffix `L` or `l` are of type `long`.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Integral – `byte`, `short`, `int`, and `long`

- Integral data types have the following ranges:

Integer Length	Name or Type	Range
8 bits	`byte`	-2^{7} to $2^{7}-1$
16 bits	`short`	-2^{15} to $2^{15}-1$
32 bits	`int`	-2^{31} to $2^{31}-1$
64 bits	`long`	-2^{63} to $2^{63}-1$

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Floating Point – `float` and `double`

The floating point primitives have the following characteristics:

- 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

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Floating Point – `float` and `double`

- Literals have a default type of `double`.
- Floating-point data types have the following sizes:

Float Length	Name or Type
32 bits	`float`
64 bits	`double`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Variables, Declarations, and Assignments

```
public class Assign {
  public static void main (String args []) {
    // declare integer variables
    int x, y;
    // declare and assign floating point
    float z = 3.414f;
    // declare and assign double
    double w = 3.1415;
    // declare and assign boolean
    boolean truth = true;
    // declare character variable
    char c;
    // declare String variable
    String str;
    // declare and assign String variable
    String str1 = "bye";
    // assign value to char variable
    c = 'A';
    // assign value to String variable
    str = "Hi out there!";
    // assign values to int variables
    x = 6;
    y = 1000;
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Java Reference Types

- In Java technology, beyond primitive types all others are reference types.
- A *reference variable* contains a *handle* to an object.
- For example:

```
public class MyDate {
  private int day = 1;
  private int month = 1;
  private int year = 2000;
  public MyDate(int day, int month, int year) { ... }
  public String toString() { ... }
}
```

```
public class TestMyDate {
  public static void main(String[] args) {
    MyDate today = new MyDate(22, 7, 1964);
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Constructing and Initializing Objects

- Calling `new` *`Xyz`*`()` performs the following actions:
  a. Memory is allocated for the object.
  b. Explicit attribute initialization is performed.
  c. A constructor is executed.
  d. The object reference is returned by the `new` operator.
- The reference to the object is assigned to a variable.
- An example is:

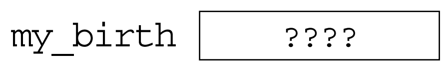
```
MyDate my_birth = new MyDate(22, 7, 1964);
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Memory Allocation and Layout

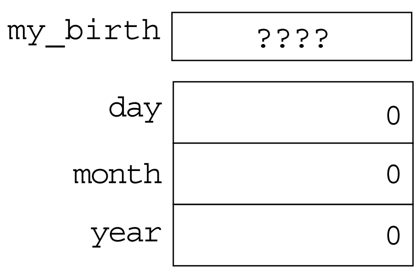
- A declaration allocates storage only for a reference:

```
MyDate my_birth = new MyDate(22, 7, 1964);
```



- Use the `new` operator to allocate space for `MyDate`:

```
MyDate my_birth = new MyDate(22, 7, 1964);
```



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Explicit Attribute Initialization

- Initialize the attributes as follows:

```
MyDate my_birth = new MyDate(22, 7, 1964);
```

my_birth	????
day	1
month	1
year	2000

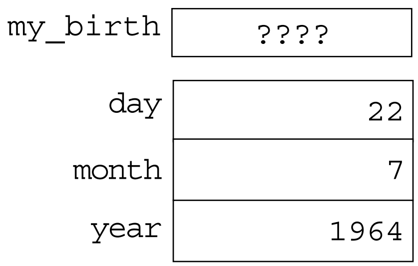
- The default values are taken from the attribute declaration in the class.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Executing the Constructor

- Execute the matching constructor as follows:

```
MyDate my_birth = new MyDate(22, 7, 1964);
```



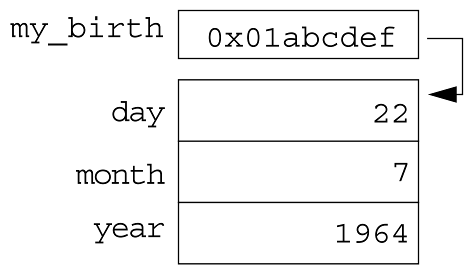
- In the case of an overloaded constructor, the first constructor can call another.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Assigning a Variable

- Assign the newly created object to the reference variable as follows:

```
MyDate my_birth = new MyDate(22, 7, 1964);
```

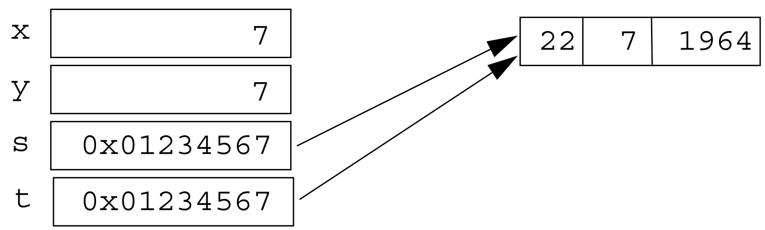


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Assigning References

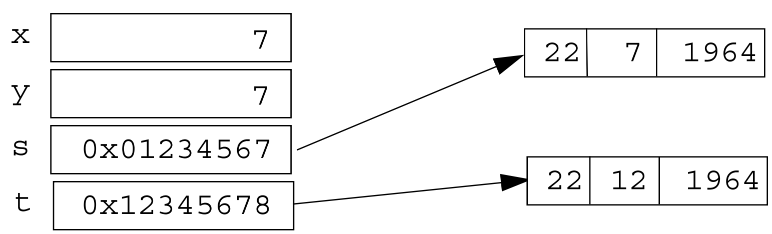
- Two variables refer to a single object:

```
int x = 7;
int y = x;
MyDate s = new MyDate(22, 7, 1964);
MyDate t = s;
```



- Reassignment makes two variables point to two objects:

```
t = new MyDate(22, 12, 1964);
```



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Pass-by-Value

- In a single virtual machine, 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 original object reference is never changed.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Pass-by-Value

```
public class PassTest {

  // Methods to change the current values
  public static void changeInt(int value) {
    value = 55;
  }
  public static void changeObjectRef(MyDate ref) {
    ref = new MyDate(1, 1, 2000);
  }
  public static void changeObjectAttr(MyDate ref){
    ref.setDay(4);
  }
```

Tommorow

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Pass-by-Value

```java

  public static void main(String args[]) {
    MyDate date;
    int val;

    // Assign the int
    val = 11;
    // Try to change it
    changeInt(val);
    // What is the current value?
    System.out.println("Int value is: " + val);
```

The result of this output is:

```
Int value is: 11
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Pass-by-Value

```
        // Assign the date
        date = new MyDate(22, 7, 1964);
        // Try to change it
        changeObjectRef(date);
        // What is the current value?
        System.out.println("MyDate: " + date);
```

The result of this output is:

```
MyDate: 22-7-1964
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Pass-by-Value

```

        // Now change the day attribute
        // through the object reference
        changeObjectAttr(date);
        // What is the current value?
        System.out.println("MyDate: " + date);
    }
}
```

The result of this output is:

```
MyDate: 4-7-1964
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `this` Reference

Here are a few uses of the `this` keyword:

- To resolve ambiguity between instance variables and parameters
- To pass the current object as a parameter to another method or constructor

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `this` Reference

```
public class MyDate {
  private int day = 1;
  private int month = 1;
  private int year = 2000;

  public MyDate(int day, int month, int year) {
    this.day   = day;
    this.month = month;
    this.year  = year;
  }
  public MyDate(MyDate date) {
    this.day   = date.day;
    this.month = date.month;
    this.year  = date.year;
  }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `this` Reference

```java

  public MyDate addDays(int moreDays) {
    MyDate newDate = new MyDate(this);
    newDate.day = newDate.day + moreDays;
    // Not Yet Implemented: wrap around code...
    return newDate;
  }
  public String toString() {
    return "" + day + "-" + month + "-" + year;
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `this` Reference

```
public class TestMyDate {
  public static void main(String[] args) {
    MyDate my_birth = new MyDate(22, 7, 1964);
    MyDate the_next_week = my_birth.addDays(7);

    System.out.println(the_next_week);
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Java Programming Language Coding Conventions

- Packages:

  `com.example.domain;`

- Classes, interfaces, and `enum` types:

  `SavingsAccount`

- Methods:

  `getAccount()`

- Variables:

  `currentCustomer`

- Constants:

  `HEAD_COUNT`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Java Programming Language Coding Conventions

- Control structures:

```
if ( condition ) {
  statement1;
} else {
  statement2;
}
```

- Spacing:
  - Use one statement per line.
  - Use two or four spaces for indentation.
- Comments:
  - Use // to comment inline code.
  - Use `/** documentation */` for class members.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 4

Expressions and Flow Control

Objectives

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

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

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

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

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?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Variables and Scope

Local variables are:

- Variables that are defined inside a method and are called *local*, *automatic*, *temporary*, or *stack* variables
- Variables that are created when the method is executed are destroyed when the method is exited

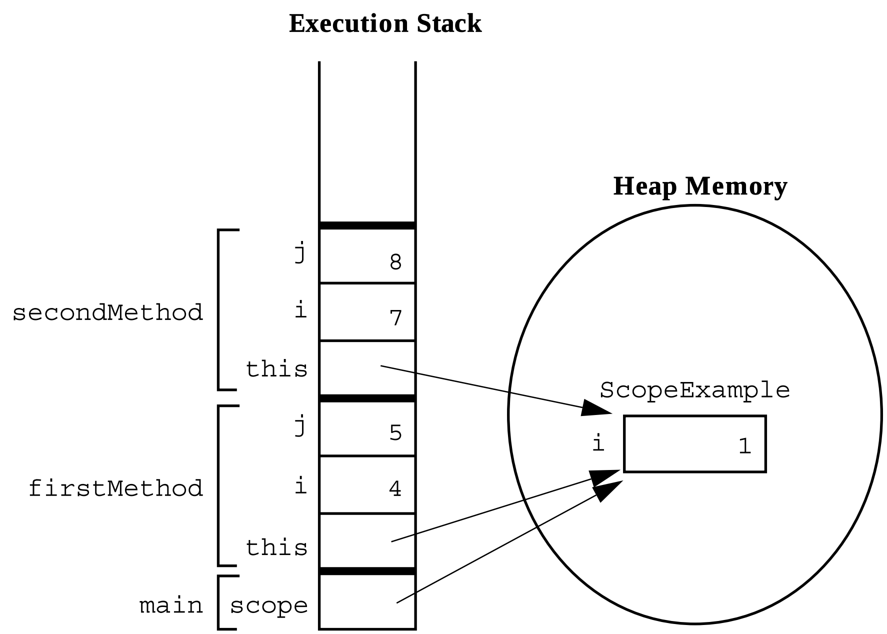
Variable initialization comprises the following:

- Local variables require explicit initialization.
- Instance variables are initialized automatically.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Variable Scope Example

```
public class ScopeExample {
  private int i=1;

  public void firstMethod() {
    int i=4, j=5;

    this.i = i + j;
    secondMethod(7);
  }
  public void secondMethod(int i) {
    int j=8;
    this.i = i + j;
  }
}


public class TestScoping {
  public static void main(String[] args) {
    ScopeExample scope = new ScopeExample();

    scope.firstMethod();
  }
}
```



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Variable Initialization

Variable	Value
`byte`	`0`
`short`	`0`
`int`	`0`
`long`	`0L`
`float`	`0.0F`
`double`	`0.0D`
`char`	`'\u0000'`
`boolean`	`false`
All reference types	`null`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Initialization Before Use Principle

The compiler will verify that local variables have been initialized before used.

```
3     public void doComputation() {
4       int x = (int)(Math.random() * 100);
5       int y;
6       int z;
7       if (x > 50) {
8         y = 9;
9       }
10      z = y + x;  // Possible use before initialization
11    }
```

```
javac TestInitBeforeUse.java
TestInitBeforeUse.java:10: variable y might not have been initialized
    z = y + x;  // Possible use before initialization
        ^
1 error
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Operator Precedence

<table>
<tr><th>Operators</th><th>Associative</th></tr>
<tr><td>++ -- + unary - unary ~ ! (<data_type>)</td><td>R to L</td></tr>
<tr><td>* / %</td><td>L to R</td></tr>
<tr><td>+ -</td><td>L to R</td></tr>
<tr><td><< >> >>></td><td>L to R</td></tr>
<tr><td>< > <= >= instanceof</td><td>L to R</td></tr>
<tr><td>== !=</td><td>L to R</td></tr>
<tr><td>&</td><td>L to R</td></tr>
<tr><td>^</td><td>L to R</td></tr>
<tr><td>|</td><td>L to R</td></tr>
<tr><td>&&</td><td>L to R</td></tr>
<tr><td>||</td><td>L to R</td></tr>
<tr><td><boolean_expr> ? <expr1> : <expr2></td><td>R to L</td></tr>
<tr><td>= *= /= %= += -= <<= >>= >>>= &= ^= |=</td><td>R to L</td></tr>
</table>

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Logical Operators

- The `boolean` operators are:

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

- The short-circuit `boolean` operators are:

```
&& – AND    || – OR
```

- You can use these operators as follows:

```
MyDate d = reservation.getDepartureDate();
if ( (d != null) && (d.day > 31) {
  // do something with d
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Bitwise Logical Operators

- The integer *bitwise* operators are:

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

- Byte-sized examples include:

```
  0 1 0 0 1 1 1 1          0 0 1 0 1 1 0 1
~                        & 0 1 0 0 1 1 1 1
-----------------        -----------------
  1 0 1 1 0 0 0 0          0 0 0 0 1 1 0 1

  0 0 1 0 1 1 0 1          0 0 1 0 1 1 0 1
^ 0 1 0 0 1 1 1 1        | 0 1 0 0 1 1 1 1
-----------------        -----------------
  0 1 1 0 0 0 1 0          0 1 1 0 1 1 1 1
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Right-Shift Operators >> and >>>

- *Arithmetic* or *signed* right shift (>>) operator:
  - Examples are:

```
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.
- *Logical* or *unsigned right-shift* (>>>) operator:
  - This operator is used for bit patterns.
  - The sign bit is not copied during the shift.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Left-Shift Operator <<

- Left-shift (<<) operator works as follows:

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


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Shift Operator Examples

1357 =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	1
-1357 =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	1	0	0	1	1
1357 >> 5 =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0
-1357 >> 5 =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1
1357 >>> 5 =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0
-1357 >>> 5 =	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1
1357 << 5 =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	1	0	0	0	0	0
-1357 << 5 =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	1	0	0	1	1	0	0	0	0	0

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

String Concatenation With +

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

```
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.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Casting

- If information might be lost in an assignment, the programmer must confirm the assignment with a cast.
- The assignment between `long` and `int` requires an explicit cast.

```
long bigValue = 99L;
int squashed = bigValue;        // Wrong, needs a cast
int squashed = (int) bigValue; // OK

int squashed = 99L;             // Wrong, needs a cast
int squashed = (int) 99L;       // OK, but...
int squashed = 99;              // default integer literal
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Promotion and Casting of Expressions

- Variables are promoted automatically 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.

```
long bigval = 6;        // 6 is an int type, OK
int smallval = 99L;     // 99L is a long, illegal

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


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Simple `if, else` Statements

The `if` statement syntax:

```
if ( <boolean_expression> )
  <statement_or_block>
```

Example:

```
if ( x < 10 )
  System.out.println("Are you finished yet?");
```

or (*recommended*):

```
if ( x < 10 ) {
  System.out.println("Are you finished yet?");
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Complex `if, else` Statements

The `if-else` statement syntax:

```
if ( <boolean_expression> )
  <statement_or_block>
else
  <statement_or_block>
```

Example:

```
if ( x < 10 ) {
  System.out.println("Are you finished yet?");
} else {
  System.out.println("Keep working...");
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Complex `if, else` Statements

The `if-else-if` statement syntax:

```
if ( <boolean_expression> )
  <statement_or_block>
else if ( <boolean_expression> )
  <statement_or_block>
```

Example:

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


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Switch Statements

The `switch` statement syntax:

```
switch ( <expression> ) {
  case <constant1>:
    <statement_or_block>*
    [break;]
  case <constant2>:
    <statement_or_block>*
    [break;]
  default:
    <statement_or_block>*
    [break;]
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Switch Statements

A `switch` statement example:

```
switch ( carModel ) {
  case DELUXE:
    addAirConditioning();
    addRadio();
    addWheels();
    addEngine();
    break;
  case STANDARD:
    addRadio();
    addWheels();
    addEngine();
    break;
  default:
    addWheels();
    addEngine();
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Switch Statements

This `switch` statement is equivalent to the previous example:

```
switch ( carModel ) {
  case DELUXE:
    addAirConditioning();
  case STANDARD:
    addRadio();
  default:
    addWheels();
    addEngine();
}
```

Without the `break` statements, the execution falls through each subsequent `case` clause.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Looping Statements

The `for` loop:

```
for ( <init_expr>; <test_expr>; <alter_expr> )
  <statement_or_block>
```

Example:

```
for ( int i = 0; i < 10; i++ )
  System.out.println(i + " squared is " + (i*i));
```

or (*recommended*):

```
for ( int i = 0; i < 10; i++ ) {
  System.out.println(i + " squared is " + (i*i));
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Looping Statements

The `while` loop:

```
while ( <test_expr> )
  <statement_or_block>
```

Example:

```
int i = 0;
while ( i < 10 ) {
  System.out.println(i + " squared is " + (i*i));
  i++;
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Looping Statements

The `do/while` loop:

```
do
  <statement_or_block>
while ( <test_expr> );
```

Example:

```
int i = 0;
do {
  System.out.println(i + " squared is " + (i*i));
  i++;
} while ( i < 10 );
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Special Loop Flow Control

- The **`break`** ***`[<label>]`***`;` command
- The **`continue`** ***`[<label>]`***`;` command
- The ***`<label>`*** **`:`** ***`<statement>`*** command, where ***`<statement>`*** should be a loop

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `break` Statement

```
do {
  statement;
  if ( condition ) {
    break;
  }
  statement;
} while ( test_expr );
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `continue` Statement

```
do {
  statement;
  if ( condition ) {
    continue;
  }
  statement;
} while ( test_expr );
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Using `break` Statements with Labels

```
outer:
  do {
    statement1;
    do {
      statement2;
      if ( condition ) {
        break outer;
      }
      statement3;
    } while ( test_expr );
    statement4;
  } while ( test_expr );
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Using `continue` Statements with Labels

```
test:
  do {
    statement1;
    do {
      statement2;
      if ( condition ) {
        continue test;
      }
      statement3;
    } while ( test_expr );
    statement4;
  } while ( test_expr );
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 5

Arrays

Objectives

- Declare and create arrays of primitive, class, or array types
- Explain why elements of an array are initialized
- Explain how to 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 to another

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Relevance

What is the purpose of an array?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Declaring Arrays

- Group data objects of the same type.
- Declare arrays of primitive or class types:

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

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

- Create space for a reference.
- An array is an object; it is created with `new`.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating Arrays

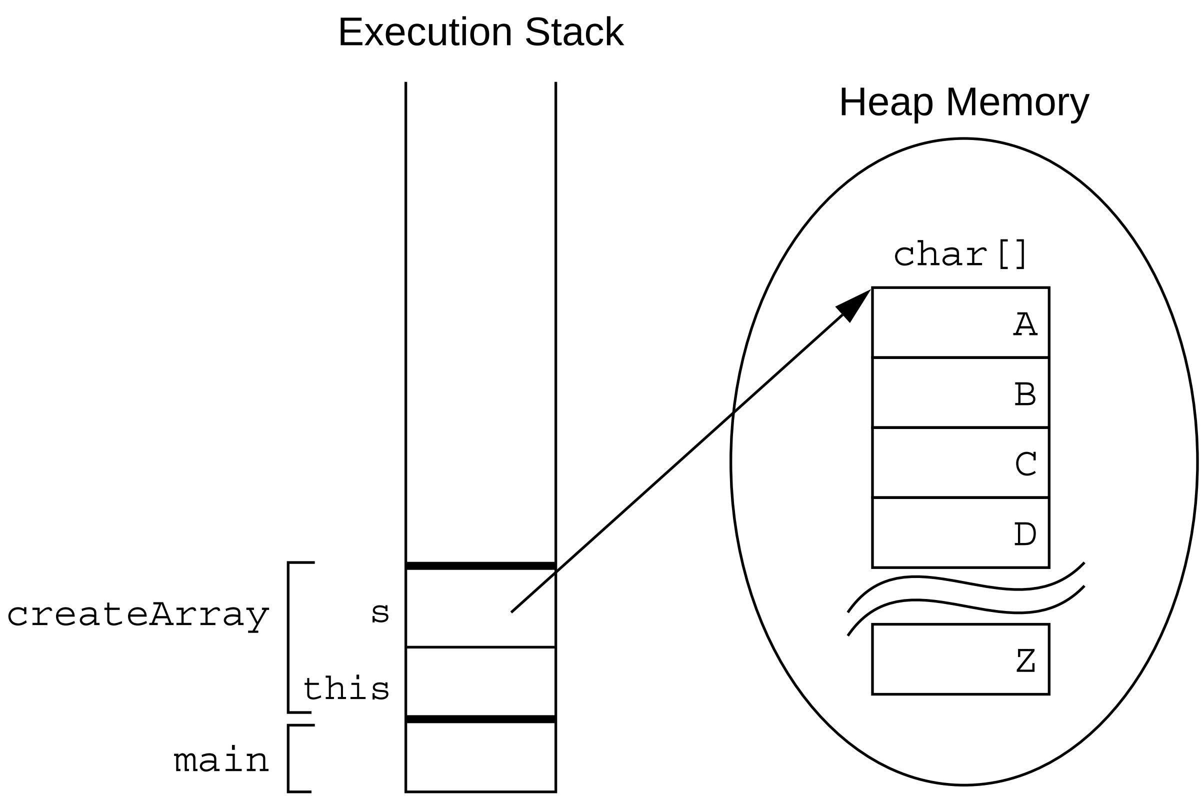
Use the `new` keyword to create an array object.

For example, a primitive (`char`) array:

```
public char[] createArray() {
  char[] s;

  s = new char[26];
  for ( int i=0; i<26; i++ ) {
    s[i] = (char) ('A' + i);
  }

  return s;
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating an Array of Character Primitives



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

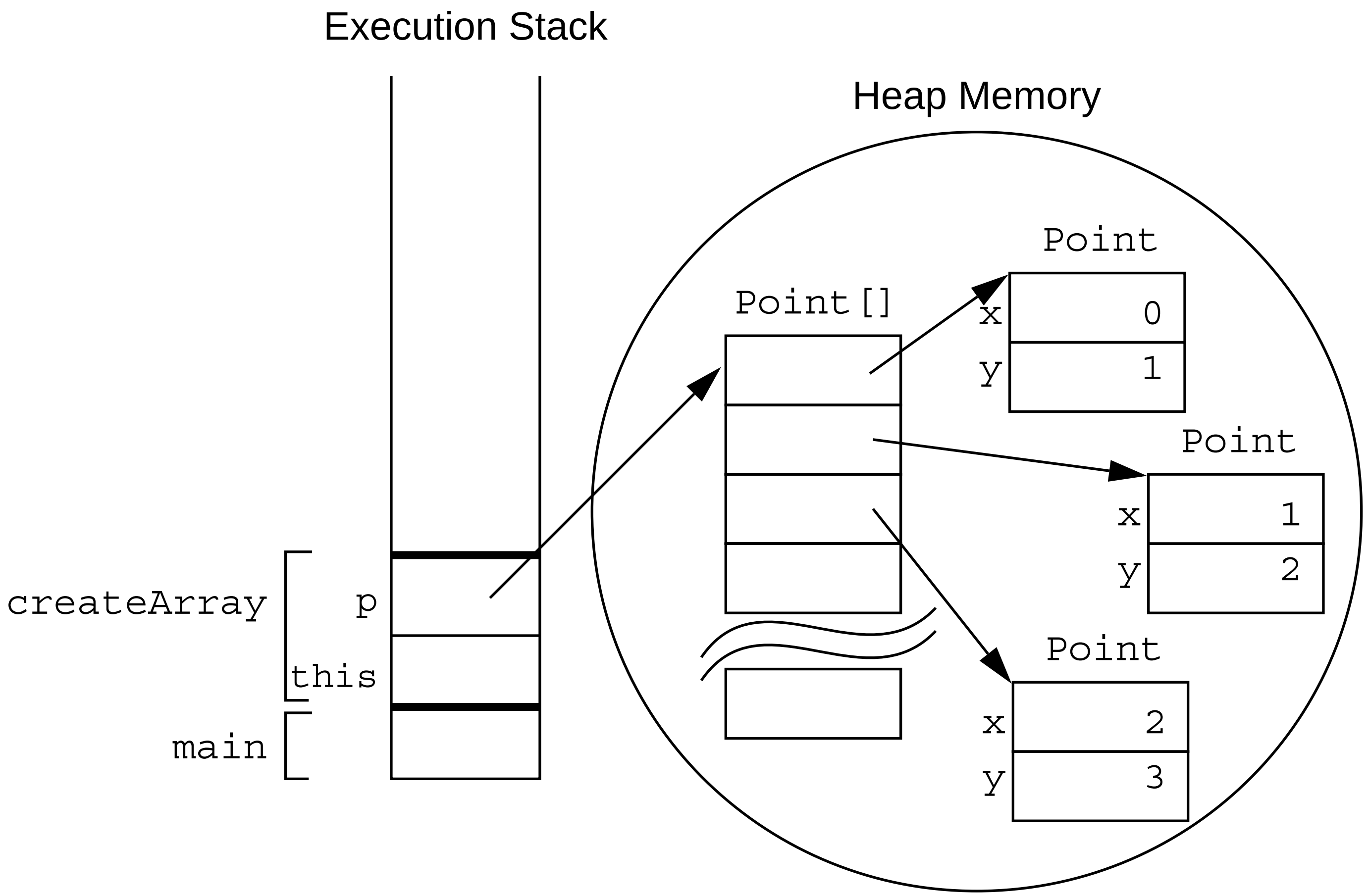
Creating Reference Arrays

Another example, an object array:

```
public Point[] createArray() {
  Point[] p;

  p = new Point[10];
  for ( int i=0; i<10; i++ ) {
    p[i] = new Point(i, i+1);
  }

  return p;
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating an Array of Character Primitives With `Point` Objects



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Initializing Arrays

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

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

```
String[] names = {
    "Georgianna",
    "Jen",
    "Simon"
};
```

```
MyDate[] dates;
dates = new MyDate[3];
dates[0] = new MyDate(22, 7, 1964);
dates[1] = new MyDate(1, 1, 2000);
dates[2] = new MyDate(22, 12, 1964);
```

```
MyDate[] dates = {
    new MyDate(22, 7, 1964),
    new MyDate(1, 1, 2000),
    new MyDate(22, 12, 1964)
};
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Multidimensional Arrays

Arrays of arrays:

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

int[][] twoDim = new int[][4]; // illegal
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Multidimensional 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];
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Array Bounds

All array subscripts begin at 0:

```
public void printElements(int[] list) {
  for (int i = 0; i < list.length; i++) {
    System.out.println(list[i]);
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Using the Enhanced `for` Loop

Java 2 Platform, Standard Edition (J2SE™) version 5.0 introduced an enhanced `for` loop for iterating over arrays:

```
public void printElements(int[] list) {
  for ( int element : list ) {
    System.out.println(element);
  }
}
```

The `for` loop can be read as *for each* `element` *in* `list` *do*.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Array Resizing

- You cannot resize an array.
- You can use the same reference variable to refer to an entirely new array, such as:

```
int[] myArray = new int[6];
myArray = new int[10];
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Copying Arrays

The `System.arraycopy()` method to copy arrays is:

```
//original array
int[] myArray = { 1, 2, 3, 4, 5, 6 };

// new larger array
int[] hold = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };

// copy all of the myArray array to the hold
// array, starting with the 0th index
System.arraycopy(myArray, 0, hold, 0, myArray.length);
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 6

Class Design

Objectives

- Define *inheritance*, *polymorphism*, *overloading*, *overriding*, and *virtual method invocation*
- Use the access modifiers `protected` and the default (*package-friendly)*
- Describe the concepts of constructor and method overloading
- Describe the complete object construction and initialization operation

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Relevance

How does the Java programming language support object inheritance?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Subclassing

The `Employee` class is shown here.

Employee
+name : String = ""   +salary : double   +birthDate : Date
+getDetails() : String

```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

  public String getDetails() {...}
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Subclassing

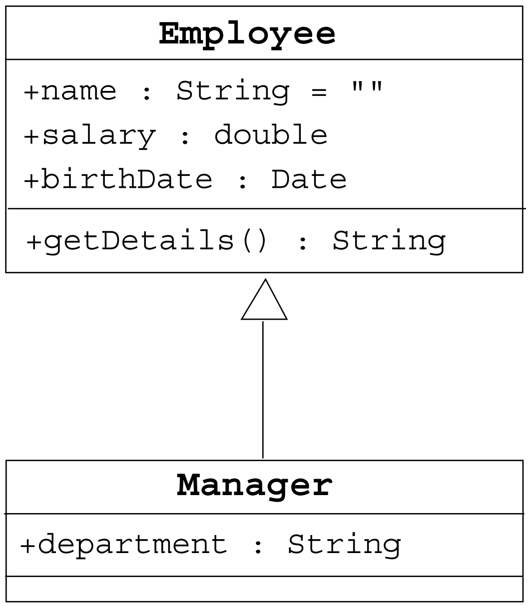
The `Manager` class is shown here.

Manager
+name : String = ""   +salary : double   +birthDate : Date   +department : String
+getDetails() : String

```
public class Manager {
  public String name = "";
  public double salary;
  public Date birthDate;
  public String department;

  public String getDetails() {...}
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Class Diagrams for `Employee` and `Manager` Using Inheritance




```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

  public String getDetails() {...}
}
```

```
public class Manager extends Employee {
  public String department;
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

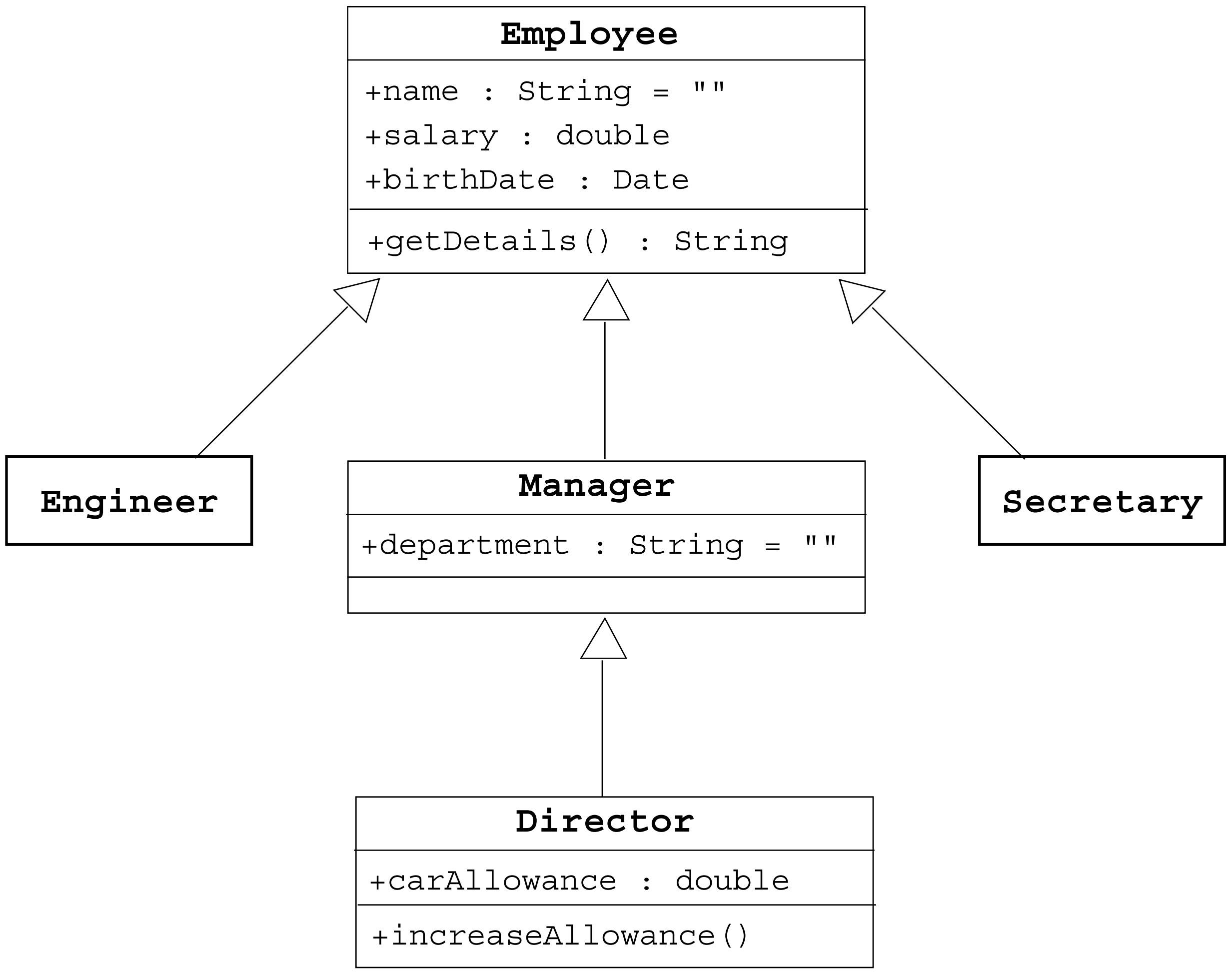
Single Inheritance

- When a class inherits from only one class, it is called *single inheritance.*
- *Interfaces* provide the benefits of multiple inheritance without drawbacks.
- Syntax of a Java class is as follows:

```
<modifier> class <name> [extends <superclass>] {
  <declaration>*
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Single Inheritance



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Access Control

Access modifiers on class member declarations are listed here.

Modifier	Same Class	Same Package	Subclass	Universe
`private`	Yes			
default	Yes	Yes		
`protected`	Yes	Yes	Yes	
`public`	Yes	Yes	Yes	Yes

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

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[1]
  - Argument list

1. In J2SE version 5, the return type can be a subclass of the overridden return type.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Overriding Methods

```
public class Employee {
  protected String name;
  protected double salary;
  protected Date birthDate;

  public String getDetails() {
    return “Name: “ + name + “\n” +
           “Salary: “ + salary;
  }
}
```

```
public class Manager extends Employee {
  protected String department;

  public String getDetails() {
    return “Name: “ + name + “\n” +
           “Salary: “ + salary + "\n" +
           “Manager of: “ + department;
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Overridden Methods Cannot Be Less Accessible

```
public class Parent {
  public void doSomething() {}
}
```

```
public class Child extends Parent {
  private void doSomething() {} // illegal
}
```

```
public class UseBoth {
  public void doOtherThing() {
    Parent p1 = new Parent();
    Parent p2 = new Child();
    p1.doSomething();
    p2.doSomething();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Invoking Overridden Methods

A subclass method may invoke a superclass method using the `super` keyword:

- The keyword `super` is used in a class to refer to its superclass.
- The keyword `super` is used to refer to the members of superclass, both data attributes and methods.
- Behavior invoked does not have to be in the superclass; it can be further up in the hierarchy.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Invoking Overridden Methods

```
public class Employee {
  private String name;
  private double salary;
  private Date birthDate;

  public String getDetails() {
    return "Name: " + name + "\nSalary: " + salary;
  }
}
```

```
public class Manager extends Employee {
  private String department;

  public String getDetails() {
    // call parent method
    return super.getDetails()
           + “\nDepartment: " + department;
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

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 can refer to objects of different forms.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Polymorphism

```
Employee e = new Manager(); // legal

// illegal attempt to assign Manager attribute
e.department = "Sales";
// the variable is declared as an Employee type,
// even though the Manager object has that attribute
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Virtual Method Invocation

- Virtual method invocation is performed as follows:

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

- Compile-time type and runtime type invocations have the following characteristics:
  - The method name must be a member of the declared variable type; in this case `Employee` has a method called `getDetails`.
  - The method implementation used is based on the runtime object's type; in this case the `Manager` class has an implementation of the `getDetails` method.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Heterogeneous Collections

- Collections of objects with the same class type are called *homogeneous* collections. For example:

```
MyDate[]  dates = new MyDate[2];
dates[0] = new MyDate(22, 12, 1964);
dates[1] = new MyDate(22, 7, 1964);
```

- Collections of objects with different class types are called *heterogeneous* collections. For example:

```
Employee [] staff = new Employee[1024];
staff[0] = new Manager();
staff[1] = new Employee();
staff[2] = new Engineer();
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Polymorphic Arguments

Because a `Manager` is an `Employee`, the following is valid:

```
public class TaxService {
  public TaxRate findTaxRate(Employee e) {
    // calculate the employee's tax rate
  }
}

// Meanwhile, elsewhere in the application class
TaxService taxSvc = new TaxService();
Manager m = new Manager();
TaxRate t = taxSvc.findTaxRate(m);
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `instanceof` Operator

```
public class Employee extends Object
public class Manager extends Employee
public class Engineer extends Employee
----------------------------------------

public void doSomething(Employee e) {
  if ( e instanceof Manager ) {
    // Process a Manager
  } else if ( e instanceof Engineer ) {
    // Process an Engineer
  } else {
    // Process any other type of Employee
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Casting Objects

```
public void doSomething(Employee e) {
  if ( e instanceof Manager ) {
    Manager m = (Manager) e;
    System.out.println("This is the manager of "
                       + m.getDepartment());
  }
  // rest of operation
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

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 *upward* in the 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.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Overloading Methods

- Use overloading 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.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Methods Using Variable Arguments

- Methods using *variable arguments* permit multiple number of arguments in methods.

  For example:

```
public class Statistics {
  public float average(int... nums) {
    int sum = 0;
    for ( int x : nums ) {
      sum += x;
    }
    return ((float) sum) / nums.length;
  }
}
```

- The *vararg* parameter is treated as an array. For example:

```
float gradePointAverage = stats.average(4, 3, 4);
float averageAge = stats.average(24, 32, 27, 18);
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Overloading Constructors

- As with methods, constructors can be overloaded.

  An example is:

```
public Employee(String name, double salary, Date DoB)
public Employee(String name, double salary)
public Employee(String name, Date DoB)
```

- Argument lists *must* differ.
- You can use the `this` reference at the first line of a constructor to call another constructor.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Overloading Constructors

```
public class Employee {
  private static final double BASE_SALARY = 15000.00;
  private String name;
  private double salary;
  private Date   birthDate;

  public Employee(String name, double salary, Date DoB) {
    this.name = name;
    this.salary = salary;
    this.birthDate = DoB;
  }
  public Employee(String name, double salary) {
    this(name, salary, null);
  }
  public Employee(String name, Date DoB) {
    this(name, BASE_SALARY, DoB);
  }
  // more Employee code...
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

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.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Invoking Parent Class Constructors

- To invoke a parent constructor, you must place a call to `super` in the first line of the constructor.
- You can call a specific parent constructor by the arguments that you use in the call to `super`.
- If no `this` or `super` call is used in a constructor, then the compiler adds an implicit call to `super()` that calls the parent no argument constructor (which could be the *default* constructor).

  If the parent class defines constructors, but does not provide a no-argument constructor, then a compiler error message is issued.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Invoking Parent Class Constructors

```java
public class Manager extends Employee {
  private String department;

  public Manager(String name, double salary, String dept) {
    super(name, salary);
    department = dept;
  }
  public Manager(String name, String dept) {
    super(name);
    department = dept;
  }
  public Manager(String dept) { // This code fails: no super()
    department = dept;
  }
  //more Manager code...
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Constructing and Initializing Objects: A Slight Reprise

Memory is allocated and default initialization occurs.

Instance variable initialization uses these steps recursively:

1. Bind constructor parameters.
2. If explicit `this()`, call recursively, and then skip to Step 5.
3. Call recursively the implicit or explicit `super` call, except for `Object`.
4. Execute the explicit instance variable initializers.
5. Execute the body of the current constructor.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Constructor and Initialization Examples

```
public class Object {
  public Object() {}
}
```

```
public class Employee extends Object {
  private String name;
  private double salary = 15000.00;
  private Date   birthDate;

  public Employee(String n, Date DoB) {
    // implicit super();
    name = n;
    birthDate = DoB;
  }
  public Employee(String n) {
    this(n, null);
  }
}
```

```
public class Manager extends Employee {
  private String department;

  public Manager(String n, String d) {
    super(n);
    department = d;
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Constructor and Initialization Examples

- `0` Basic initialization
  - `0.1` Allocate memory for the complete `Manager` object
  - `0.2` Initialize all instance variables to their default values (`0` or `null`)
- `1` Call constructor: `Manager("Joe Smith", "Sales")`
  - `1.1` Bind constructor parameters: `n="Joe Smith", d="Sales"`
  - `1.2` No explicit `this()` call
  - `1.3` Call `super(n)` for `Employee(String)`
    - `1.3.1` Bind constructor parameters: `n="Joe Smith"`
    - `1.3.2` Call `this(n, null)` for `Employee(String, Date)`
      - `1.3.2.1` Bind constructor parameters: `n="Joe Smith", DoB=null`
      - `1.3.2.2` No explicit `this()` call
      - `1.3.2.3` Call `super()` for `Object()`
        - `1.3.2.3.1` No binding necessary
        - `1.3.2.3.2` No `this()` call
        - `1.3.2.3.3` No `super()` call `(Object` is the root)
        - `1.3.2.3.4` No explicit variable initialization for `Object`
        - `1.3.2.3.5` No method body to call

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Constructor and Initialization Examples

- `1.3.2.4` Initialize explicit Employee variables: `salary=15000.00;`
- `1.3.2.5` Execute body: `name="Joe Smith"; date=null;`
- `1.3.3 - 1.3.4` Steps skipped
- `1.3.5` Execute body: No body in `Employee(String)`
- `1.4` No explicit initializers for `Manager`
- `1.5` Execute body: `department="Sales"`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `Object` Class

- The `Object` class is the root of all classes in Java.
- A class declaration with no `extends` clause implies `extends Object`. For example:

```
public class Employee {
  ...
}
```

is equivalent to:

```
public class Employee extends Object {
  ...
}
```

- Two important methods are:
  - `equals`
  - `toString`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `equals` Method

- The == operator determines if two references are identical to each other (that is, refer to the same object).
- The `equals` method determines if objects are *equal* but not necessarily identical.
- The `Object` implementation of the `equals` method uses the == operator.
- User classes can override the `equals` method to implement a domain-specific test for equality.
- Note: You should override the `hashCode` method if you override the `equals` method.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

An `equals` Example

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

  public MyDate(int day, int month, int year) {
    this.day   = day;
    this.month = month;
    this.year  = year;
  }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

An `equals` Example

```

  public boolean equals(Object o) {
    boolean result = false;
    if ( (o != null) && (o instanceof MyDate) ) {
      MyDate d = (MyDate) o;
      if ( (day == d.day) && (month == d.month)
           && (year == d.year) ) {
        result = true;
      }
    }
    return result;
  }

  public int hashCode() {
    return (day ^ month ^ year);
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

An `equals` Example

```
class TestEquals {
  public static void main(String[] args) {
    MyDate  date1 = new MyDate(14, 3, 1976);
    MyDate  date2 = new MyDate(14, 3, 1976);

    if ( date1 == date2 ) {
      System.out.println("date1 is identical to date2");
    } else {
      System.out.println("date1 is not identical to date2");
    }

    if ( date1.equals(date2) ) {
      System.out.println("date1 is equal to date2");
    } else {
      System.out.println("date1 is not equal to date2");
    }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

An `equals` Example

```
    System.out.println("set date2 = date1;");
    date2 = date1;

    if ( date1 == date2 ) {
      System.out.println("date1 is identical to date2");
    } else {
      System.out.println("date1 is not identical to date2");
    }
  }
}
```

This example generates the following output:

```
date1 is not identical to date2
date1 is equal to date2
set date2 = date1;
date1 is identical to date2
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `toString` Method

The `toString` method has the following characteristics:

- This method converts an object to a `String`.
- Use this method during string concatenation.
- Override this method to provide information about a user-defined object in readable format.
- Use the wrapper class's `toString` static method to convert primitive types to a `String`.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Wrapper Classes

Look at primitive data elements as objects.

Primitive Data Type	**Wrapper Class**
`boolean`	`Boolean`
`byte`	`Byte`
`char`	`Character`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Wrapper Classes

An example of a wrapper class is:

```
int pInt = 420;
Integer wInt = new Integer(pInt); // this is called boxing
int p2 = wInt.intValue(); // this is called unboxing
```

Other methods are:

```
int x = Integer.valueOf(str).intValue();
int x = Integer.parseInt(str);
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Autoboxing of Primitive Types

Autoboxing has the following description:

- Conversion of primitive types to the object equivalent
- Wrapper classes not always needed
- Example:

```
int pInt = 420;
Integer wInt = pInt; // this is called autoboxing
int p2 = wInt; // this is called autounboxing
```

- Language feature used most often when dealing with collections
- Wrapped primitives also usable in arithmetic expressions
- Performance loss when using autoboxing

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 7

Advanced Class Features

Objectives

- Create static variables, methods, and initializers
- Create final classes, methods, and variables
- Create and use enumerated types
- Use the static import statement
- Create abstract classes and methods
- Create and use an interface

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Relevance

- How can you create a constant?
- How can you declare data that is shared by all instances of a given class?
- How can you keep a class or method from being subclassed or overridden?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

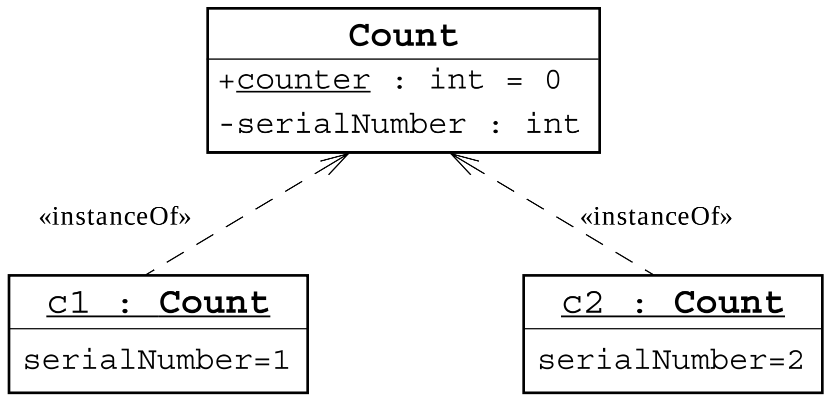
The `static` Keyword

- The `static` keyword is used as a modifier on variables, methods, and nested classes.
- The `static` keyword declares the attribute or method is associated with the class as a whole rather than any particular instance of that class.
- Thus static members are often called *class members*, such as *class attributes* or *class methods*.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Class Attributes

Class attributes are shared among all instances of a class:




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

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


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Class Attributes

If the static member is `public`:

```
public class Count1 {
  private int serialNumber;
  public static int counter = 0;
  public Count1() {
    counter++;
    serialNumber = counter;
  }
}
```

it can be accessed from outside the class without an instance:

```
public class OtherClass {
  public void incrementNumber() {
    Count1.counter++;
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Class Methods

You can create `static` methods:

```
public class Count2 {
  private int serialNumber;
  private static int counter = 0;

  public static int getTotalCount() {
    return counter;
  }

  public Count2() {
    counter++;
    serialNumber = counter;
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Class Methods

You can invoke `static` methods without any instance of the class to which it belongs:

```
public class TestCounter {
  public static void main(String[] args) {
    System.out.println("Number of counter is "
                       + Count2.getTotalCount());
    Count2 counter = new Count2();
    System.out.println("Number of counter is "
                       + Count2.getTotalCount());
  }
}
```

The output of the `TestCounter` program is:

```
Number of counter is 0
Number of counter is 1
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Class Methods

Static methods cannot access instance variables:

```
public class Count3 {
  private int serialNumber;
  private static int counter = 0;

  public static int getSerialNumber() {
    return serialNumber;  // COMPILER ERROR!
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Static Initializers

- A class can contain code in a *static block* that does not exist within a method body.
- Static block code executes once only, when the class is loaded.
- Usually, a static block is used to initialize static (class) attributes.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Static Initializers

```
public class Count4 {
  public static int counter;
  static {
    counter = Integer.getInteger("myApp.Count4.counter").intValue();
  }
}
```

```
public class TestStaticInit {
  public static void main(String[] args) {
    System.out.println("counter = "+ Count4.counter);
  }
}
```

The output of the `TestStaticInit` program is:

```
java -DmyApp.Count4.counter=47 TestStaticInit
counter = 47
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `final` Keyword

- You cannot subclass a `final` class.
- You cannot override a `final` method.
- A `final` variable is a constant.
- You can set a `final` variable once only, but that assignment can occur independently of the declaration; this is called a *blank final variable*.
  - A blank final instance attribute must be set in every constructor.
  - A blank final method variable must be set in the method body before being used.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Final Variables

Constants are static final variables.

```
public class Bank {
  private static final double  DEFAULT_INTEREST_RATE = 3.2;
  ... // more declarations
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Blank Final Variables

```
public class Customer {

  private final long customerID;

  public Customer() {
    customerID = createID();
  }

  public long getID() {
    return customerID;
  }

  private long createID() {
    return ... // generate new ID
  }

  // more declarations

}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Old-Style Enumerated Type Idiom

Enumerated types are a common idiom in programming.

```
package cards.domain;

public class PlayingCard {

  // pseudo enumerated type
  public static final int SUIT_SPADES   = 0;
  public static final int SUIT_HEARTS   = 1;
  public static final int SUIT_CLUBS    = 2;
  public static final int SUIT_DIAMONDS = 3;

  private int suit;
  private int rank;

  public PlayingCard(int suit, int rank) {
    this.suit = suit;
    this.rank = rank;
  }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Old-Style Enumerated Type Idiom

```
public String getSuitName() {
  String name = "";
  switch ( suit ) {
    case SUIT_SPADES:
      name = "Spades";
      break;
    case SUIT_HEARTS:
      name = "Hearts";
      break;
    case SUIT_CLUBS:
      name = "Clubs";
      break;
    case SUIT_DIAMONDS:
      name = "Diamonds";
      break;
    default:
      System.err.println("Invalid suit.");
  }
  return name;
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Old-Style Enumerated Type Idiom

Old-style idiom is not type-safe:

```
package cards.tests;

import cards.domain.PlayingCard;

public class TestPlayingCard {
  public static void main(String[] args) {

    PlayingCard card1
      = new PlayingCard(PlayingCard.SUIT_SPADES, 2);
    System.out.println("card1 is the " + card1.getRank()
                       + " of " + card1.getSuitName());

    // You can create a playing card with a bogus suit.
    PlayingCard card2 = new PlayingCard(47, 2);
    System.out.println("card2 is the " + card2.getRank()
                       + " of " + card2.getSuitName());
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Old-Style Enumerated Type Idiom

This enumerated type idiom has several problems:

- Not type-safe
- No namespace
- Brittle character
- Uninformative printed values

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The New Enumerated Type

Now you can create type-safe enumerated types:

```
package cards.domain;

public enum Suit {
  SPADES,
  HEARTS,
  CLUBS,
  DIAMONDS
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The New Enumerated Type

Using enumerated types is easy:

```
package cards.domain;

public class PlayingCard {

  private Suit suit;
  private int rank;

  public PlayingCard(Suit suit, int rank) {
    this.suit = suit;
    this.rank = rank;
  }

  public Suit getSuit() {
    return suit;
  }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The New Enumerated Type

```
  public String getSuitName() {
    String name = "";
    switch ( suit ) {
      case SPADES:
        name = "Spades";
        break;
      case HEARTS:
        name = "Hearts";
        break;
      case CLUBS:
        name = "Clubs";
        break;
      case DIAMONDS:
        name = "Diamonds";
        break;
      default:
      // No need for error checking as the Suit
      // enum is finite.
    }
    return name;
  }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The New Enumerated Type

Enumerated types are type-safe:

```
package cards.tests;

import cards.domain.PlayingCard;
import cards.domain.Suit;

public class TestPlayingCard {
  public static void main(String[] args) {

    PlayingCard card1
      = new PlayingCard(Suit.SPADES, 2);
    System.out.println("card1 is the " + card1.getRank()
                       + " of " + card1.getSuitName());

    // PlayingCard card2 = new PlayingCard(47, 2);
    // This will not compile.
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Advanced Enumerated Types

Enumerated types can have attributes and methods:

```
package cards.domain;

public enum Suit {
  SPADES   ("Spades"),
  HEARTS   ("Hearts"),
  CLUBS    ("Clubs"),
  DIAMONDS ("Diamonds");

  private final String name;

  private Suit(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Advanced Enumerated Types

Public methods on enumerated types are accessible:

```
package cards.tests;

import cards.domain.PlayingCard;
import cards.domain.Suit;

public class TestPlayingCard {
  public static void main(String[] args) {

    PlayingCard card1
      = new PlayingCard(Suit.SPADES, 2);
    System.out.println(“card1 is the “ + card1.getRank()
                       + “ of “ + card1.getSuit().getName());

    // NewPlayingCard card2 = new NewPlayingCard(47, 2);
    // This will not compile.
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Static Imports

- A *static import* imports the static members from a class:

```
import static <pkg_list>.<class_name>.<member_name>;
OR
import static <pkg_list>.<class_name>.*;
```

- A static import imports members individually or collectively:

```
import static cards.domain.Suit.SPADES;
OR
import static cards.domain.Suit.*;
```

- There is no need to qualify the static constants:

```
PlayingCard card1 = new PlayingCard(SPADES, 2);
```

- *Use this feature sparingly.*

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Static Imports

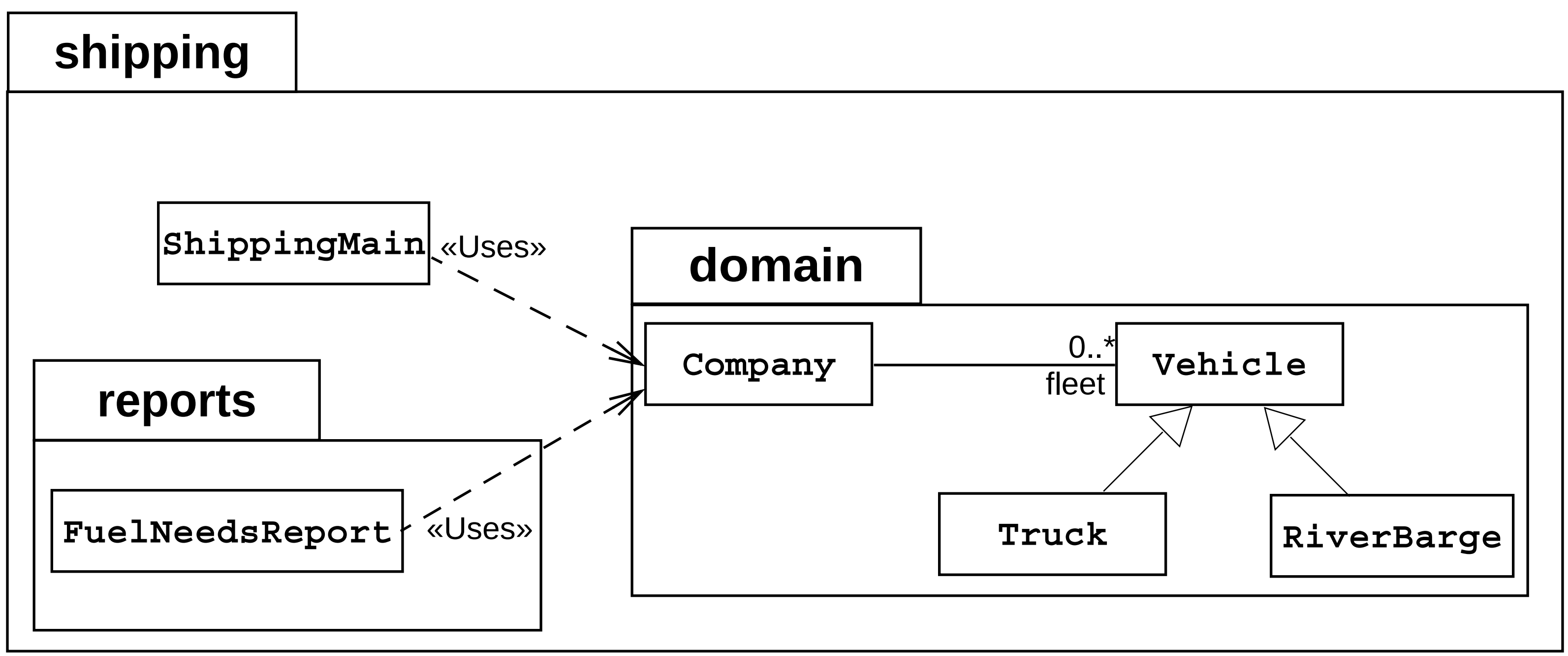
An example of a static import is:

```
package cards.tests;

import cards.domain.PlayingCard;
import static cards.domain.Suit.*;

public class TestPlayingCard {
  public static void main(String[] args) {

    PlayingCard card1 = new PlayingCard(SPADES, 2);
    System.out.println("card1 is the " + card1.getRank()
                       + " of " + card1.getSuit().getName());

    // NewPlayingCard card2 = new NewPlayingCard(47, 2);
    // This will not compile.
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Abstract Classes

The design of the Shipping system looks like this:



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Abstract Classes

Fleet initialization code is shown here:

```
public class ShippingMain {
  public static void main(String[] args) {
    Company c = new Company();

    // populate the company with a fleet of vehicles
    c.addVehicle( new Truck(10000.0) );
    c.addVehicle( new Truck(15000.0) );
    c.addVehicle( new RiverBarge(500000.0) );
    c.addVehicle( new Truck(9500.0) );
    c.addVehicle( new RiverBarge(750000.0) );

    FuelNeedsReport report = new FuelNeedsReport(c);
    report.generateText(System.out);
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Abstract Classes

```
public class FuelNeedsReport {
  private Company company;

  public FuelNeedsReport(Company company) {
    this.company = company;
  }

  public void generateText(PrintStream output) {
    Vehicle1 v;
    double fuel;
    double total_fuel = 0.0;

    for ( int i = 0; i < company.getFleetSize(); i++ ) {
      v = company.getVehicle(i);

```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

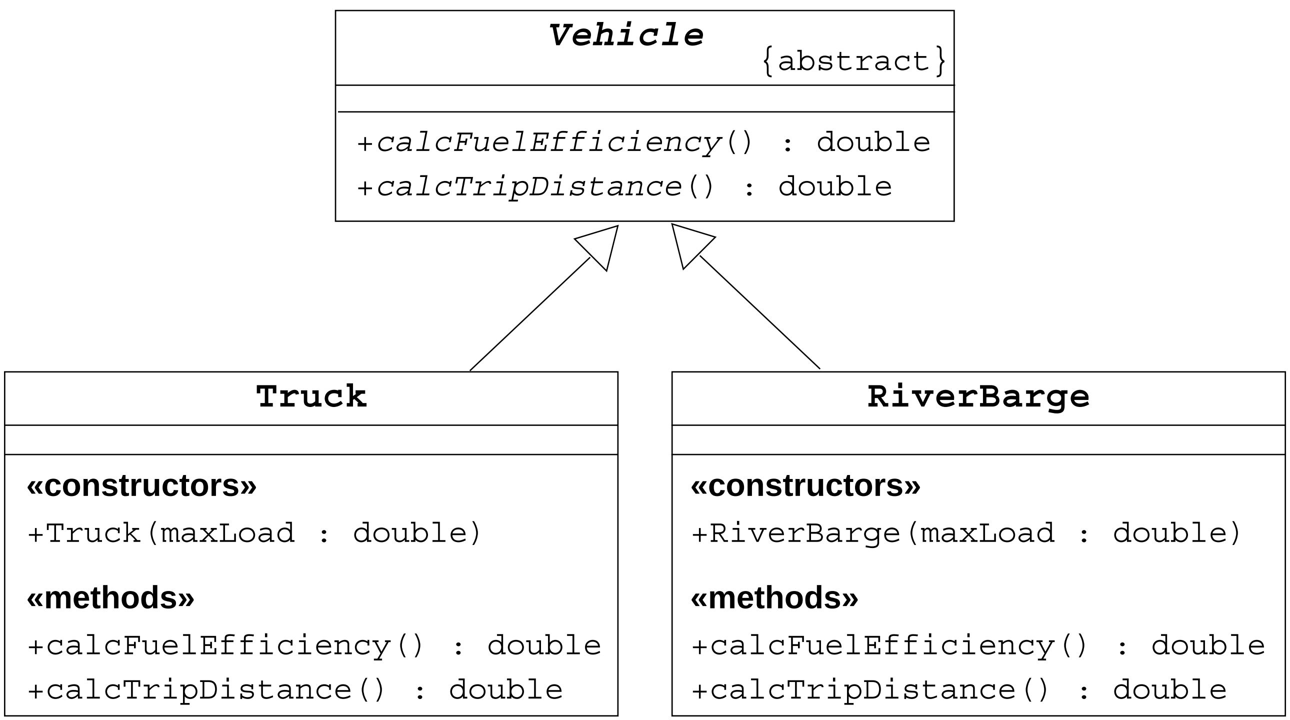
Abstract Classes

```
      // Calculate the fuel needed for this trip
      fuel = v.calcTripDistance() / v.calcFuelEfficency();

      output.println("Vehicle " + v.getName() + " needs "
                     + fuel + " liters of fuel.");
      total_fuel += fuel;
    }
    output.println("Total fuel needs is " + total_fuel + " liters.");
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Solution

An abstract class models a class of objects in which the full implementation is not known but is supplied by the concrete subclasses.



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Solution

The declaration of the `Vehicle` class is:

```
public abstract class Vehicle {
  public abstract double calcFuelEfficiency();
  public abstract double calcTripDistance();
}
```

The `Truck` class must create an implementation:

```
public class Truck extends Vehicle {
  public Truck(double maxLoad) {...}
  public double calcFuelEfficiency() {
    /* calculate the fuel consumption of a truck at a given load */
  }
  public double calcTripDistance() {
    /* calculate the distance of this trip on highway */
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Solution

Likewise, the `RiverBarge` class must create an implementation:

```
public class RiverBarge extends Vehicle {
  public RiverBarge(double maxLoad) {...}
  public double calcFuelEfficiency() {
    /* calculate the fuel efficiency of a river barge */
  }
  public double calcTripDistance() {
    /* calculate the distance of this trip along the river-ways */
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

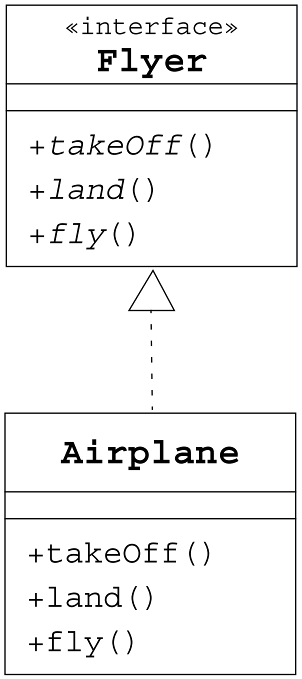
Interfaces

- A *public interface* is a contract between *client code* and the class that implements that interface.
- A Java *interface* is a formal declaration of such a contract in which all methods contain no implementation.
- Many unrelated classes can implement the same interface.
- A class can implement many unrelated interfaces.
- Syntax of a Java class is as follows:

```
<modifier> class <name> [extends <superclass>]
           [implements <interface> [,<interface>]* ] {
  <member_declaration>*
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Flyer Example




```
public interface Flyer {
  public void takeOff();
  public void land();
  public void fly();
}
```

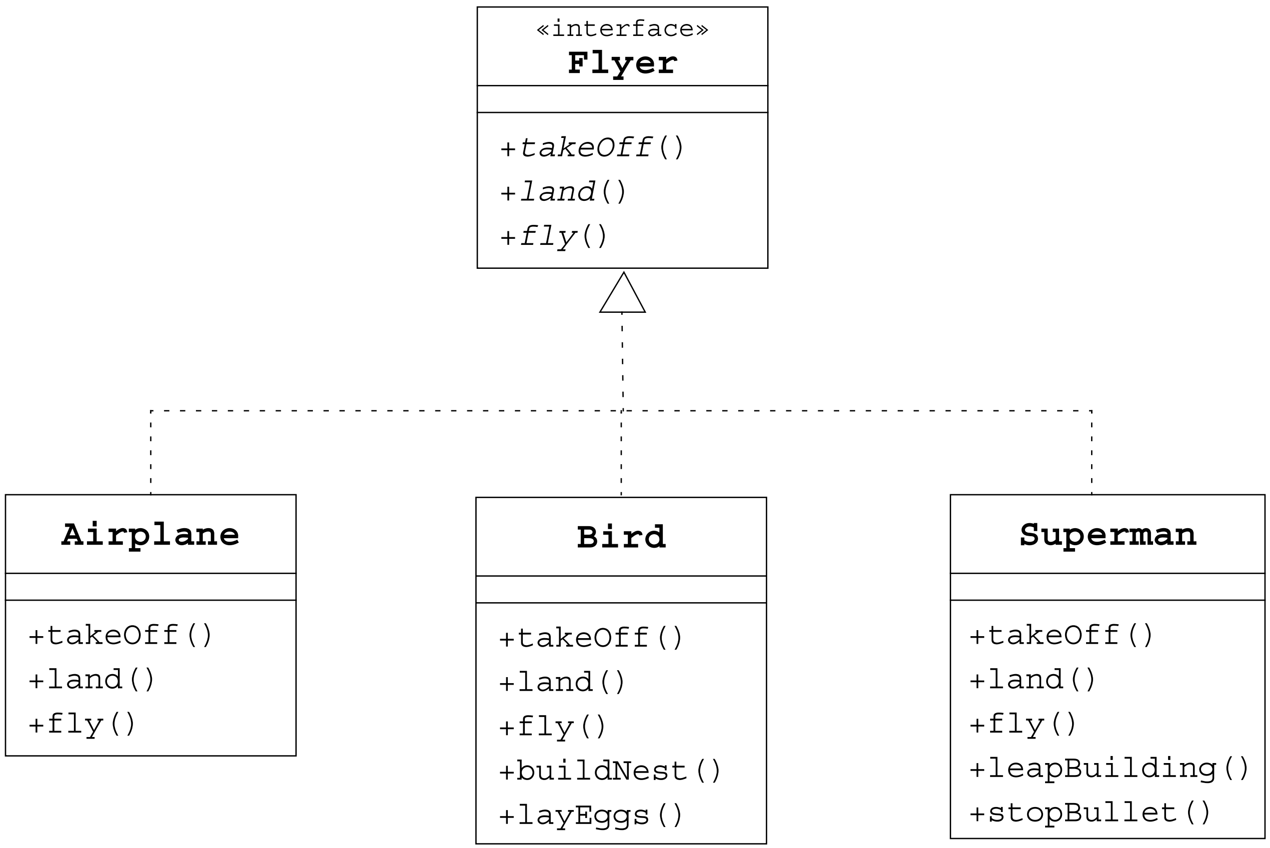

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Flyer Example

```
public class Airplane implements Flyer {
  public void takeOff() {
    // accelerate until lift-off
    // raise landing gear
  }
  public void land() {
    // lower landing gear
    // decelerate and lower flaps until touch-down
    // apply brakes
  }
  public void fly() {
    // keep those engines running
  }
}
```

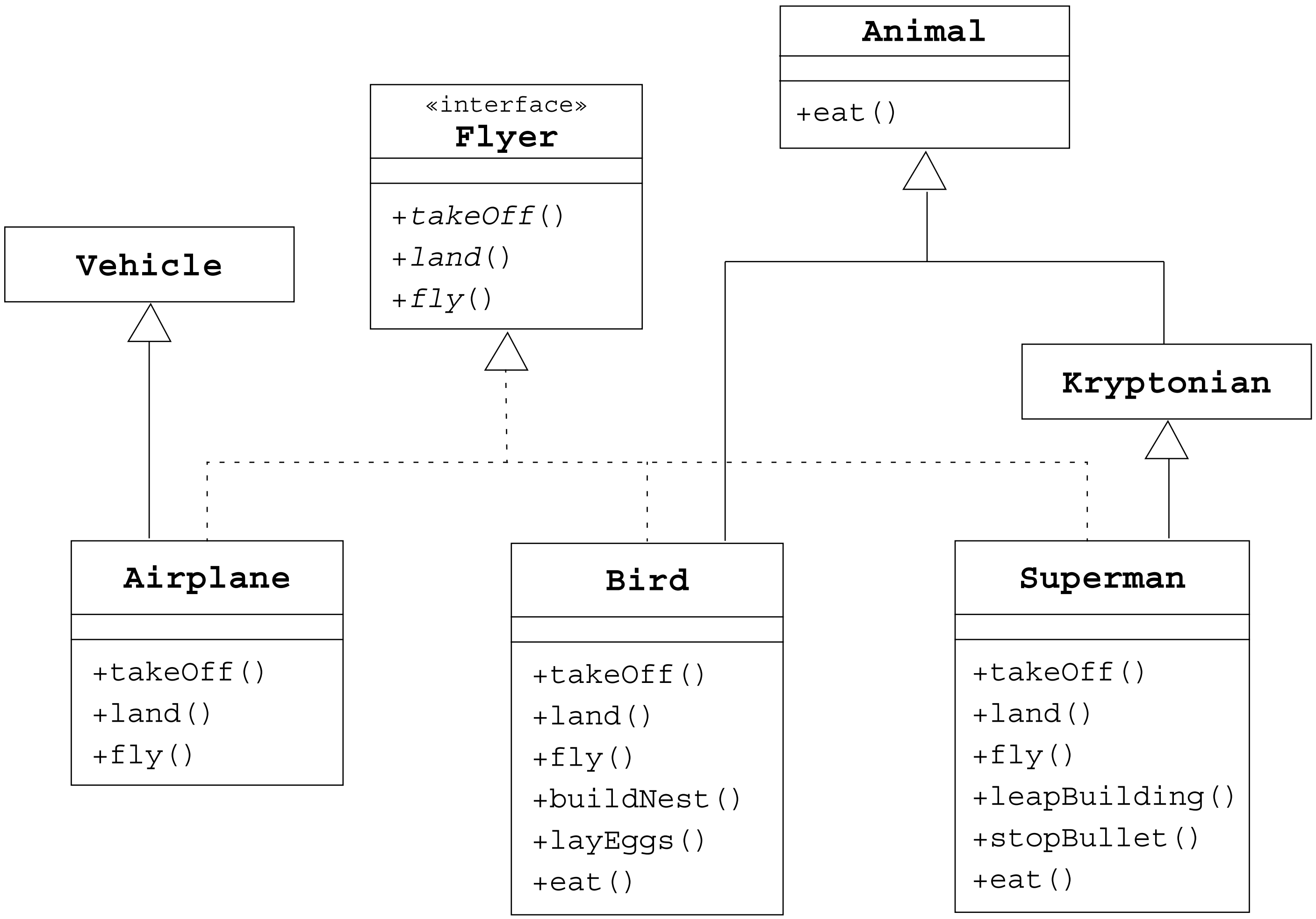

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Flyer Example



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Flyer Example



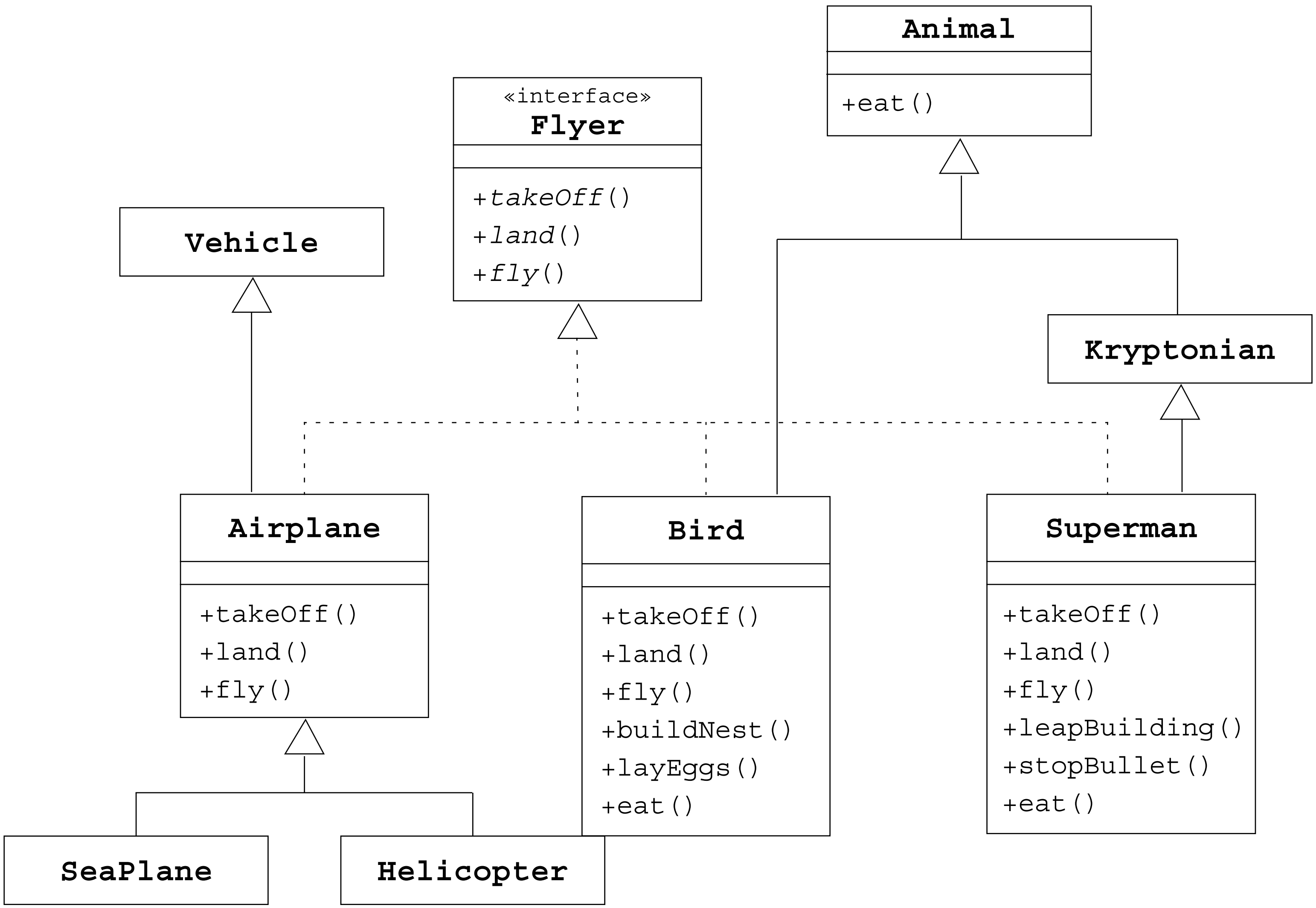
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Flyer Example

```
public class Bird extends Animal implements Flyer {
  public void takeOff()   { /* take-off implementation  */ }
  public void land()      { /* landing implementation   */ }
  public void fly()       { /* fly implementation       */ }
  public void buildNest() { /* nest building behavior   */ }
  public void layEggs()   { /* egg laying behavior      */ }
  public void eat()       { /* override eating behavior */ }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Flyer Example



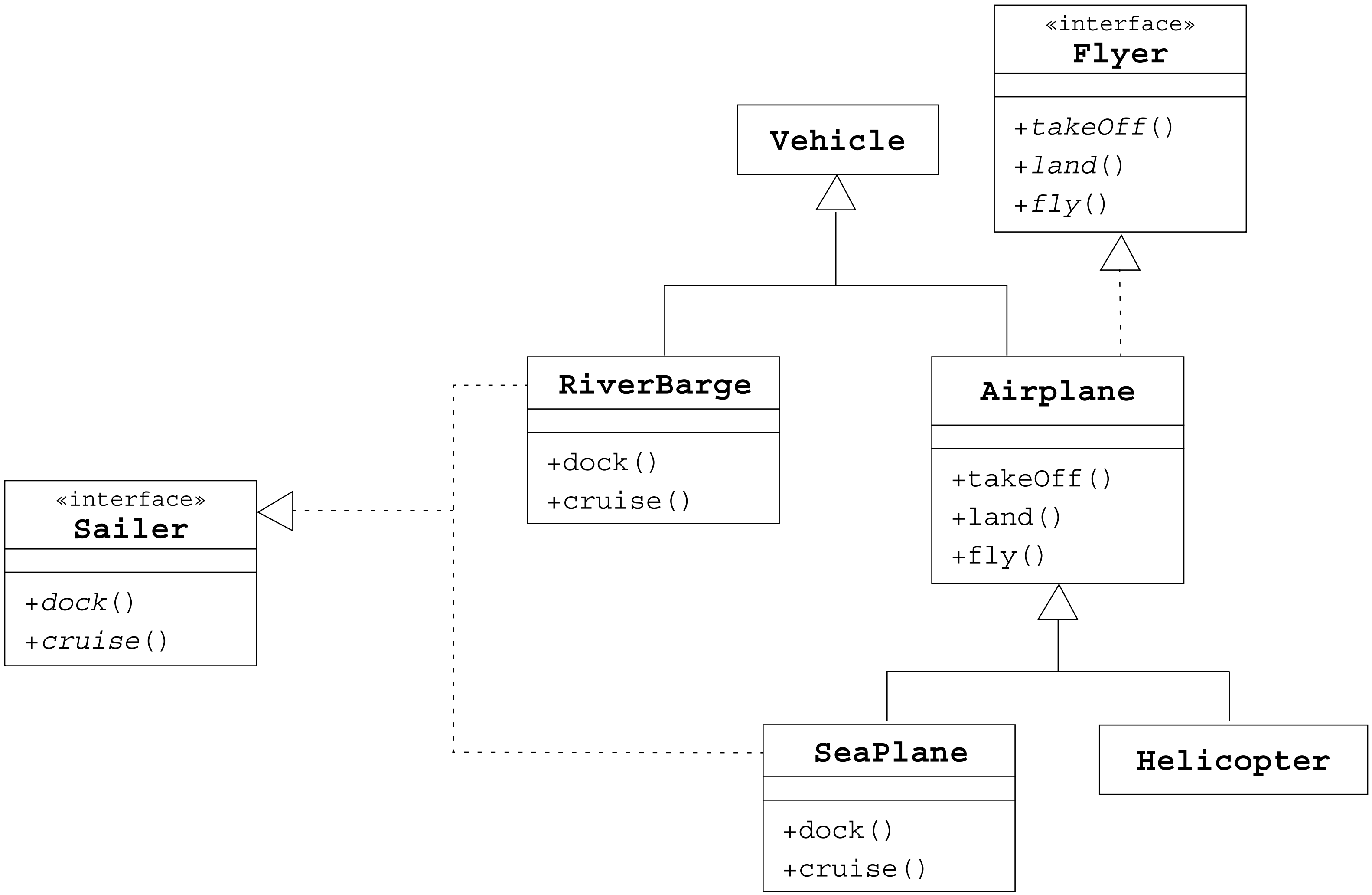
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Flyer Example

```
public class Airport {
  public static void main(String[] args) {
    Airport metropolisAirport = new Airport();
    Helicopter copter = new Helicopter();
    SeaPlane sPlane = new SeaPlane();

    metropolisAirport.givePermissionToLand(copter);
    metropolisAirport.givePermissionToLand(sPlane);
  }

  private void givePermissionToLand(Flyer f) {
    f.land();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Multiple Interface Example



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Multiple Interface Example

```
public class Harbor {
  public static void main(String[] args) {
    Harbor bostonHarbor = new Harbor();
    RiverBarge barge = new RiverBarge();
    SeaPlane sPlane = new SeaPlane();

    bostonHarbor.givePermissionToDock(barge);
    bostonHarbor.givePermissionToDock(sPlane);
  }

  private void givePermissionToDock(Sailer s) {
    s.dock();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Uses of Interfaces

Interface uses include the following:

- 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
- Simulating multiple inheritance by declaring a class that implements several interfaces

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 8

Exceptions and Assertions

Objectives

- Define exceptions
- Use `try`, `catch`, and `finally` statements
- Describe exception categories
- Identify common exceptions
- Develop programs to handle your own exceptions
- Use assertions
- Distinguish appropriate and inappropriate uses of assertions
- Enable assertions at runtime

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Relevance

- In most programming languages, how do you resolve runtime errors?
- If you make assumptions about the way your code works, and those assumptions are wrong, what might happen?
- Is it always necessary or desirable to expend CPU power testing assertions in production programs?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Exceptions and Assertions

- Exceptions handle unexpected situations – Illegal argument, network failure, or file not found
- Assertions document and test programming assumptions – *This can never be negative here*
- Assertion tests can be removed entirely from code at runtime, so the code is not slowed down at all.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Exceptions

- Conditions that can readily occur in a correct program are *checked exceptions*.

  These are represented by the `Exception` class.
- Severe problems that normally are treated as fatal or situations that probably reflect program bugs are *unchecked exceptions*.

  Fatal situations are represented by the `Error` class.

  Probable bugs are represented by the `RuntimeException` class.
- The API documentation shows checked exceptions that can be thrown from a method.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Exception Example

```
public class AddArguments {
  public static void main(String args[]) {
    int sum = 0;
    for ( String arg : args ) {
      sum += Integer.parseInt(arg);
    }
    System.out.println("Sum = " + sum);
  }
}
```

```
java AddArguments 1 2 3 4
Sum = 10
```

```
java AddArguments 1 two 3.0 4
Exception in thread "main" java.lang.NumberFormatException: For input string: "two"
   at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
   at java.lang.Integer.parseInt(Integer.java:447)
   at java.lang.Integer.parseInt(Integer.java:497)
   at AddArguments.main(AddArguments.java:5)
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `try-catch` Statement

```
public class AddArguments2 {
  public static void main(String args[]) {
    try {
      int sum = 0;
      for ( String arg : args ) {
        sum += Integer.parseInt(arg);
      }
      System.out.println("Sum = " + sum);
    } catch (NumberFormatException nfe) {
      System.err.println("One of the command-line "
                         + "arguments is not an integer.");
    }
  }
}
```

```
java AddArguments2 1 two 3.0 4
One of the command-line arguments is not an integer.
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `try-catch` Statement

```
public class AddArguments3 {
  public static void main(String args[]) {
    int sum = 0;
    for ( String arg : args ) {
      try {
        sum += Integer.parseInt(arg);
      } catch (NumberFormatException nfe) {
        System.err.println("[" + arg + "] is not an integer"
                           + " and will not be included in the sum.");
      }
    }
    System.out.println("Sum = " + sum);
  }
}
```

```
java AddArguments3 1 two 3.0 4
[two] is not an integer and will not be included in the sum.
[3.0] is not an integer and will not be included in the sum.
Sum = 5
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `try-catch` Statement

A `try-catch` statement can use multiple catch clauses:

```
try {
  // code that might throw one or more exceptions

} catch (MyException e1) {
  // code to execute if a MyException exception is thrown

} catch (MyOtherException e2) {
  // code to execute if a MyOtherException exception is thrown

} catch (Exception e3) {
  // code to execute if any other exception is thrown
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

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.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

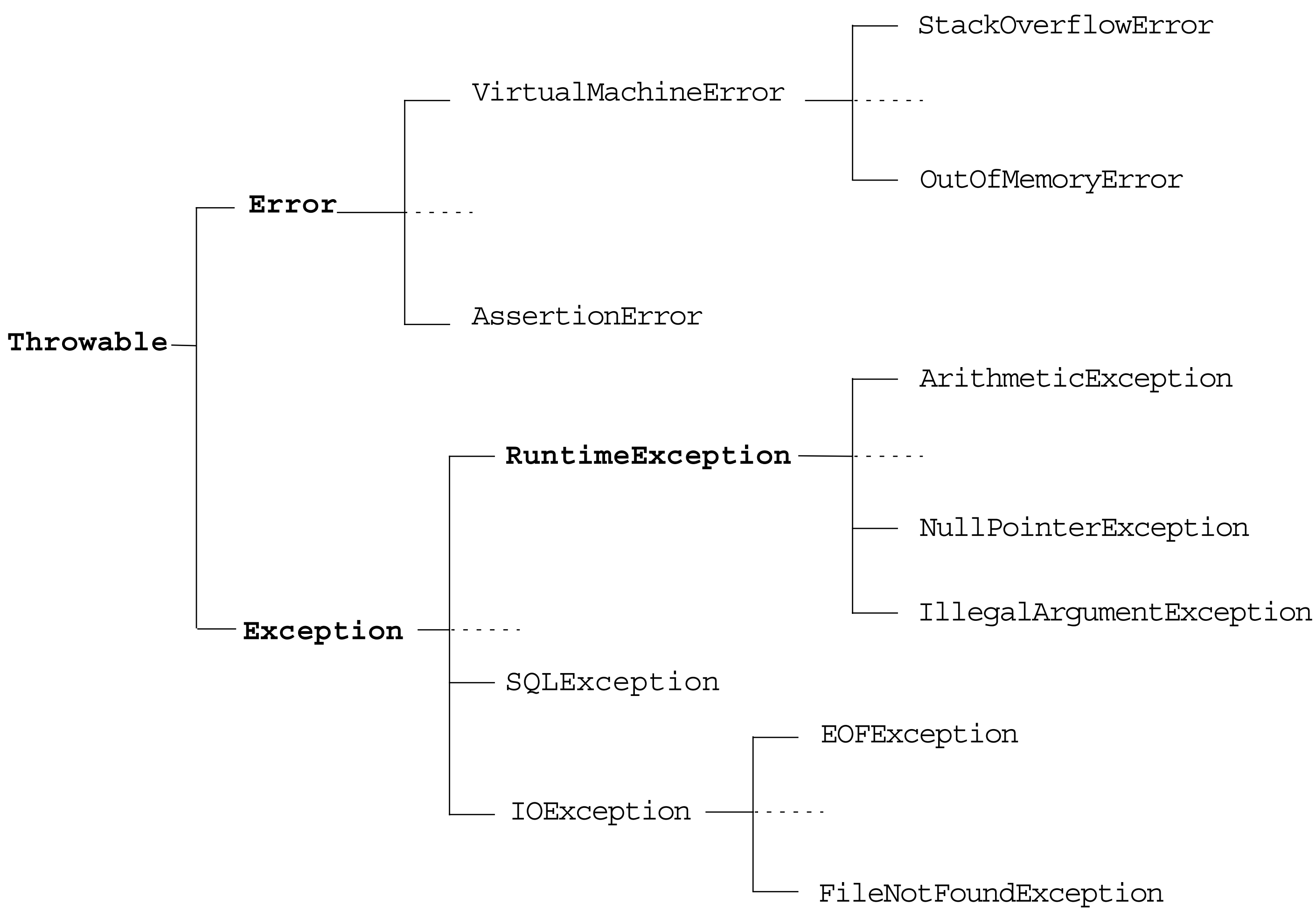
The `finally` Clause

The `finally` clause defines a block of code that *always* executes.

```
try {
  startFaucet();
  waterLawn();
} catch (BrokenPipeException e) {
  logProblem(e);
} finally {
  stopFaucet();
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Exception Categories



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Common Exceptions

- `NullPointerException`
- `FileNotFoundException`
- `NumberFormatException`
- `ArithmeticException`
- `SecurityException`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Handle or Declare Rule

Use the handle or declare rule as follows:

- Handle the exception by using the `try-catch-finally` block.
- Declare that the code causes an exception by using the `throws` clause.

```
void trouble() throws IOException { ... }
void trouble() throws IOException, MyException { ... }
```

Other Principles

- You do not need to declare runtime exceptions or errors.
- You can choose to handle runtime exceptions.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Method Overriding and Exceptions

The overriding method can throw:

- No exceptions
- One or more of the exceptions thrown by the overridden method
- One or more subclasses of the exceptions thrown by the overridden method

The overriding method cannot throw:

- Additional exceptions not thrown by the overridden method
- Superclasses of the exceptions thrown by the overridden method

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Method Overriding and Exceptions

```
public class TestA {
  public void methodA() throws IOException {
    // do some file manipulation
  }
}
```

```
public class TestB1 extends TestA {
  public void methodA() throws EOFException {
    // do some file manipulation
  }
}
```

```
public class TestB2 extends TestA {
  public void methodA() throws Exception { // WRONG
    // do some file manipulation
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating Your Own Exceptions

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

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

  public int getPort() {
    return port;
  }
}
```

Use the `getMessage` method, inherited from the `Exception` class, to get the reason for which the exception was made.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Handling a User-Defined Exception

A method can throw a user-defined, checked exception:

```
public void connectMe(String serverName)
       throws ServerTimedOutException {
  boolean successful;
  int portToConnect = 80;

  successful = open(serverName, portToConnect);

  if ( ! successful ) {
    throw new ServerTimedOutException("Could not connect",
                                      portToConnect);
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Handling a User-Defined Exception

Another method can use a `try-catch` block to capture user-defined exceptions:

```
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.getMessage() +
                           " connecting to port " + e1.getPort());
      }
    }
  }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Assertions

- Syntax of an assertion is:

```
assert <boolean_expression> ;
assert <boolean_expression> : <detail_expression> ;
```

- If `<boolean_expression>` evaluates `false`, then an `AssertionError` is thrown.
- The second argument is converted to a string and used as descriptive text in the `AssertionError` message.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Recommended Uses of Assertions

Use assertions to document and verify the assumptions and internal logic of a single method:

- Internal invariants
- Control flow invariants
- Postconditions and class invariants

Inappropriate Uses of Assertions

- Do not use assertions to check the parameters of a public method.
- Do not use methods in the assertion check that can cause side-effects.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Internal Invariants

The problem is:

```
if (x > 0) {
  // do this
} else {
  // do that
}
```

The solution is:

```
if (x > 0) {
  // do this
} else {
  assert ( x == 0 );
  // do that, unless x is negative
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Control Flow Invariants

For example:

```
switch (suit) {
    case Suit.CLUBS: // ...
      break;
    case Suit.DIAMONDS: // ...
      break;
    case Suit.HEARTS: // ...
      break;
    case Suit.SPADES: // ...
      break;
    default: assert false : "Unknown playing card suit";
      break;
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Postconditions and Class Invariants

For example:

```
public Object pop() {
  int size = this.getElementCount();
  if (size == 0) {
    throw new RuntimeException("Attempt to pop from empty stack");
  }

  Object result = /* code to retrieve the popped element */ ;

  // test the postcondition
  assert (this.getElementCount() == size - 1);

  return result;
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Controlling Runtime Evaluation of Assertions

- If assertion checking is disabled, the code runs as fast as if the check was never there.
- Assertion checks are disabled by default. Enable assertions with the following commands:

  ```
  java -enableassertions MyProgram
  ```

  or:

  ```
  java -ea MyProgram
  ```

- Assertion checking can be controlled on class, package, and package hierarchy bases, see: `docs/guide/language/assert.html`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 9

Text-Based Applications

Objectives

- Write a program that uses command-line arguments and system properties
- Write a program that reads from *standard input*
- Describe the C-type formatted input and output
- Write a program that can create, read, and write files
- Describe the basic hierarchy of collections in the Java 2 Software Development Kit (Java 2 SDK)
- Write a program that uses sets and lists
- Write a program to iterate over a collection
- Write a program that uses generic collections

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Relevance

- It is often the case that certain elements of a program should not be hard-coded, such as file names or the name of a database. How can a program be coded to supply these elements at runtime?
- Simple arrays are far too static for most collections (that is, a fixed number of elements). What Java technology features exist to support more flexible collections?
- Besides computation, what are key elements of any text-based application?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Command-Line Arguments

- Any Java technology application can use command-line arguments.
- These string arguments are placed on the command line to launch the Java interpreter, after the class name:

```
java TestArgs arg1 arg2 "another arg"
```

- Each command-line argument is placed in the `args` array that is passed to the static `main` method:

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


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Command-Line Arguments

```
public class TestArgs {
  public static void main(String[] args) {
    for ( int i = 0; i < args.length; i++ ) {
      System.out.println("args[" + i + "] is '" + args[i] + "'");
    }
  }
}
```

Example execution:

```
java TestArgs arg1 arg2 "another arg"
args[0] is 'arg1'
args[1] is 'arg2'
args[2] is 'another arg'
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

System Properties

- System properties are a feature that replaces the concept of *environment variables* (which are platform-specific).
- The `System.getProperties` method returns a `Properties` object.
- The `getProperty` method returns a `String` representing the value of the named property.
- Use the `-D` option to include a new property.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `Properties` Class

- The `Properties` class implements a mapping of names to values (a `String` to `String` map).
- The `propertyNames` method returns an `Enumeration` of all property names.
- The `getProperty` method returns a `String` representing the value of the named property.
- You can also read and write a properties collection into a file using `load` and `store`.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `Properties` Class

```
import java.util.Properties;
import java.util.Enumeration;

public class TestProperties {
  public static void main(String[] args) {
    Properties props = System.getProperties();
    Enumeration propNames = props.propertyNames();

    while ( propNames.hasMoreElements() ) {
      String propName = (String) propNames.nextElement();
      String property = props.getProperty(propName);
      System.out.println("property ’" + propName
                         + "’ is ’" + property + "’");
    }
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `Properties` Class

Here is an example test run of this program:

```
java -DmyProp=theValue TestProperties
```

Here is the (partial) output:

```
property 'java.version' is '1.5.0-rc'
property 'java.compiler' is 'NONE'
property 'path.separator' is ':'
property 'file.separator' is '/'
property 'user.home' is '/home/basham'
property 'java.specification.vendor' is 'Sun Microsystems Inc.'
property 'user.language' is 'en'
property 'user.name' is 'basham'
property 'myProp' is 'theValue'
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Console I/O

- The variable `System.out` enables you to write to *standard output*.

  It is an object of type `PrintStream`.
- The variable `System.in` enables you to read from *standard input*.

  It is an object of type `InputStream`.
- The variable `System.err` enables you to write to *standard error*.

  It is an object of type `PrintStream`.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Writing to Standard Output

- The `println` methods print the argument and a newline character (`\n`).
- The `print` methods print the argument without a newline character.
- The `print` and `println` methods are overloaded for most primitive types (`boolean`, `char`, `int`, `long`, `float`, and `double`) and for `char[]`, `Object`, and `String`.
- The `print(Object)` and `println(Object)` methods call the `toString` method on the argument.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Reading From Standard Input

```
import java.io.*;

public class KeyboardInput {
  public static void main (String args[]) {
    String s;
    // Create a buffered reader to read
    // each line from the keyboard.
    InputStreamReader ir
      = new InputStreamReader(System.in);
    BufferedReader in = new BufferedReader(ir);

    System.out.println("Unix: Type ctrl-d to exit." +
                       "\nWindows: Type ctrl-z to exit");
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Reading From Standard Input

```
      try {
        // Read each input line and echo it to the screen.
        s = in.readLine();
        while ( s != null ) {
          System.out.println("Read: " + s);
          s = in.readLine();
        }

        // Close the buffered reader.
        in.close();
      } catch (IOException e) { // Catch any IO exceptions.
        e.printStackTrace();
      }
    }
  }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Simple Formatted Output

- You can use the formatting functionality as follows

```
out.printf("name count\n");
String s = String.format("%s %5d%n", user, total);
```

- Common formatting codes are listed in this table.

Code	**Description**
`%s`	Formats the argument as a string, usually by calling the `toString` method on the object.
`%d %o %x`	Formats an integer, as a decimal, octal, or hexadecimal value.
`%f %g`	Formats a floating point number. The `%g` code uses scientific notation.
`%n`	Inserts a newline character to the string or stream.
`%%`	Inserts the `%` character to the string or stream.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Simple Formatted Input

- The Scanner API provides a formatted input function.
- A Scanner can be used with console input streams as well as file or network streams.
- You can read console input as follows:

```
import java.io.*;
import java.util.Scanner;
public class ScanTest {
  public static void main(String [] args) {
    Scanner s = new Scanner(System.in);
    String param = s.next();
    System.out.println("the param 1" + param);
    int value = s.nextInt();
    System.out.println("second param" + value);
    s.close();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Files and File I/O

The `java.io` package enables you to do the following:

- Create `File` objects
- Manipulate `File` objects
- Read and write to file streams

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating a New `File` Object

The `File` class provides several utilities:

- `File myFile;`
- `myFile = new File("myfile.txt");`
- `myFile = new File("MyDocs", "myfile.txt");`

Directories are treated just like files in Java; the `File` class supports methods for retrieving an array of files in the directory, as follows:

```
File myDir = new File("MyDocs");
myFile = new File(myDir, "myfile.txt");
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `File` Tests and Utilities

- File information:

```
String getName()
String getPath()
String getAbsolutePath()
String getParent()
long lastModified()
long length()
```

- File modification:

```
boolean renameTo(File newName)
boolean delete()
```

- Directory utilities:

```
boolean mkdir()
String[] list()
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `File` Tests and Utilities

- File tests:

```
boolean exists()
boolean canWrite()
boolean canRead()
boolean isFile()
boolean isDirectory()
boolean isAbsolute();
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

File Stream I/O

- For file input:
  - Use the `FileReader` class to read characters.
  - Use the `BufferedReader` class to use the `readLine` method.
- For file output:
  - Use the `FileWriter` class to write characters.
  - Use the `PrintWriter` class to use the `print` and `println` methods.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

File Stream I/O

A file input example is:

```
import java.io.*;
public class ReadFile {
  public static void main (String[] args) {
    // Create file
    File file = new File(args[0]);

    try {
      // Create a buffered reader
      // to read each line from a file.
      BufferedReader in
        = new BufferedReader(new FileReader(file));
      String s;

```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

File Stream I/O

```
        // Read each line from the file and echo it to the screen.
        s = in.readLine();
        while ( s != null ) {
          System.out.println("Read: " + s);
          s = in.readLine();
        }
        // Close the buffered reader
        in.close();

      } catch (FileNotFoundException e1) {
        // If this file does not exist
        System.err.println("File not found: " + file);

      } catch (IOException e2) {
        // Catch any other IO exceptions.
        e2.printStackTrace();
      }
    }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

File Output Example

```
import java.io.*;

public class WriteFile {
  public static void main (String[] args) {
    // Create file
    File file = new File(args[0]);

    try {
      // Create a buffered reader to read each line from standard in.
      InputStreamReader isr
        = new InputStreamReader(System.in);
      BufferedReader in
        = new BufferedReader(isr);
      // Create a print writer on this file.
      PrintWriter out
        = new PrintWriter(new FileWriter(file));
      String s;
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

File Output Example

```
        System.out.print("Enter file text.  ");
        System.out.println("[Type ctrl-d to stop.]");

        // Read each input line and echo it to the screen.
        while ((s = in.readLine()) != null) {
          out.println(s);
        }

        // Close the buffered reader and the file print writer.
        in.close();
        out.close();

      } catch (IOException e) {
      // Catch any IO exceptions.
        e.printStackTrace();
      }
    }
}
```

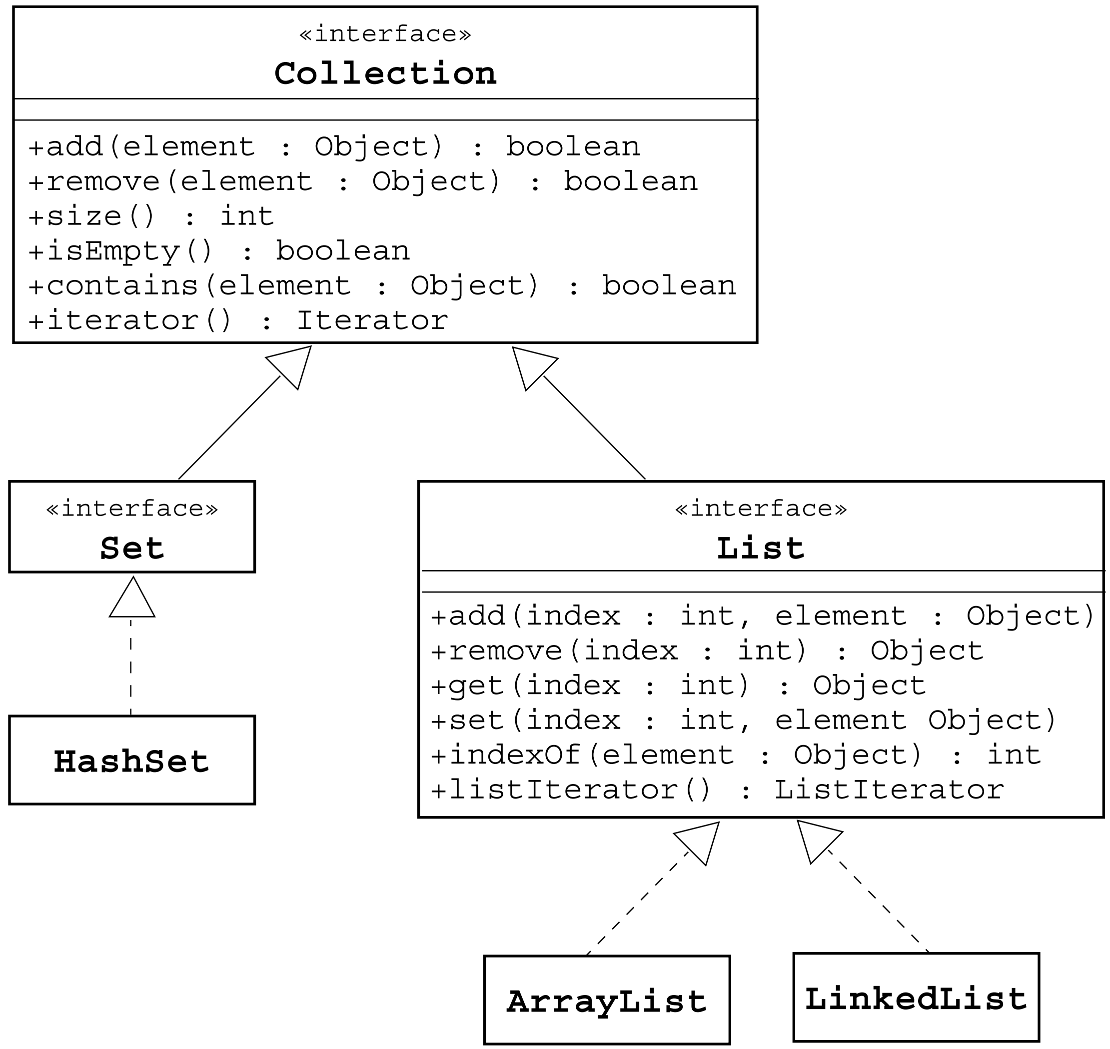

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Collections API

- A *collection* is a single object representing a group of objects known as its elements.
- The Collections API contains interfaces that group objects as one of the following:
  - `Collection` – A group of objects called elements; any specific ordering (or lack of) and allowance of duplicates is specified by each implementation
  - `Set` – An unordered collection; no duplicates are permitted
  - `List` – An ordered collection; duplicates are permitted

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Collections API



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

A Set Example

```
import java.util.*;

public class SetExample {
  public static void main(String[] args) {
    Set set = new HashSet();
    set.add("one");
    set.add("second");
    set.add("3rd");
    set.add(new Integer(4));
    set.add(new Float(5.0F));
    set.add("second");        // duplicate, not added
    set.add(new Integer(4));  // duplicate, not added
    System.out.println(set);
  }
}
```

The output generated from this program is:

```
[one, second, 5.0, 3rd, 4]
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

A `List` Example

```
import java.util.*

public class ListExample {
  public static void main(String[] args) {
    List list = new ArrayList();
    list.add("one");
    list.add("second");
    list.add("3rd");
    list.add(new Integer(4));
    list.add(new Float(5.0F));
    list.add("second");        // duplicate, is added
    list.add(new Integer(4));  // duplicate, is added
    System.out.println(list);
  }
}
```

The output generated from this program is:

```
[one, second, 3rd, 4, 5.0, second, 4]
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Collections in JDK™ Version 1.1

Collections in the Java Development Kit (JDK™) include:

- The class `Vector` implements the `List` interface.
- The class `Stack` is a subclass of `Vector` and supports the `push`, `pop`, and `peek` methods.
- The class `Hashtable` implements the `Map` interface.
- The `Enumeration` interface is a variation on the `Iterator` interface.

  An enumeration is returned by the `elements` method in `Vector`, `Stack`, and `Hashtable`.
- Classes are thread-safe, and therefore, *heavy weight*.
- These classes also support generics.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Generics

Generics are described as follows:

- Provides compile-time type safety
- Eliminates the need for casts

Before Generics

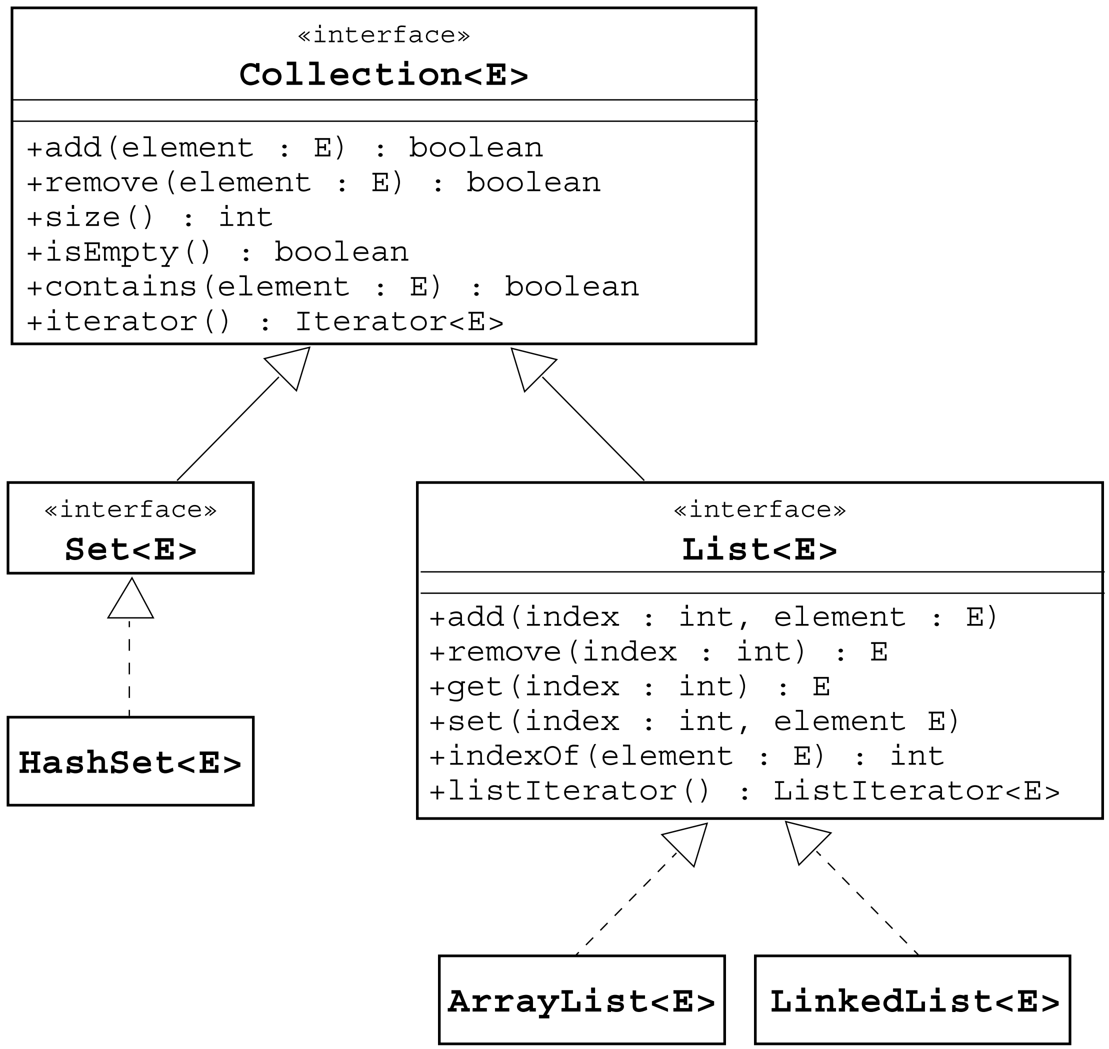
```
ArrayList list = new ArrayList();
list.add(0, new Integer(42));
int total = ((Integer)list.get(0)).intValue();
```

After Generics

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(0, new Integer(42));
int total = list.get(0).intValue();
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Generic Collections API



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Compiler Warnings

```
import java.util.*;
public class GenericsWarning {
  public static void main(String[] args) {
    List list = new ArrayList();
    list.add(0, new Integer(42));
    int total = ((Integer)list.get(0)).intValue();
  }
}
```

```
javac GenericsWarning.java
Note: GenericsWarning.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

javac -Xlint:unchecked GenericsWarning.java
GenericsWarning.java:7: warning: [unchecked] unchecked call to add(int,E)
as a member of the raw type java.util.ArrayList
    list.add(0, new Integer(42));
            ^
1 warning
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

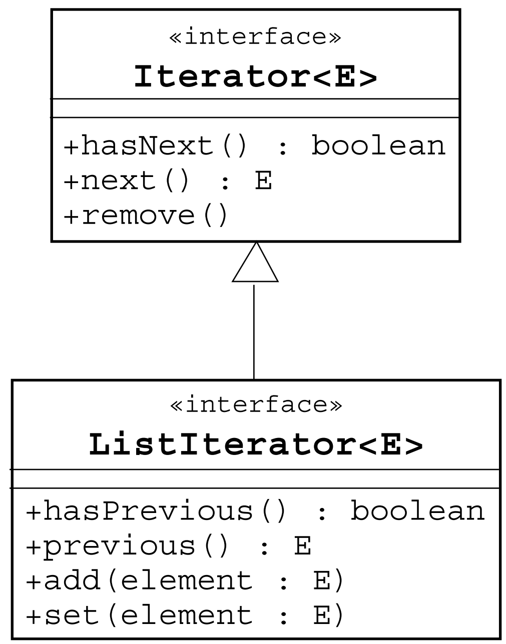
Iterators

- Iteration is the process of retrieving every element in a collection.
- An `Iterator` of a `Set` is unordered.
- A `ListIterator` of a `List` can be scanned forwards (using the `next` method) or backwards (using the `previous` method).

```
List list = new ArrayList();
// add some elements
Iterator elements = list.iterator();
while ( elements.hasNext() ) {
  System.out.println(elements.next());
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Iterator Interface Hierarchy



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Enhanced `for` Loop

The enhanced `for` loop has the following characteristics:

- Simplified iteration over collections
- Much shorter, clearer, and safer
- Effective for arrays
- Simpler when using nested loops
- Iterator disadvantages removed

Iterators are error prone:

- Iterator variables occur three times per loop.
- This provides the opportunity for code to go wrong.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Enhanced `for` Loop

An enhanced `for` loop can look like this:

- Using iterators:

```
public void deleteAll(Collection<NameList> c){
  for ( Iterator<NameList> i = c.iterator() ; i.hasNext() ; ){
    NameList nl = i.next();
    nl.deleteItem();
  }
}
```

- Using enhanced `for` loop in collections:

```
public void deleteAll(Collection<NameList> c){
  for ( NameList nl : c ){
    nl.deleteItem();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Enhanced `for` Loop

- Using enhanced `for` loop in arrays:

```
public int sum(int[] array){
  int result = 0;
  for ( int element : array ) {
    result += element;
  }
  return result;
}
```

- Using enhanced `for` loop in nested loops:

```
List<Subject> subjects=...;
List<Teacher> teachers=...;
List<Course> courseList = new ArrayList<Course>();
for ( Subject subj : subjects ) {
  for ( Teacher tchr : teachers ) {
    courseList.add(new Course(subj, tchr));
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 10

Building Java GUIs

Objectives

- Describe the Abstract Window Toolkit (AWT) package and its components
- Define the terms *containers*, *components*, and *layout managers*, and describe how they work together to build a GUI
- Use layout managers
- Use the `FlowLayout`, `BorderLayout`, and `GridLayout` managers to achieve a desired dynamic layout
- Add components to a container
- Use the `Frame` and `Panel` containers appropriately
- Describe how complex layouts with nested containers work

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Relevance

As a platform-independent programming language, how is Java technology used to make the graphical user interface (GUI) platform-independent?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

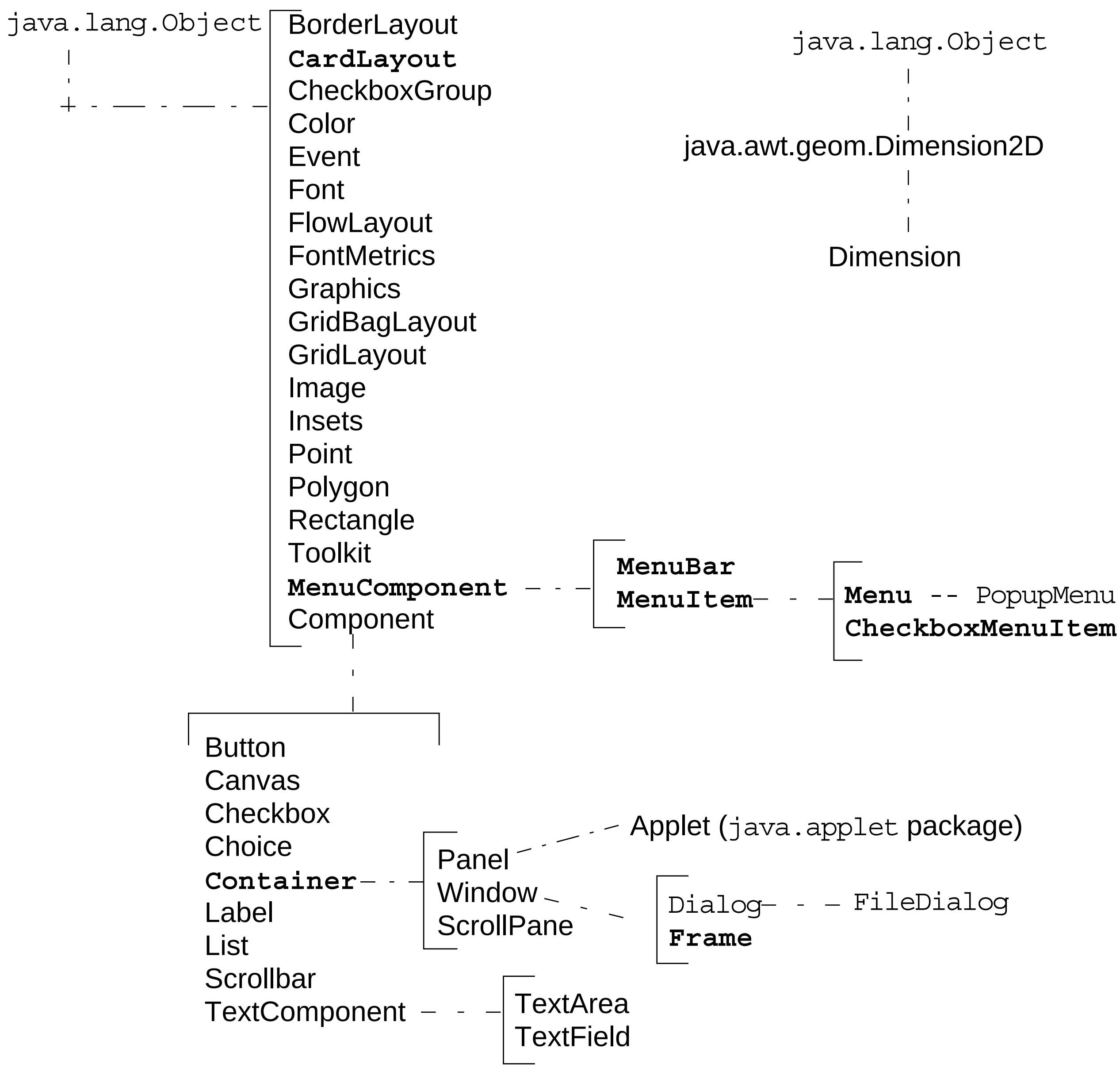
Abstract Window Toolkit

The AWT performs the following:

- Provides GUI components that are used in all Java applets and applications
- Contains classes that can be composed or extended; classes can also be abstract
- Ensures that every GUI component that is displayed on the screen is a subclass of the abstract class `Component` or `MenuComponent`
- Has `Container`, which is an abstract subclass of `Component` and includes two subclasses:
  - `Panel`
  - `Window`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `java.awt` Package



Exceptions – `AWTException` Errors – `AWTError`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Containers

- Add components with the `add()` method.
- The two main types of containers are `Window` and `Panel`.
- A `Window` is a free floating window on the display.
- A `Panel` is a container of GUI components that must exist in the context of some other container, such as a window or applet.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Positioning Components

- 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.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Frames

Frames have the following characteristics:

- Are a subclass of `Window`
- Have title and resizing corners
- Are invisible initially; use `setVisible(true)` to expose the frame
- Have `BorderLayout` as the default layout manager
- Use the `setLayout` method to change the default layout manager

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The FrameExample Class

```
import java.awt.*;

public class FrameExample {
  private Frame f;
  public FrameExample() {
    f = new Frame("Hello Out There!");
  }

  public void launchFrame() {
    f.setSize(170,170);
    f.setBackground(Color.blue);
    f.setVisible(true);
  }

  public static void main(String args[]) {
    FrameExample guiWindow = new FrameExample();
    guiWindow.launchFrame();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Example `Frame`

Solaris OS

Microsoft Windows

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Panels

- Panels provide a space for components.
- This enables subpanels to have their own layout manager.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `FrameWithPanel` Class

```
import java.awt.*;

public class FrameWithPanel {
  private Frame f;
  private Panel pan;

  public FrameWithPanel(String title) {
    f = new Frame(title);
    pan = new Panel();
  }
```

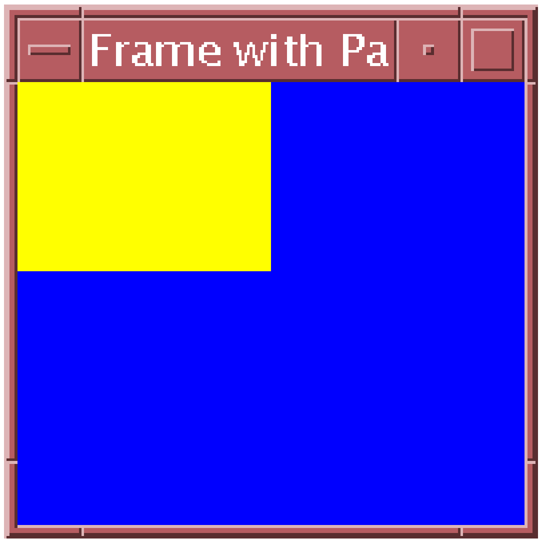

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `FrameWithPanel` Class

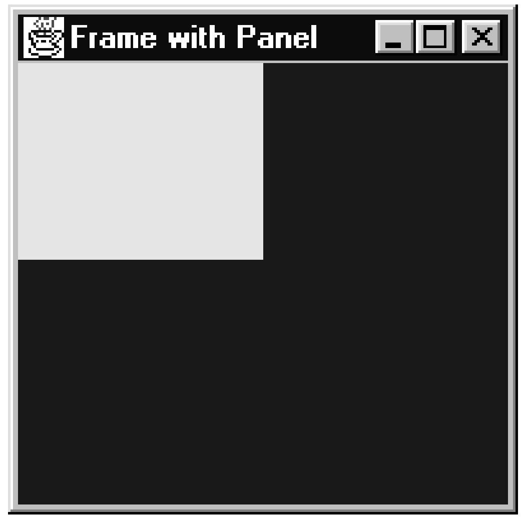
```
  public void launchFrame() {
    f.setSize(200,200);
    f.setBackground(Color.blue);
    f.setLayout(null);  // Use default layout

    pan.setSize(100,100);
    pan.setBackground(Color.yellow);
    f.add(pan);
    f.setVisible(true);
  }

  public static void main(String args[]) {
    FrameWithPanel guiWindow =
        new FrameWithPanel("Frame with Panel");
    guiWindow.launchFrame();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Example `Panel`



Solaris OS



Microsoft Windows

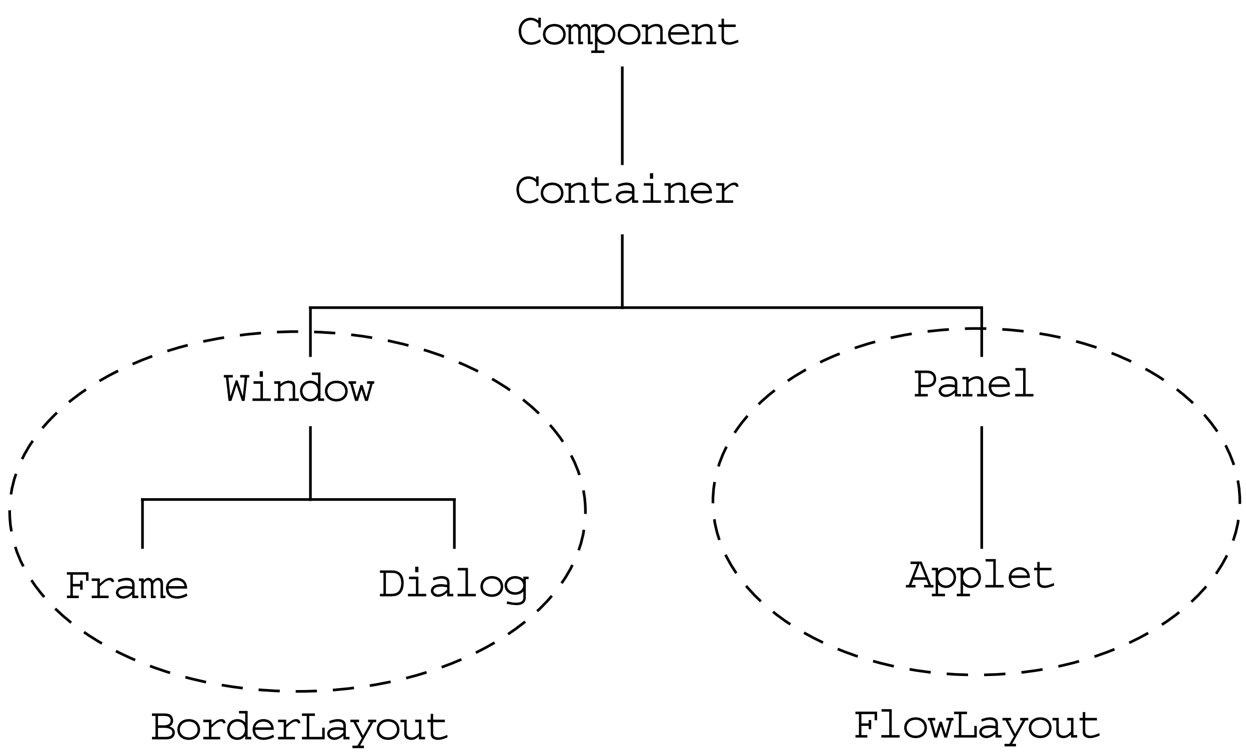
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Layout Managers

- `FlowLayout`
- `BorderLayout`
- `GridLayout`
- `CardLayout`
- `GridBagLayout`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Default Layout Managers



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

A Simple `FlowLayout` Example

```
import java.awt.*;

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

  public LayoutExample() {
    f = new Frame("GUI example");
    b1 = new Button("Press Me");
    b2 = new Button("Don't press Me");
  }
```

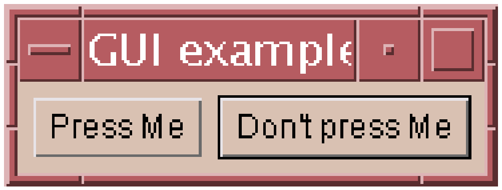

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

A Simple FlowLayout Example

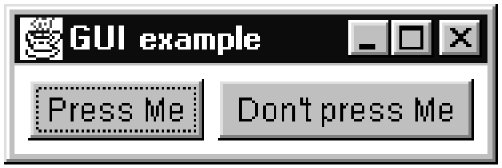
```

  public void launchFrame() {
    f.setLayout(new FlowLayout());
    f.add(b1);
    f.add(b2);
    f.pack();
    f.setVisible(true);
  }

  public static void main(String args[]) {
    LayoutExample guiWindow = new LayoutExample();
    guiWindow.launchFrame();
  }

} // end of LayoutExample class
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Example of `FlowLayout`



Solaris OS



Microsoft Windows

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

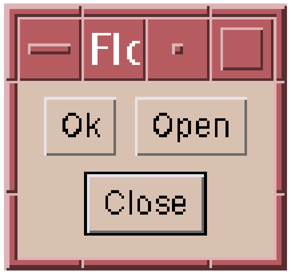
The `FlowLayout` Manager

The `FlowLayout` manager has the following characteristics:

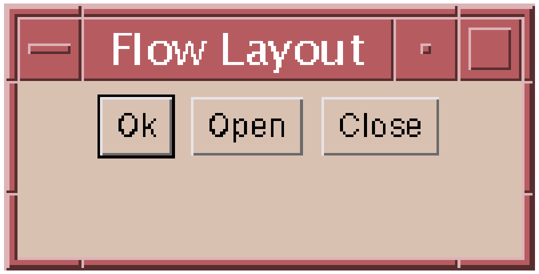
- Forms the default layout for the `Panel` class
- Adds components from left to right
- Alignment default is centered
- Uses components' preferred sizes
- Uses the constructor to tune behavior

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `FlowLayout` Resizing



After user or
program resizes



Solaris OS

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The FlowExample Class

```
import java.awt.*;

public class FlowExample {
  private Frame f;
  private Button button1;
  private Button button2;
  private Button button3;

  public FlowExample() {
    f = new Frame("Flow Layout");
    button1 = new Button("Ok");
    button2 = new Button("Open");
    button3 = new Button("Close");
  }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `FlowExample` Class

```java

    public void launchFrame() {
      f.setLayout(new FlowLayout());
      f.add(button1);
      f.add(button2);
      f.add(button3);
      f.setSize(100,100);
      f.setVisible(true);
    }

    public static void main(String args[]) {
      FlowExample guiWindow = new FlowExample();
      guiWindow.launchFrame();
    }
}
```

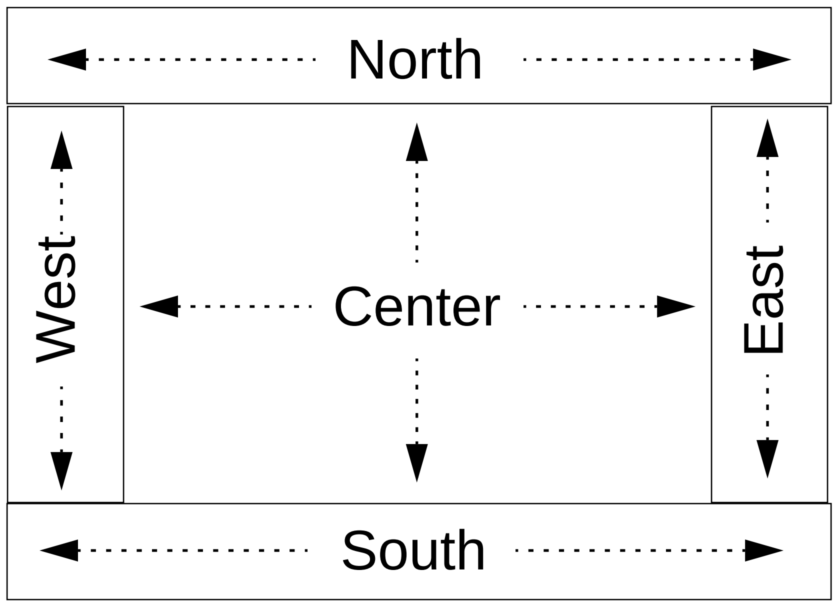

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `BorderLayout` Manager

- The `BorderLayout` manager is the default layout for the `Frame` class.
- Components are added to specific regions.
- The resizing behavior is as follows:
  - North, South, and Center regions adjust horizontally
  - East, West, and Center regions adjust vertically

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Organization of the Border Layout Components



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `BorderExample` Class

```
import java.awt.*;

public class BorderExample {
  private Frame f;
  private Button bn, bs, bw, be, bc;

  public BorderExample() {
    f = new Frame("Border Layout");
    bn = new Button("B1");
    bs = new Button("B2");
    bw = new Button("B3");
    be = new Button("B4");
    bc = new Button("B5");
  }
```

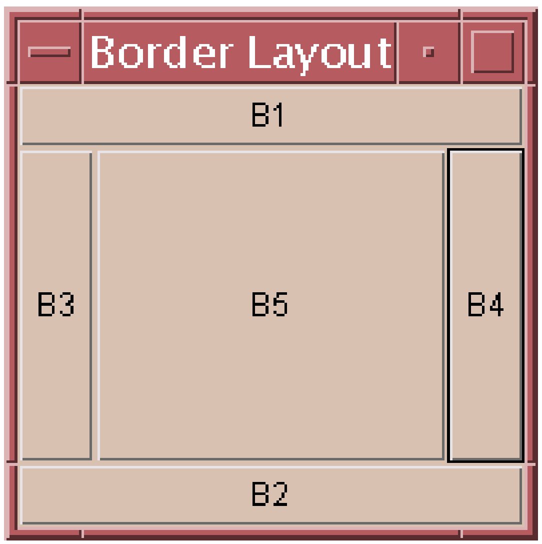

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `BorderExample` Class

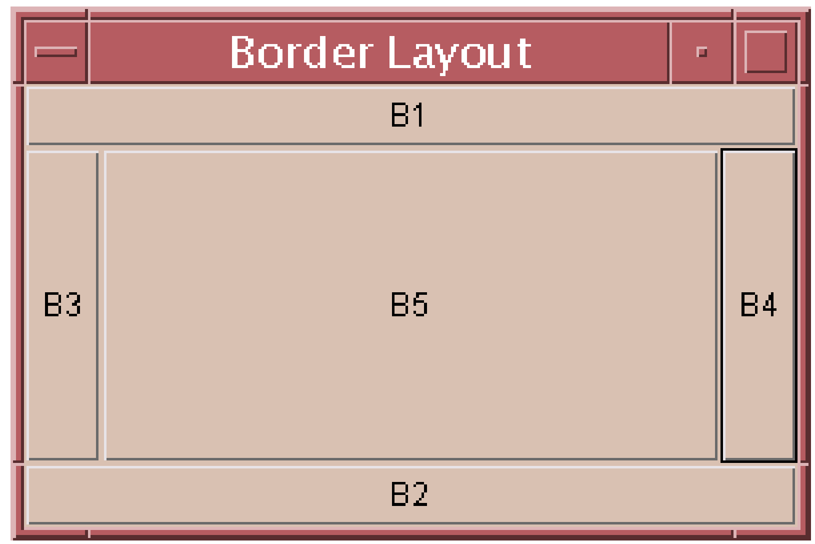
```

  public void launchFrame() {
    f.add(bn, BorderLayout.NORTH);
    f.add(bs, BorderLayout.SOUTH);
    f.add(bw, BorderLayout.WEST);
    f.add(be, BorderLayout.EAST);
    f.add(bc, BorderLayout.CENTER);
    f.setSize(200,200);
    f.setVisible(true);
  }

  public static void main(String args[]) {
    BorderExample guiWindow2 = new BorderExample();
    guiWindow2.launchFrame();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Example of `BorderLayout`



After user or program resizes



Solaris OS

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `GridLayout` Manager

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

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `GridExample` Class

```
import java.awt.*;

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

  public GridExample() {
    f = new Frame("Grid Example");
    b1 = new Button("1");
    b2 = new Button("2");
    b3 = new Button("3");
    b4 = new Button("4");
    b5 = new Button("5");
    b6 = new Button("6");
  }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The GridExample Class

```
    public void launchFrame() {
      f.setLayout (new GridLayout(3,2));
      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[]) {
      GridExample grid = new GridExample();
      grid.launchFrame();
    }
}
```

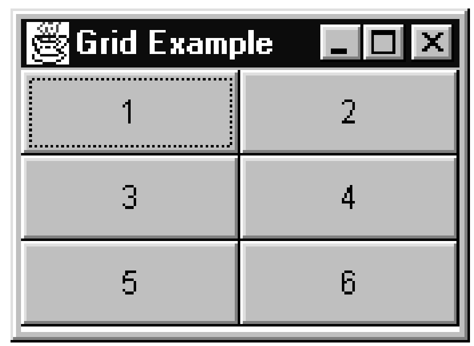

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Example of `GridLayout`

After user or program resizes

Solaris OS

After user or program resizes



Microsoft Windows

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `ComplexLayoutExample` Class

```
import java.awt.*;

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

  public ComplexLayoutExample() {
    f = new Frame("GUI example 3");
    bw = new Button("West");
    bc = new Button("Work space region");
    bfile = new Button("File");
    bhelp = new Button("Help");
  }
```

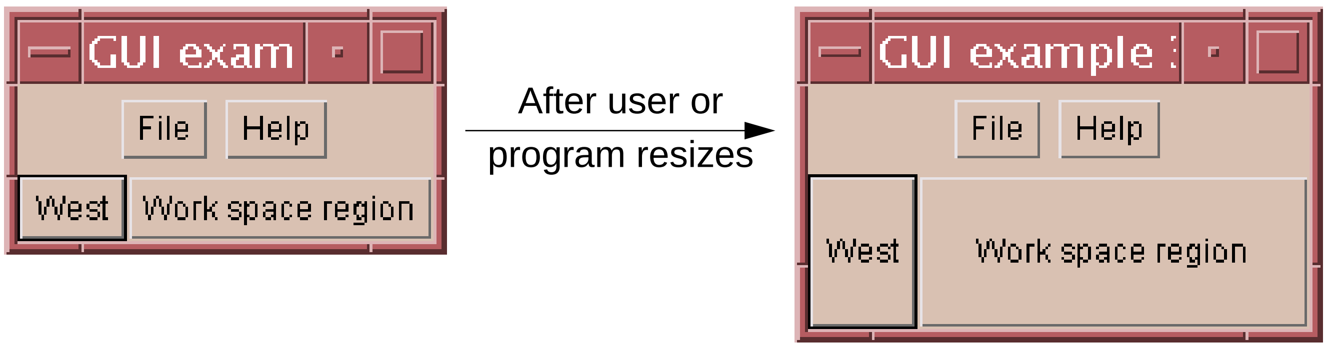

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The ComplexLayoutExample Class

```
public void launchFrame() {
    // Add bw and bc buttons in the frame border
    f.add(bw, BorderLayout.WEST);
    f.add(bc, BorderLayout.CENTER);
    // Create panel for the buttons in the north border
    p = new Panel();
    p.add(bfile);
    p.add(bhelp);
    f.add(p, BorderLayout.NORTH);
    // Pack the frame and make it visible
    f.pack();
    f.setVisible(true);
  }

  public static void main(String args[]) {
    ComplexLayoutExample gui = new ComplexLayoutExample();
    gui.launchFrame();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Combining Layout Managers



Solaris OS

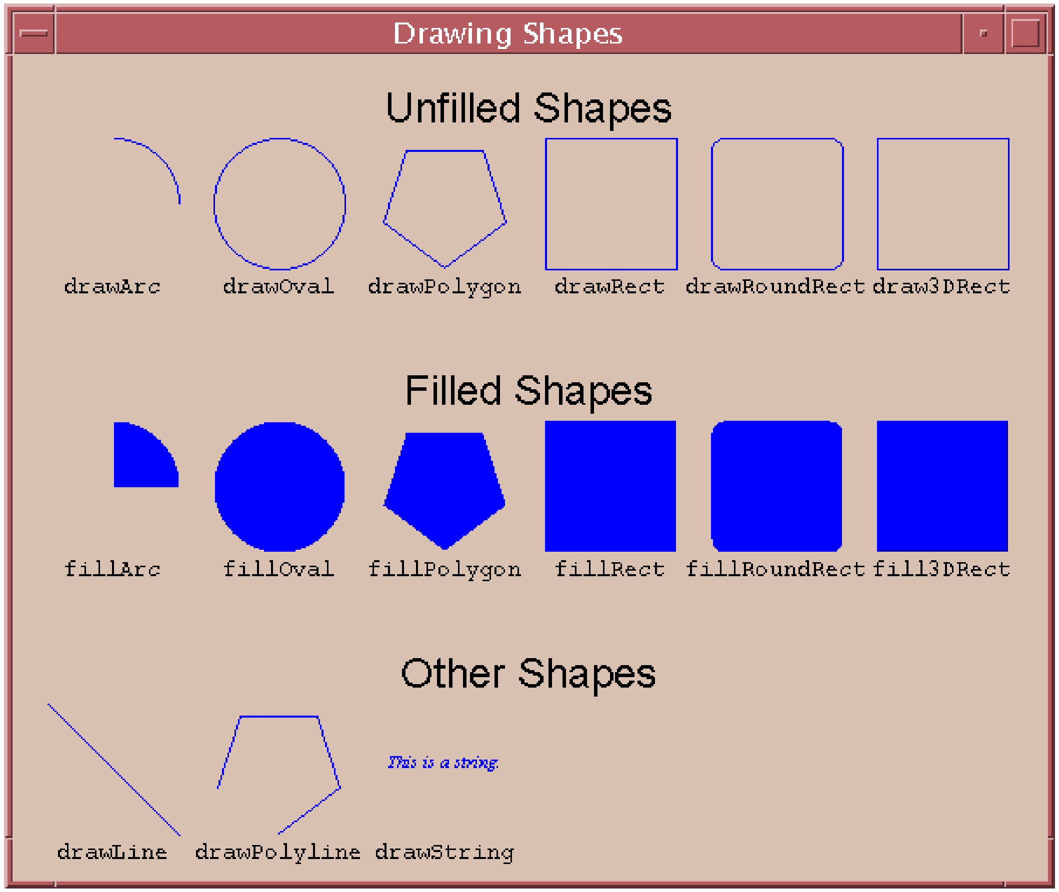
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Drawing in AWT

- You can draw in any `Component` (although AWT provides the `Canvas` and `Panel` classes just for this purpose).
- Typically, you create a subclass of `Canvas` or `Panel` and override the `paint` method.
- The `paint` method is called every time the component is shown (for example, if another window overlapped the component and was then removed).
- Every component has a `Graphics` object.
- The `Graphics` class implements many drawing methods.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Various Shapes Drawn by the `Graphics` Object



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 11

GUI Event Handling

Objectives

- Define events and event handling
- 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
- Identify the appropriate listener interface for a variety of event types
- Create the appropriate event handler methods for a variety of event types
- Understand the use of inner classes and anonymous classes in event handling

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

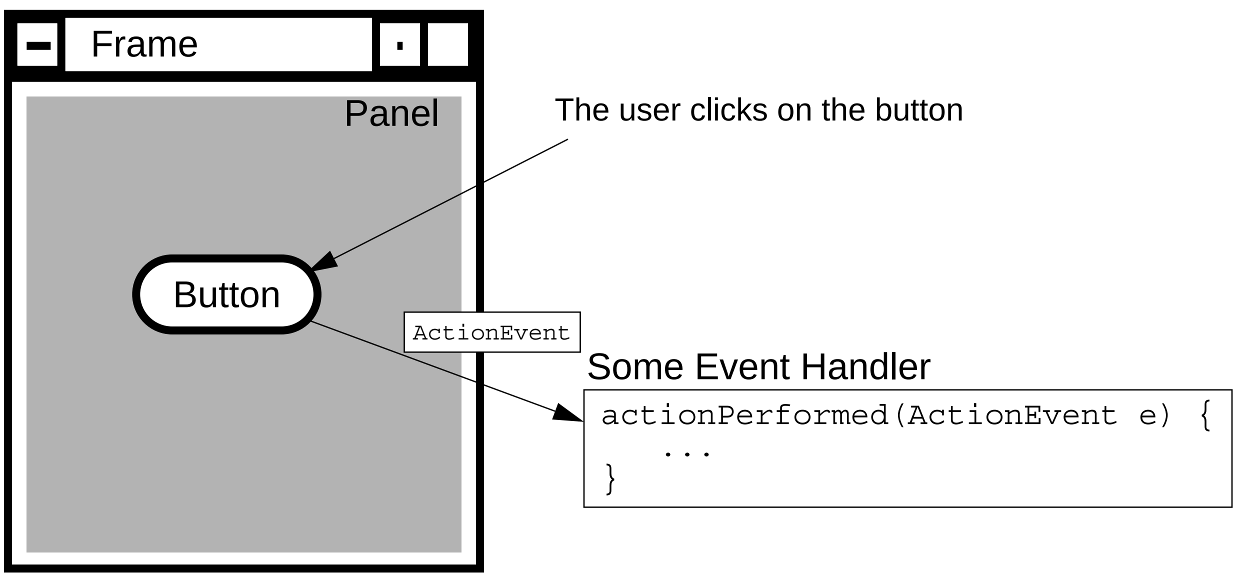
Relevance

- What parts of a GUI are required to make it useful?
- How does a graphical program handle a mouse click or any other type of user interaction?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

What Is an Event?

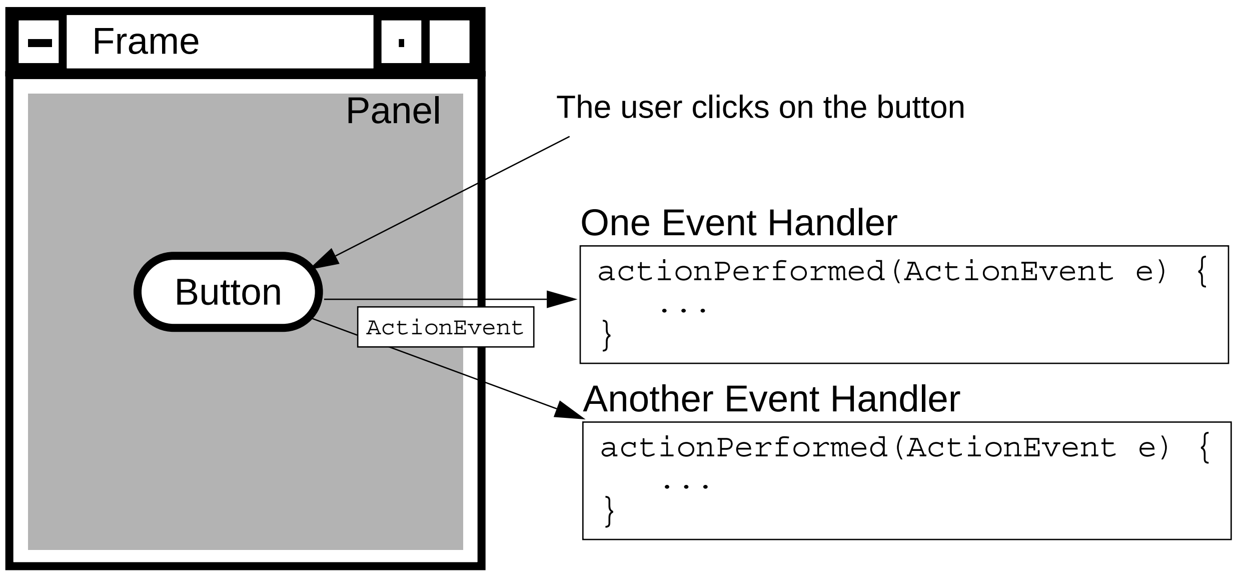
- Events – Objects that describe what happened
- Event sources – The generator of an event
- Event handlers – A method that receives an event object, deciphers it, and processes the user's interaction



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Delegation Model

- An event can be sent to many event handlers.



- Event handlers register with components when they are interested in events generated by that component.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Delegation Model

- Client objects (handlers) register with a GUI component that they want to observe.
- GUI components only trigger the handlers for the type of event that has occurred.
- Most components can trigger more than one type of event.
- The delegation model distributes the work among multiple classes.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

A Listener Example

```
import java.awt.*;

public class TestButton {
  private Frame f;
  private Button b;

  public TestButton() {
    f = new Frame("Test");
    b = new Button("Press Me!");
    b.setActionCommand("ButtonPressed");
  }

  public void launchFrame() {
    b.addActionListener(new ButtonHandler());
    f.add(b,BorderLayout.CENTER);
    f.pack();
    f.setVisible(true);
  }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

A Listener Example

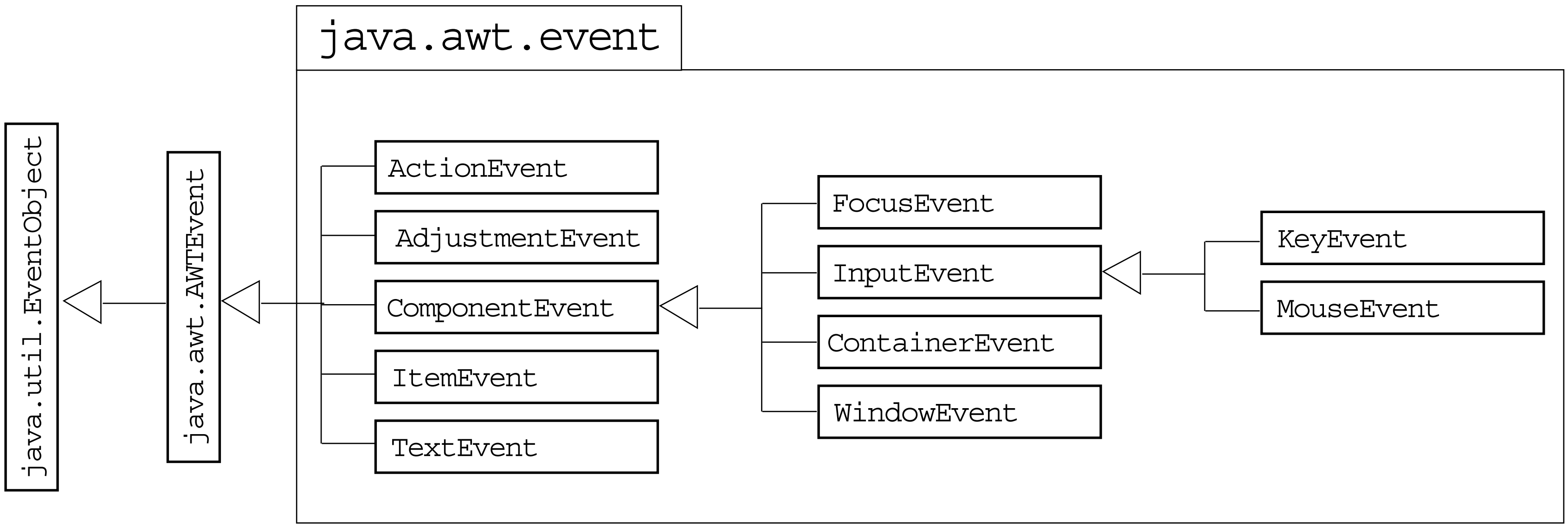
```
  public static void main(String args[]) {
    TestButton guiApp = new TestButton();
    guiApp.launchFrame();
  }
}
```

Code for the event listener looks like this:

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

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


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Event Categories



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Method Categories and Interfaces

Category	Interface Name	Methods
Action	`ActionListener`	`actionPerformed(ActionEvent)`
Item	`ItemListener`	`itemStateChanged(ItemEvent)`

Mouse	`MouseListener`	`mousePressed(MouseEvent)`
		`mouseReleased(MouseEvent)`
		`mouseEntered(MouseEvent)`
		`mouseExited(MouseEvent)`
		`mouseClicked(MouseEvent)`

Mouse motion	`MouseMotionListener`	`mouseDragged(MouseEvent)` `mouseMoved(MouseEvent)`

Key	`KeyListener`	`keyPressed(KeyEvent)`
		`keyReleased(KeyEvent)`
		`keyTyped(KeyEvent)`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Method Categories and Interfaces

Category	Interface Name	Methods
Focus	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
Adjustment	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent)
Component	ComponentListener	componentMoved(ComponentEvent) componentHidden(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Method Categories and Interfaces

Category	Interface Name	Methods

Window	`WindowListener`	`windowClosing(WindowEvent)`
		`windowOpened(WindowEvent)`
		`windowIconified(WindowEvent)`
		`windowDeiconified(WindowEvent)`
		`windowClosed(WindowEvent)`
		`windowActivated(WindowEvent)`
		`windowDeactivated(WindowEvent)`

Container	`ContainerListener`	`componentAdded(ContainerEvent)` `componentRemoved(ContainerEvent)`
Text	`TextListener`	`textValueChanged(TextEvent)`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Complex Example

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

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

  public TwoListener() {
    f = new Frame("Two listeners example");
    tf = new TextField(30);
  }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Complex Example

```

    public void launchFrame() {
      Label label = new Label("Click and drag the mouse");
      // Add components to the frame
      f.add(label, BorderLayout.NORTH);
      f.add(tf, BorderLayout.SOUTH);
      // Add this object as a listener
      f.addMouseMotionListener(this);
      f.addMouseListener(this);
      // Size the frame and make it visible
      f.setSize(300, 200);
      f.setVisible(true);
    }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Complex Example

```java

    // 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);
    }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Complex Example

```java

    // Unused MouseMotionListener method.
    // All methods of a listener must be present in the
    // class even if they are not used.
    public void mouseMoved(MouseEvent e) { }

    // Unused MouseListener methods.
    public void mousePressed(MouseEvent e) { }
    public void mouseClicked(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }

    public static void main(String args[]) {
      TwoListener two = new TwoListener();
      two.launchFrame();
    }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Multiple Listeners

- Multiple listeners cause unrelated parts of a program to react to the same event.
- The handlers of all registered listeners are called when the event occurs.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Event Adapters

- The listener classes that you define can extend adapter classes and override only the methods that you need.
- An example is:

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

public class MouseClickHandler extends MouseAdapter {

  // We just need the mouseClick 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...
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Event Handling Using Inner Classes

```
import java.awt.*;
import java.awt.event.*;
public class TestInner {
  private Frame f;
  private TextField tf; // used by inner class

  public TestInner() {
    f = new Frame("Inner classes example");
    tf = new TextField(30);
  }

  class MyMouseMotionListener extends MouseMotionAdapter {
      public void mouseDragged(MouseEvent e) {
        String s = "Mouse dragging:  X = "+ e.getX()
                    + " Y = " + e.getY();
        tf.setText(s);
      }
    }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Event Handling Using Inner Classes

```java

  public void launchFrame() {
    Label label = new Label("Click and drag the mouse");
    // Add components to the frame
    f.add(label, BorderLayout.NORTH);
    f.add(tf, BorderLayout.SOUTH);
    // Add a listener that uses an Inner class
    f.addMouseMotionListener(new MyMouseMotionListener());
    f.addMouseListener(new MouseClickHandler());
    // Size the frame and make it visible
    f.setSize(300, 200);
    f.setVisible(true);
  }

  public static void main(String args[]) {
    TestInner obj = new TestInner();
    obj.launchFrame();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Event Handling Using Anonymous Classes

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

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

  public TestAnonymous() {
    f = new Frame("Anonymous classes example");
    tf = new TextField(30);
  }

  public static void main(String args[]) {
    TestAnonymous obj = new TestAnonymous();
    obj.launchFrame();
  }

```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Event Handling Using Anonymous Classes

```
  public void launchFrame() {
    Label label = new Label("Click and drag the mouse");
    // Add components to the frame
    f.add(label, BorderLayout.NORTH);
    f.add(tf, BorderLayout.SOUTH);
    // Add a listener that uses an anonymous class
    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()); // Not shown
    // Size the frame and make it visible
    f.setSize(300, 200);
    f.setVisible(true);
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 12

GUI-Based Applications

Objectives

- Identify the key AWT components and the events that they trigger
- Describe how to construct a menu bar, menu, and menu items in a Java GUI
- Understand how to change the color and font of a component

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

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?
- How can you create a menu for your GUI frame?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

AWT Components

Component Type	Description
`Button`	A named rectangular box used for receiving mouse clicks
`Canvas`	A panel used for drawing
`Checkbox`	A component enabling the user to select an item
`CheckboxMenuItem`	A checkbox within a menu
`Choice`	A pull-down static list of items
`Component`	The parent of all AWT components, except menu components
`Container`	The parent of all AWT containers
`Dialog`	A top-level window with a title and a border; dialogs can be modeless or modal
`Frame`	The base class of all GUI windows with window manager controls

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

AWT Components

Component Type	**Description**
`Label`	A text string component
`List`	A component that contains a dynamic set of items
`Menu`	An element under the menu bar, which contains a set of menu items
`MenuItem`	An item within a menu
`Panel`	A basic container class used most often to create complex layouts
`Scrollbar`	A component that enables a user to *select from a range of values*
`ScrollPane`	A container class that implements automatic horizontal and vertical scrolling for a single child component

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

AWT Components

Component Type	Description
`TextArea`	A component that enables the user to enter a block of text
`TextField`	A component that enables the user to enter a single line of text
`Window`	The base class of all GUI windows, without window manager controls

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

AWT Listeners

Component Type	Act	Adj	Cmp	Cnt	Foc	Itm	Key	Mou	MM	Text	Win
Button	✓		✓		✓		✓	✓	✓		
Canvas			✓		✓		✓	✓	✓		
Checkbox			✓		✓	✓	✓	✓	✓		
Checkbox-MenuItem						✓					
Choice			✓		✓	✓	✓	✓	✓		
Component			✓		✓		✓	✓	✓		
Container			✓	✓	✓		✓	✓	✓		
Dialog			✓	✓	✓		✓	✓	✓		
Frame			✓	✓	✓		✓	✓	✓		
Label			✓		✓		✓	✓	✓		

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

AWT Listeners

Component Type	Act	Adj	Cmp	Cnt	Foc	Itm	Key	Mou	MM	Text	Win
List	✓		✓		✓	✓	✓	✓	✓		
MenuItem	✓										
Panel			✓	✓	✓		✓	✓	✓		
Scrollbar		✓	✓		✓		✓	✓	✓		
ScrollPane			✓	✓	✓		✓	✓	✓		
TextArea			✓		✓		✓	✓	✓	✓	
TextField	✓		✓		✓		✓	✓	✓	✓	
Window			✓	✓	✓		✓	✓	✓	✓	✓

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

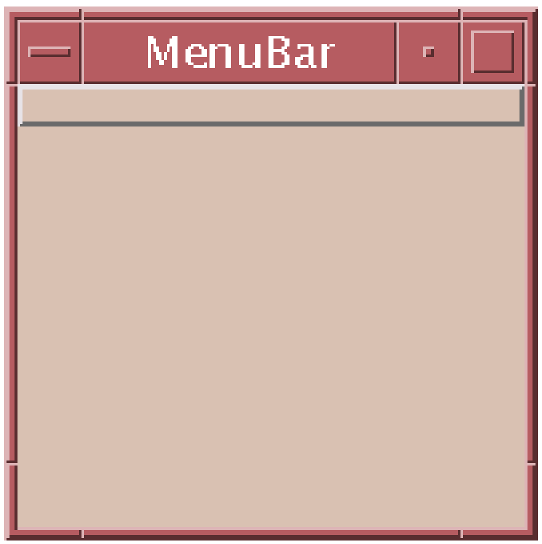
How to Create a Menu

1. Create a `MenuBar` object, and set it into a menu container, such as a `Frame`.
2. Create one or more `Menu` objects, and add them to the menu bar object.
3. Create one or more `MenuItem` objects, and add them to the menu object.

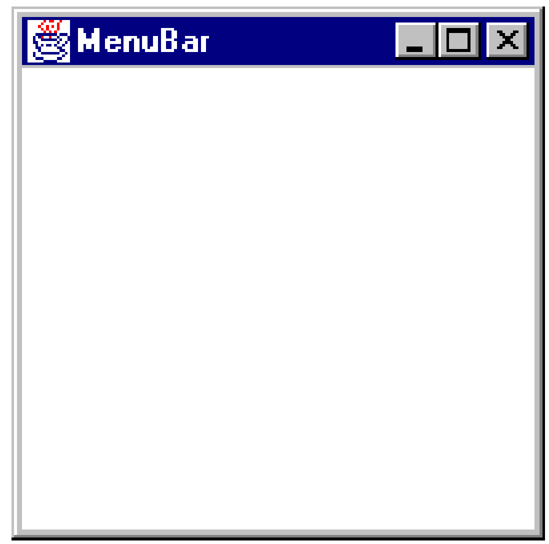
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating a `MenuBar`

```
Frame f = new Frame("MenuBar");
MenuBar mb = new MenuBar();
f.setMenuBar(mb);
```



Solaris OS



Microsoft Windows

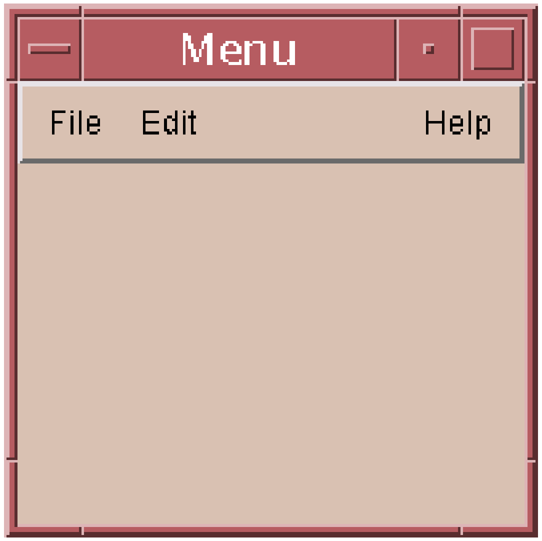
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating a `Menu`

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


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating a `Menu`



Solaris OS

Microsoft Windows

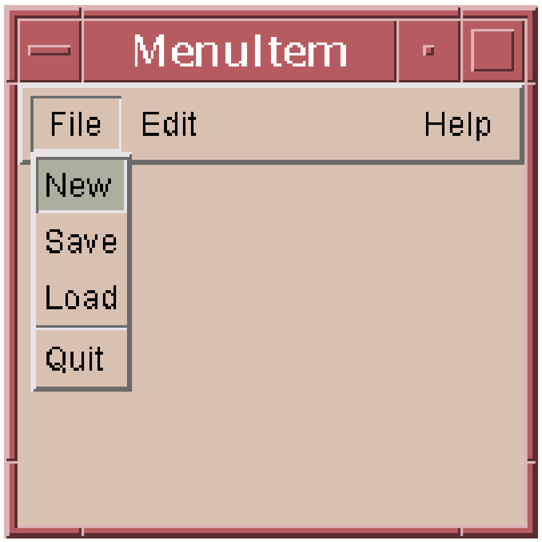
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating a `MenuItem`

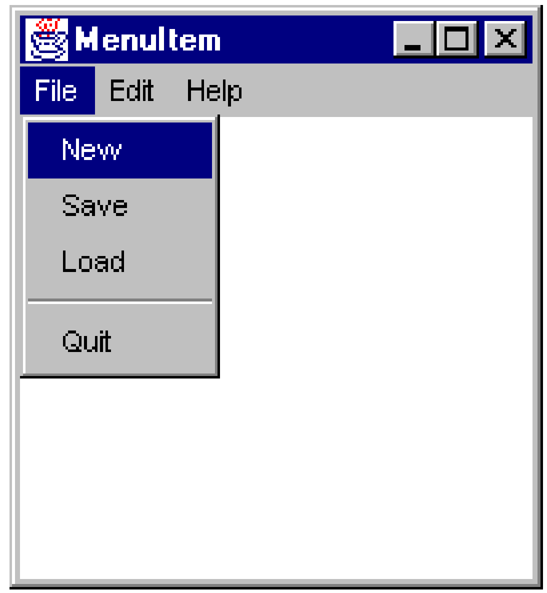
```
MenuItem mi1 = new MenuItem("New");
MenuItem mi2 = new MenuItem("Save");
MenuItem mi3 = new MenuItem("Load");
MenuItem mi4 = new MenuItem("Quit");
mi1.addActionListener(this);
mi2.addActionListener(this);
mi3.addActionListener(this);
mi4.addActionListener(this);
m1.add(mi1);
m1.add(mi2);
m1.add(mi3);
m1.addSeparator();
m1.add(mi4);
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating a `MenuItem`



Solaris OS



Microsoft Windows

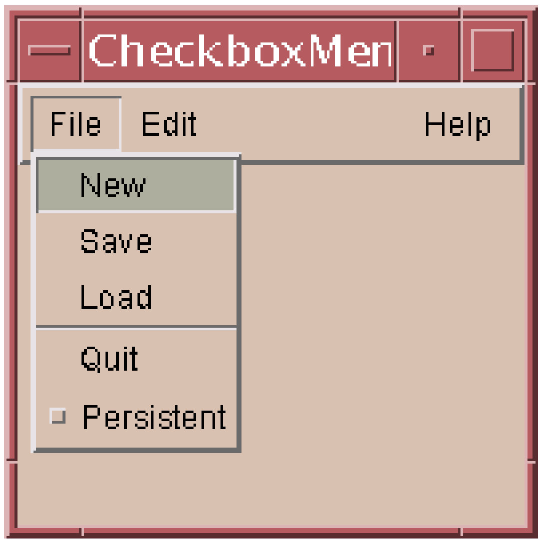
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating a `CheckBoxMenuItem`

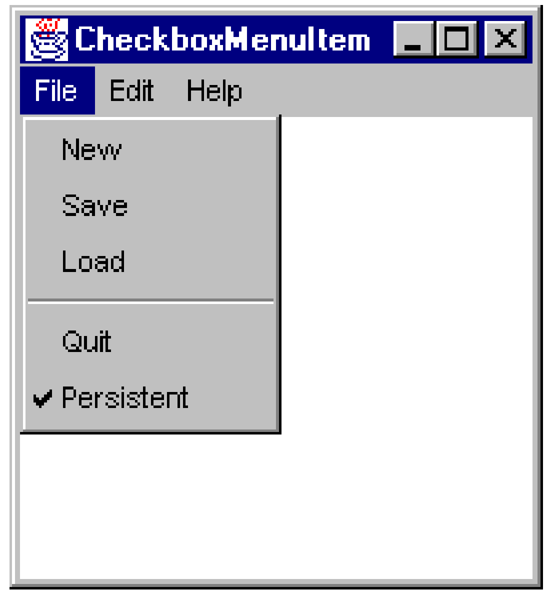
```
MenuBar mb = new MenuBar();
Menu m1 = new Menu("File");
Menu m2 = new Menu("Edit");
Menu m3 = new Menu("Help");
mb.add(m1);
mb.add(m2);
mb.setHelpMenu(m3);
f.setMenuBar(mb);
.....
MenuItem mi2 = new MenuItem("Save");
mi2.addActionListener(this);
m1.add(mi2);
......
CheckboxMenuItem mi5 = new CheckboxMenuItem("Persistent");
mi5.addItemListener(this);
m1.add(mi5);
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating a `CheckBoxMenuItem`



Solaris OS



Microsoft Windows

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Controlling Visual Aspects

Commands to control visual aspects of the GUI include:

- Colors:

```
setForeground()
setBackground()
```

- Example:

```
Color purple = new Color(255, 0, 255);
Button b = new Button("Purple");
b.setBackground(purple);
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

J.F.C./Swing Technology

- Java Foundation Class / Swing (J.F.C. / Swing) technology is a second-generation GUI toolkit.
- It builds on top of AWT, but supplants the components with *lightweight* versions.
- There are many more components, and much more complex components, including `JTable`, `JTree`, and `JComboBox`.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 13

Threads

Objectives

- Define a thread
- Create separate threads in a Java technology 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
- Use `synchronized` to protect data from corruption

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Relevance

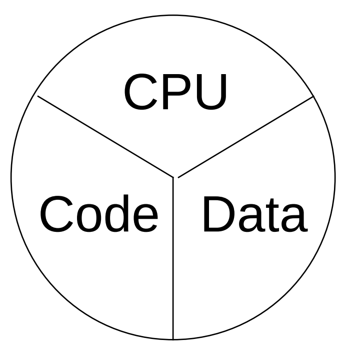
How do you get programs to perform multiple tasks concurrently?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Threads

- What are threads?

  Threads are a virtual CPU.
- The three parts of at thread are:
  - CPU
  - Code
  - Data



A thread or execution context

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating the Thread

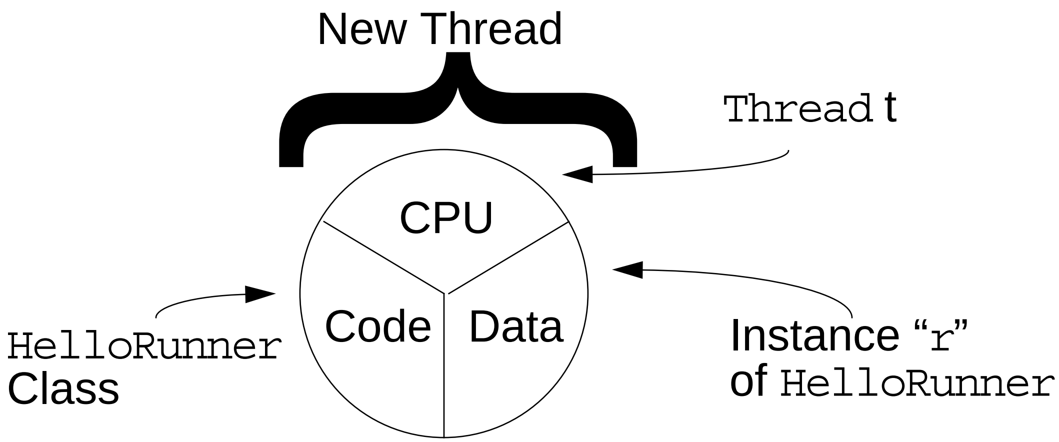
```
public class ThreadTester {
  public static void main(String args[]) {
    HelloRunner r = new HelloRunner();
    Thread t = new Thread(r);
    t.start();
  }
}
class HelloRunner implements Runnable {
  int i;
  public void run() {
    i = 0;
    while (true) {
      System.out.println("Hello " + i++);
      if ( i == 50 ) {
        break;
      }
    }
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Creating the Thread

- Multithreaded programming has these characteristics:
  - Multiple threads are from one `Runnable` instance.
  - Threads share the same data and code.
- For example:

```
Thread t1 = new Thread(r);
Thread t2 = new Thread(r);
```



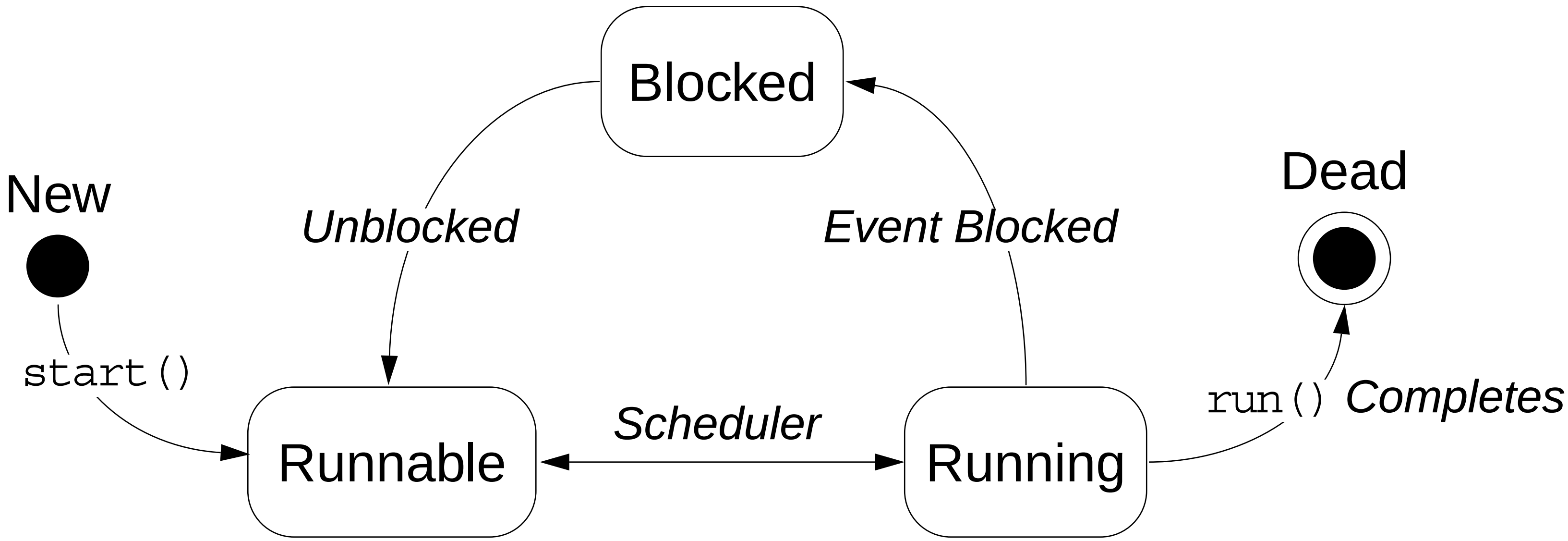
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Starting the Thread

- Use the `start` method.
- Place the thread in a runnable state.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Thread Scheduling



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Thread Scheduling Example

```
public class Runner 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
      }
    }
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Terminating a Thread

```
public class Runner implements Runnable {
  private boolean timeToQuit=false;

  public void run() {
    while ( ! timeToQuit ) {
      // continue doing work
    }
    // clean up before run() ends
  }

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


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Terminating a Thread

```
public class ThreadController {
  private Runner r = new Runner();
  private Thread t = new Thread(r);

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

  public void stopThread() {
    // use specific instance of Runner
    r.stopRunning();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Basic Control of Threads

- Test threads:

```
isAlive()
```

- Access thread priority:

```
getPriority()
setPriority()
```

- Put threads on hold:

```
Thread.sleep()  // static method
join()
Thread.yield()  // static method
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `join` Method

```
public static void main(String[] args) {
  Thread t = new Thread(new Runner());
  t.start();
  ...
  // Do stuff in parallel with the other thread for a while
  ...
  // Wait here for the other thread to finish
  try {
    t.join();
  } catch (InterruptedException e) {
    // the other thread came back early
  }
  ...
  // Now continue in this thread
  ...
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Other Ways to Create Threads

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

  public static void main(String args[]) {
    Thread t = new MyThread();
    t.start();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Selecting a Way to Create Threads

- Implement `Runnable`:
  - Better object-oriented design
  - Single inheritance
  - Consistency
- Extend `Thread`:

  Simpler code

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Using the `synchronized` Keyword

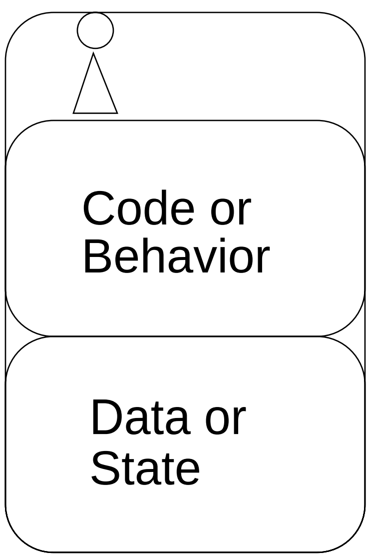
```
public class MyStack {

  int idx = 0;
  char [] data = new char[6];

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

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


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Object Lock Flag

- Every object has a flag that is a type of *lock flag*.
- The `synchronized` enables interaction with the lock flag.

Object `this`



Thread before `synchronized(this)`

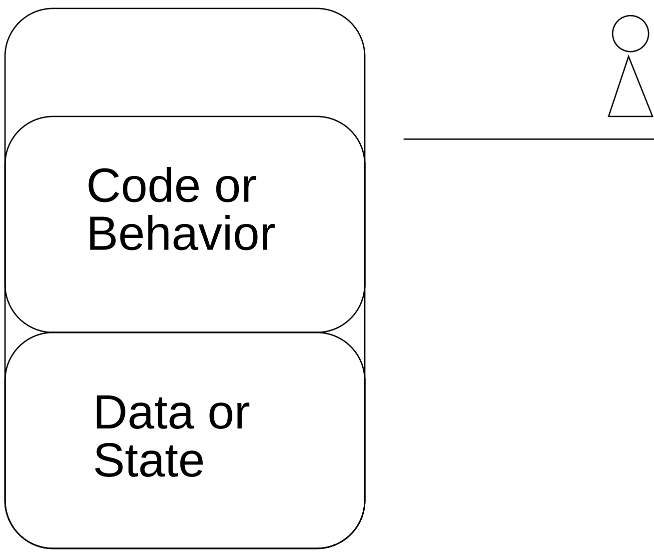
```
public void push(char c) {
  synchronized (this) {
    data[idx] = c;
    idx++;
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Object Lock Flag

Object `this`

Thread after `synchronized(this)`

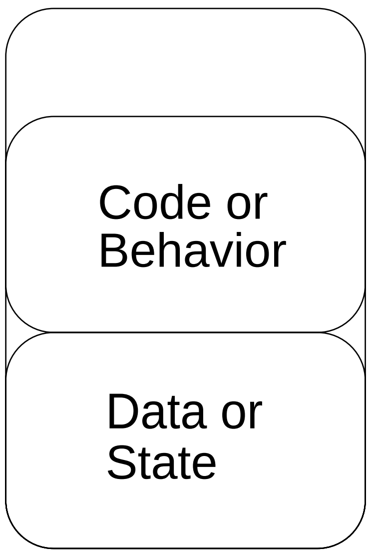



```
public void push(char c) {
  synchronized (this) {
    data[idx] = c;
    idx++;
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Object Lock Flag

Object `this`
lock flag missing



Another thread, trying to
execute `synchronized(this)`

Waiting for
object lock

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


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Releasing the Lock Flag

The lock flag is released in the following events:

- Released when the thread passes the end of the `synchronized` code block
- Released automatically when a break, return, or exception is thrown by the `synchronized` code block

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Using `synchronized` – Putting It Together

- *All* access to delicate data should be `synchronized`.
- Delicate data protected by `synchronized` should be `private`.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

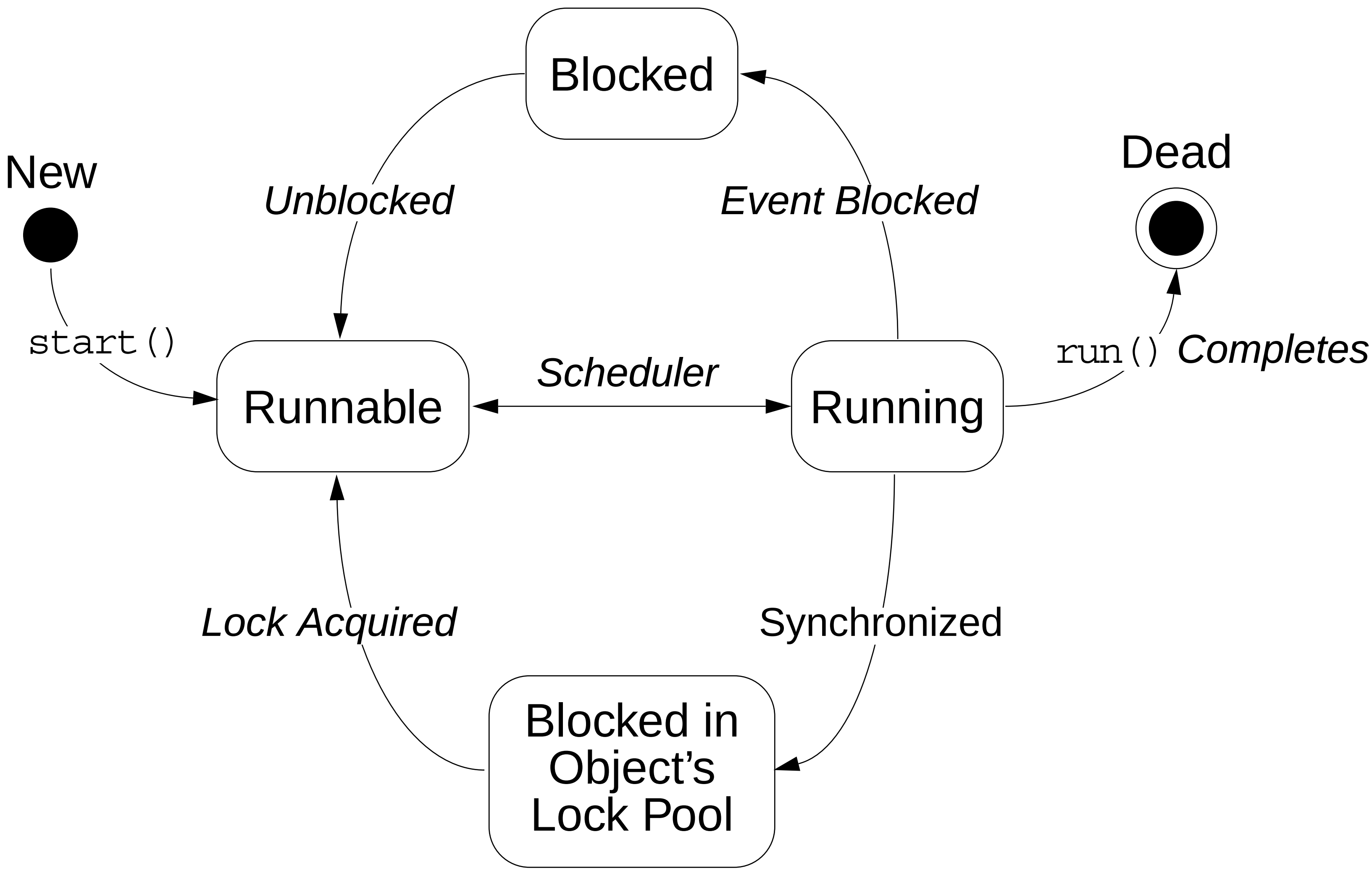
Using synchronized – Putting It Together

The following two code segments are equivalent:

```
public void push(char c) {
  synchronized(this) {
    // The push method code
  }
}

public synchronized void push(char c) {
  // The push method code
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Thread State Diagram With Synchronization



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Deadlock

A deadlock has the following characteristics:

- It is two threads, each waiting for a lock from the other.
- It is not detected or avoided.
- Deadlock can be avoided by:
  - Deciding on the order to obtain locks
  - Adhering to this order throughout
  - Releasing locks in reverse order

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Thread Interaction – `wait` and `notify`

- Scenario:

  Consider yourself and a cab driver as two threads.
- The problem:

  How do you determine when you are at your destination?
- The solution:
  - You notify the cab driver of your destination and relax.
  - The driver drives and notifies you upon arrival at your destination.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

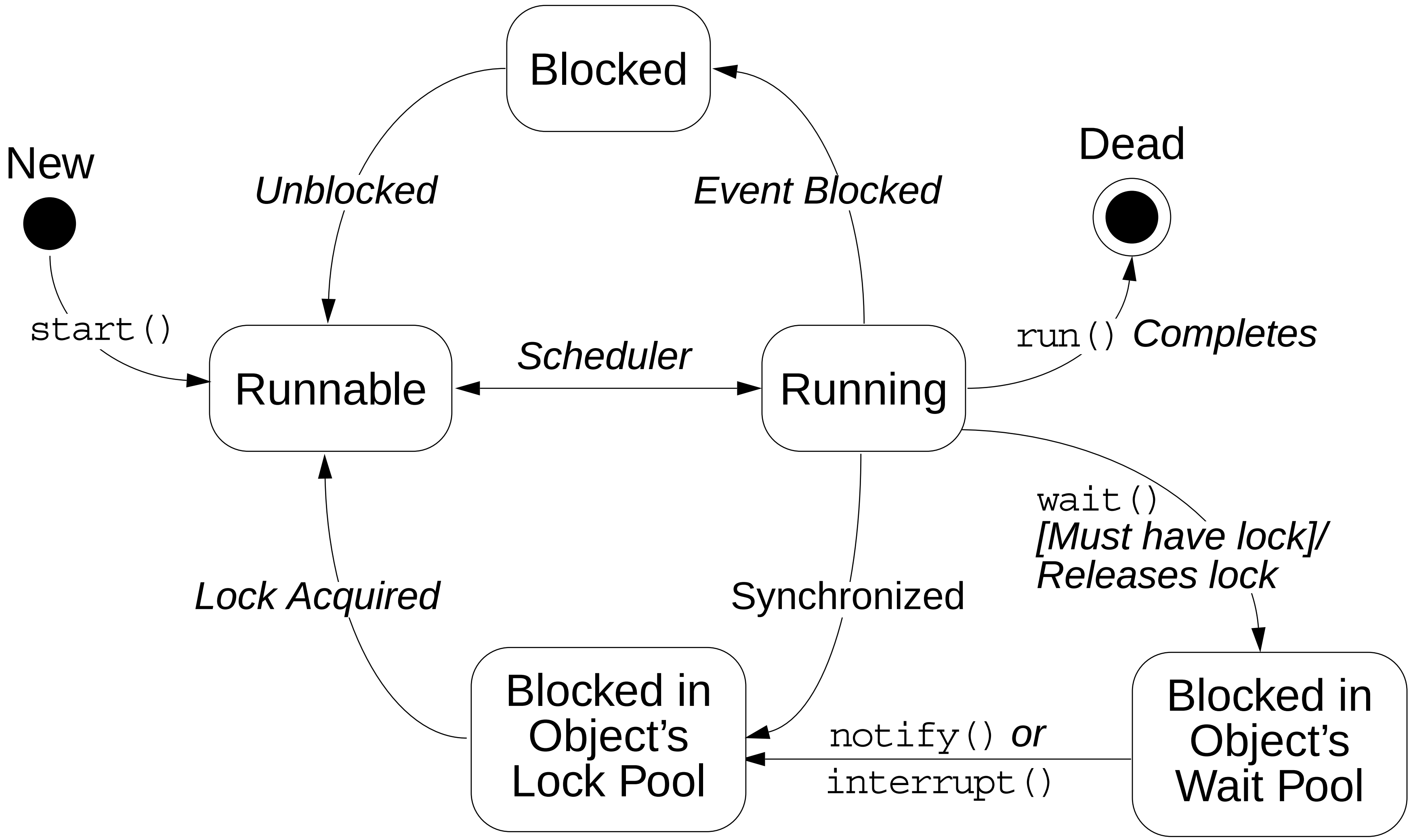
Thread Interaction

Thread interactions include:

- The `wait` and `notify` methods
- The pools:
  - Wait pool
  - Lock pool

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Thread State Diagram With `wait` and `notify`



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

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.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `Producer` Class

```
package mod13;

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

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

```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `Producer` Class

```
  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
      }
    }
  } // END run method

} // END Producer class
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Consumer Class

```
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++;
  }

```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Consumer Class

```
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) {
      // ignore it
    }
  }
} // END run method

```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The SyncStack Class

This is a sketch of the SyncStack class:

```
public class SyncStack {

  private List<Character> buffer = new ArrayList<Character>(400);

  public synchronized char pop() {
    // pop code here
  }

  public synchronized void push(char c) {
    // push code here
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The pop Method

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


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The push Method

```
public synchronized void push(char c) {
  this.notify();
  buffer.add(c);
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The SyncTest Class

```
package mod13;
public class SyncTest {
  public static void main(String[] args) {
    SyncStack stack = new SyncStack();
    Producer p1 = new Producer(stack);
    Thread prodT1 = new Thread (p1);
    prodT1.start();
    Producer p2 = new Producer(stack);
    Thread prodT2 = new Thread (p2);
    prodT2.start();

    Consumer c1 = new Consumer(stack);
    Thread consT1 = new Thread (c1);
    consT1.start();
    Consumer c2 = new Consumer(stack);
    Thread consT2 = new Thread (c2);
    consT2.start();
  }
}
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `SyncTest` Class

```
Producer2: F
Consumer1: F
Producer2: K
Consumer2: K
Producer2: T
Producer1: N
Producer1: V
Consumer2: V
Consumer1: N
Producer2: V
Producer2: U
Consumer2: U
Consumer2: V
Producer1: F
Consumer1: F
Producer2: M
Consumer2: M
Consumer2: T
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 14

Advanced I/O Streams

Objectives

- Describe the main features of the `java.io` package
- Construct node and processing streams, and use them appropriately
- Distinguish readers and writers from streams, and select appropriately between them

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Relevance

- What mechanisms are in place within the Java programming language to read and write from sources (or sinks) other than files?
- How are international character sets supported in I/O operations?
- What are the possible sources and sinks of character and byte streams?

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

I/O Fundamentals

- A *stream* can be thought of as a flow of data from a source or to a sink.
- A *source* stream initiates the flow of data, also called an input stream.
- A *sink* stream terminates the flow of data, also called an output stream.
- Sources and sinks are both *node streams*.
- Types of node streams are files, memory, and pipes between threads or processes.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Fundamental Stream Classes

Stream	**Byte Streams**	**Character Streams**
Source streams	`InputStream`	`Reader`
Sink streams	`OutputStream`	`Writer`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Data Within Streams

- Java technology supports two types of streams: character and byte.
- Input and output of character data is handled by readers and writers.
- Input and output of byte data is handled by input streams and output streams:
  - Normally, the term *stream* refers to a byte stream.
  - The terms *reader* and *writer* refer to character streams.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `InputStream` Methods

- The three basic `read` methods are:

```
int read()
int read(byte[] buffer)
int read(byte[] buffer, int offset, int length)
```

- Other methods include:

```
void close()
int available()
long skip(long n)
boolean markSupported()
void mark(int readlimit)
void reset()
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `OutputStream` Methods

- The three basic `write` methods are:

```
void write(int c)
void write(byte[] buffer)
void write(byte[] buffer, int offset, int length)
```

- Other methods include:

```
void close()
void flush()
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `Reader` Methods

- The three basic `read` methods are:

```
int read()
int read(char[] cbuf)
int read(char[] cbuf, int offset, int length)
```

- Other methods include:

```
void close()
boolean ready()
long skip(long n)
boolean markSupported()
void mark(int readAheadLimit)
void reset()
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `Writer` Methods

- The basic `write` methods are:

```
void write(int c)
void write(char[] cbuf)
void write(char[] cbuf, int offset, int length)
void write(String string)
void write(String string, int offset, int length)
```

- Other methods include:

```
void close()
void flush()
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Node Streams

Type	Character Streams	Byte Streams
File	`FileReader` `FileWriter`	`FileInputStream` `FileOutputStream`
Memory: array	`CharArrayReader` `CharArrayWriter`	`ByteArrayInputStream` `ByteArrayOutputStream`
Memory: string	`StringReader` `StringWriter`	N/A
Pipe	`PipedReader` `PipedWriter`	`PipedInputStream` `PipedOutputStream`

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

A Simple Example

This program performs a copy file operation using a manual buffer:

`java TestNodeStreams file1 file2`

```
import java.io.*;

public class TestNodeStreams {
  public static void main(String[] args) {
    try {
      FileReader input = new FileReader(args[0]);
      FileWriter output = new FileWriter(args[1]);
      char[]     buffer = new char[128];
      int        charsRead;

      // read the first buffer
      charsRead = input.read(buffer);
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

A Simple Example

```

        while ( charsRead != -1 ) {
          // write the buffer out to the output file
          output.write(buffer, 0, charsRead);

          // read the next buffer
          charsRead = input.read(buffer);
        }

        input.close();
        output.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Buffered Streams

This program performs a copy file operation using a built-in buffer:

java TestBufferedStreams file1 file2

```
import java.io.*;

public class TestBufferedStreams {
  public static void main(String[] args) {
    try {
      FileReader     input     = new FileReader(args[0]);
      BufferedReader bufInput  = new BufferedReader(input);
      FileWriter     output    = new FileWriter(args[1]);
      BufferedWriter bufOutput = new BufferedWriter(output);
      String line;

      // read the first line
      line = bufInput.readLine();
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

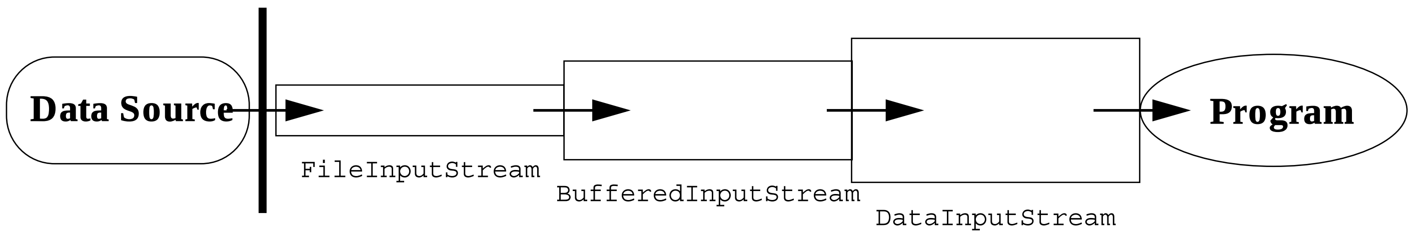
Buffered Streams

```

      while ( line != null ) {
        // write the line out to the output file
        bufOutput.write(line, 0, line.length());
        bufOutput.newLine();
        // read the next line
        line = bufInput.readLine();
      }
      bufInput.close();
      bufOutput.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

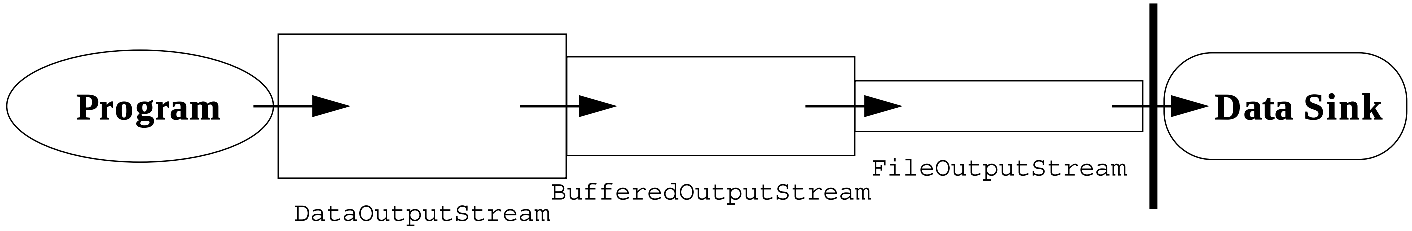

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

I/O Stream Chaining

Input Stream Chain



Output Stream Chain



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Processing Streams

Type	Character Streams	Byte Streams
Buffering	`BufferedReader` `BufferedWriter`	`BufferedInputStream` `BufferedOutputStream`
Filtering	*`FilterReader`* *`FilterWriter`*	*`FilterInputStream`* *`FilterOutputStream`*
Converting between bytes and character	`InputStreamReader` `OutputStreamWriter`	
Performing object serialization		`ObjectInputStream` `ObjectOutputStream`

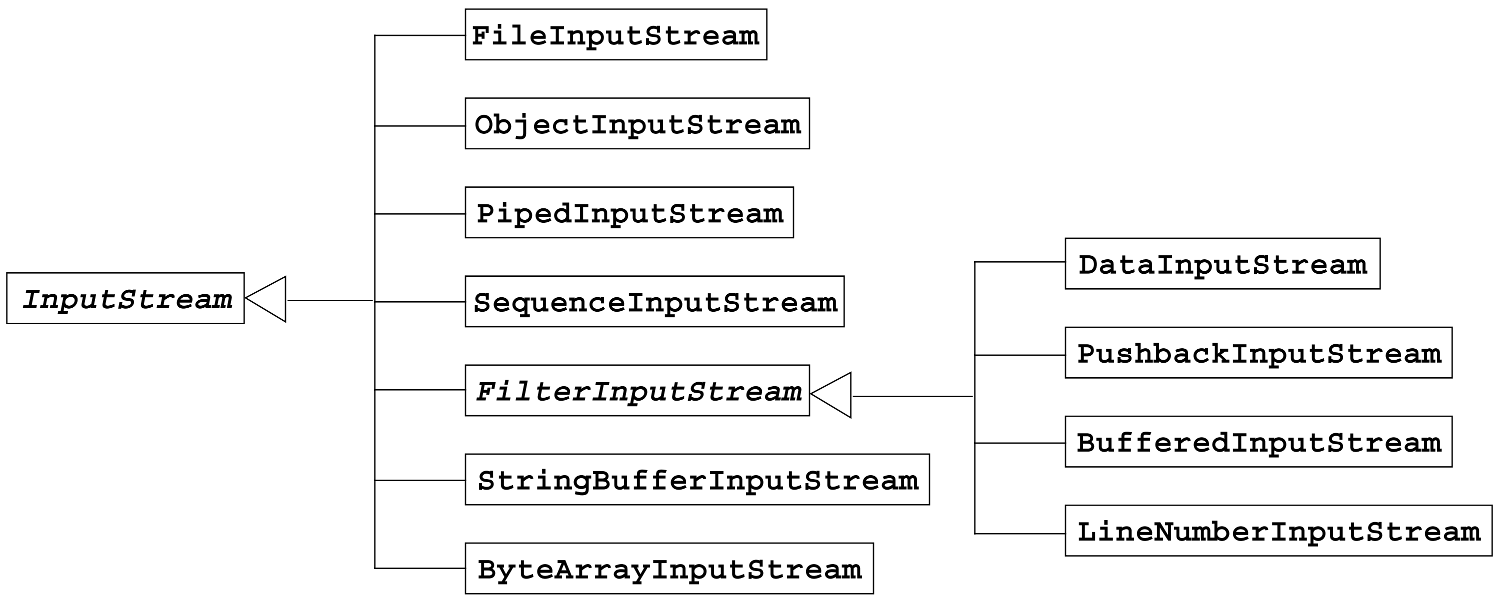
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Processing Streams

Type	Character Streams	Byte Streams
Performing data conversion		`DataInputStream` `DataOutputStream`
Counting	`LineNumberReader`	`LineNumberInputStream`
Peeking ahead	`PushbackReader`	`PushbackInputStream`
Printing	`PrintWriter`	`PrintStream`

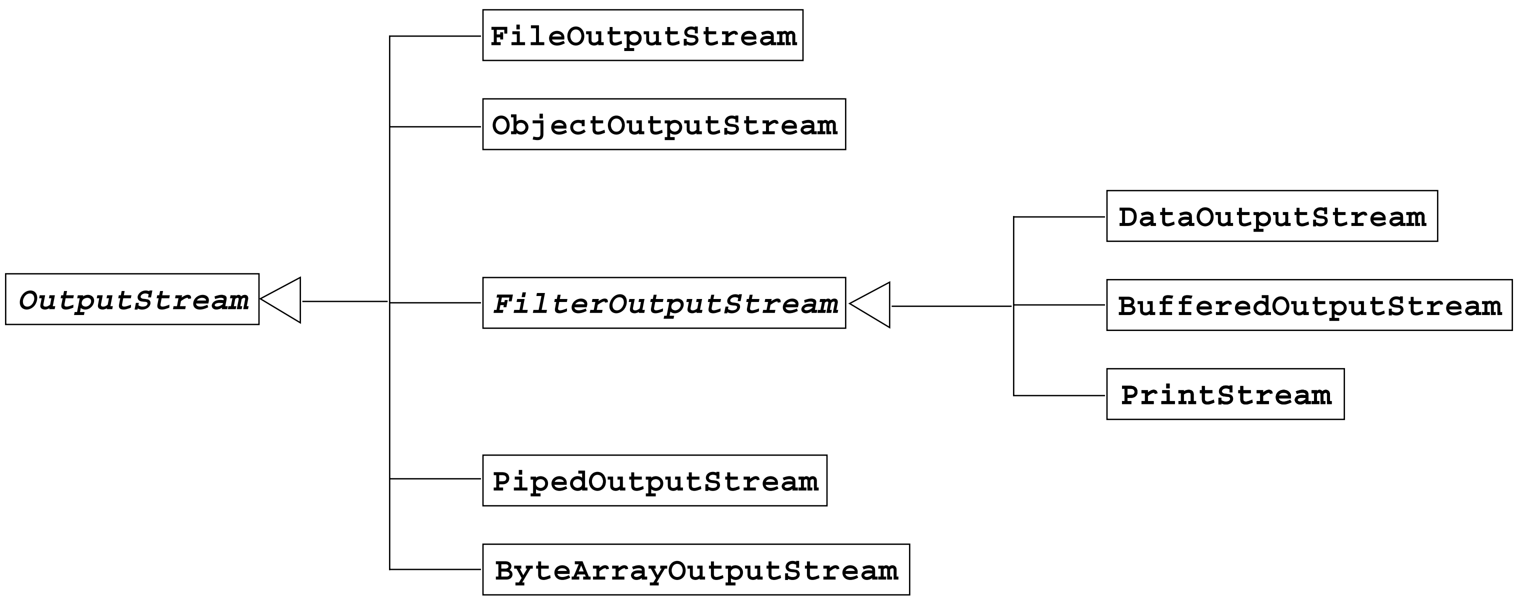
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The InputStream Class Hierarchy



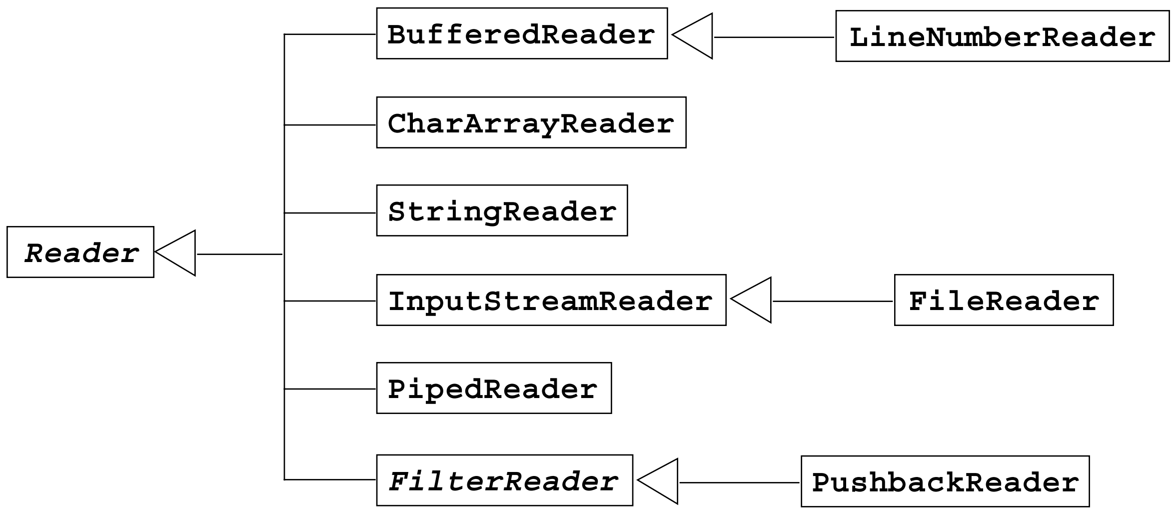
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `OutputStream` Class Hierarchy



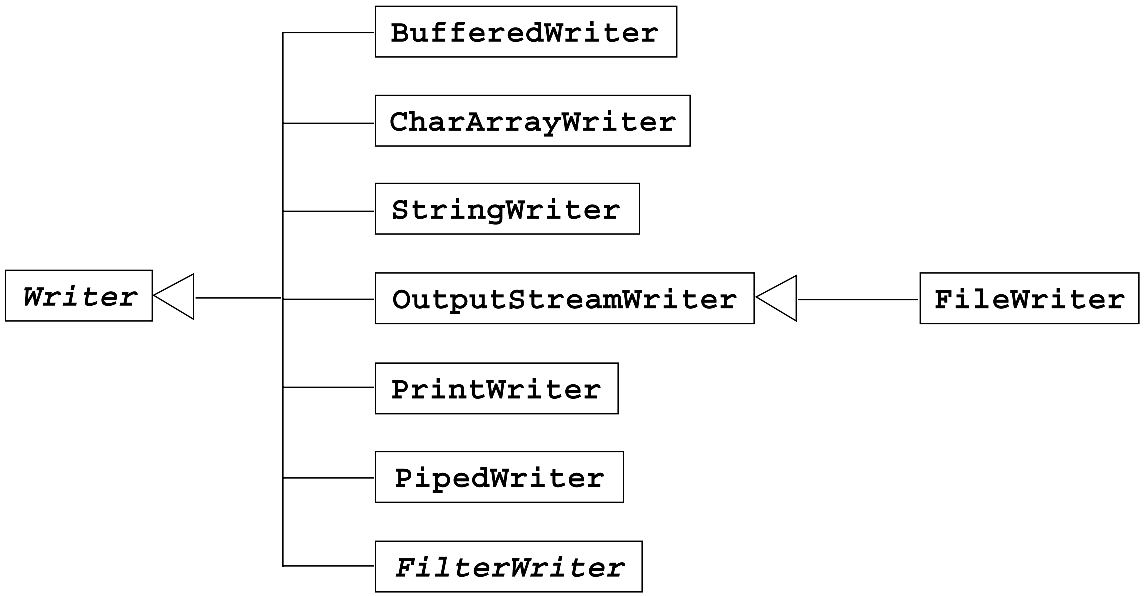
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The Reader Class Hierarchy



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

The `Writer` Class Hierarchy



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Module 15

Networking

Objectives

- Develop code to set up the network connection
- Understand the TCP/IP Protocol
- Use `ServerSocket` and `Socket` classes for implementation of TCP/IP clients and servers

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Relevance

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

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Networking

This section describes networking concepts.

Sockets

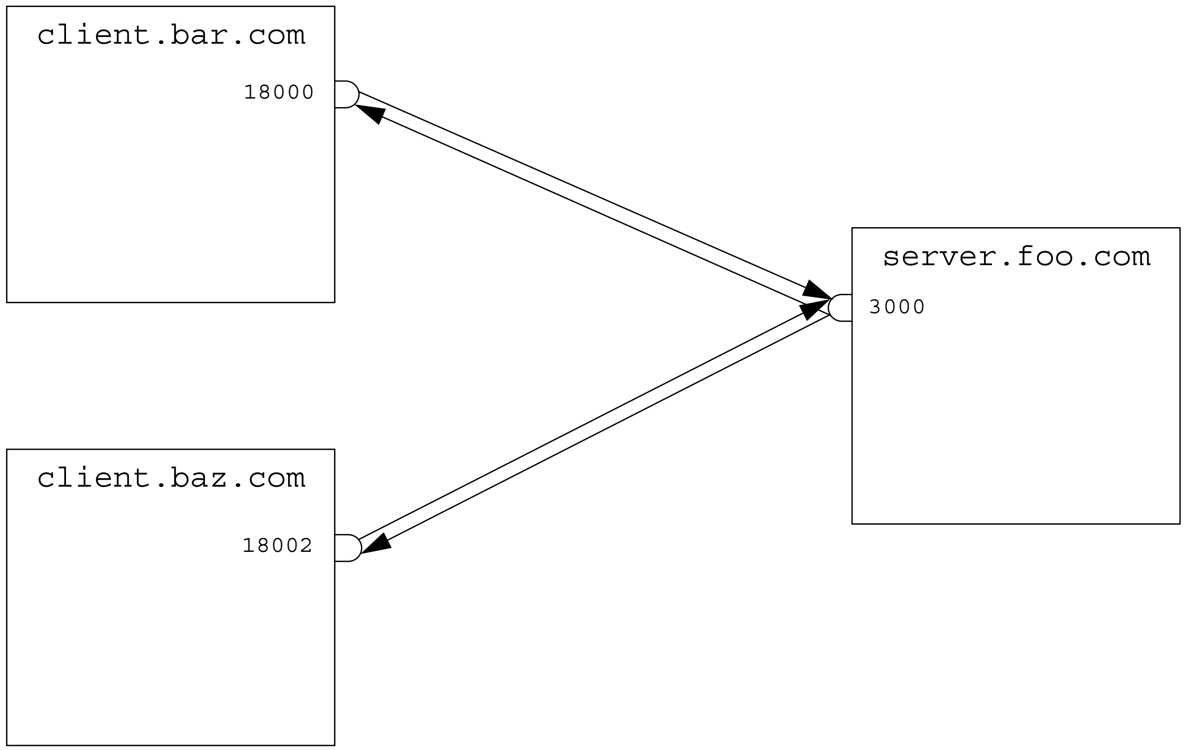
- Sockets hold two streams: an input stream and an output stream.
- Each end of the socket has a pair of streams.

Setting Up the Connection

Set up of a network connection is similar to a telephone system: One end must *dial* the other end, which must be *listening*.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Networking



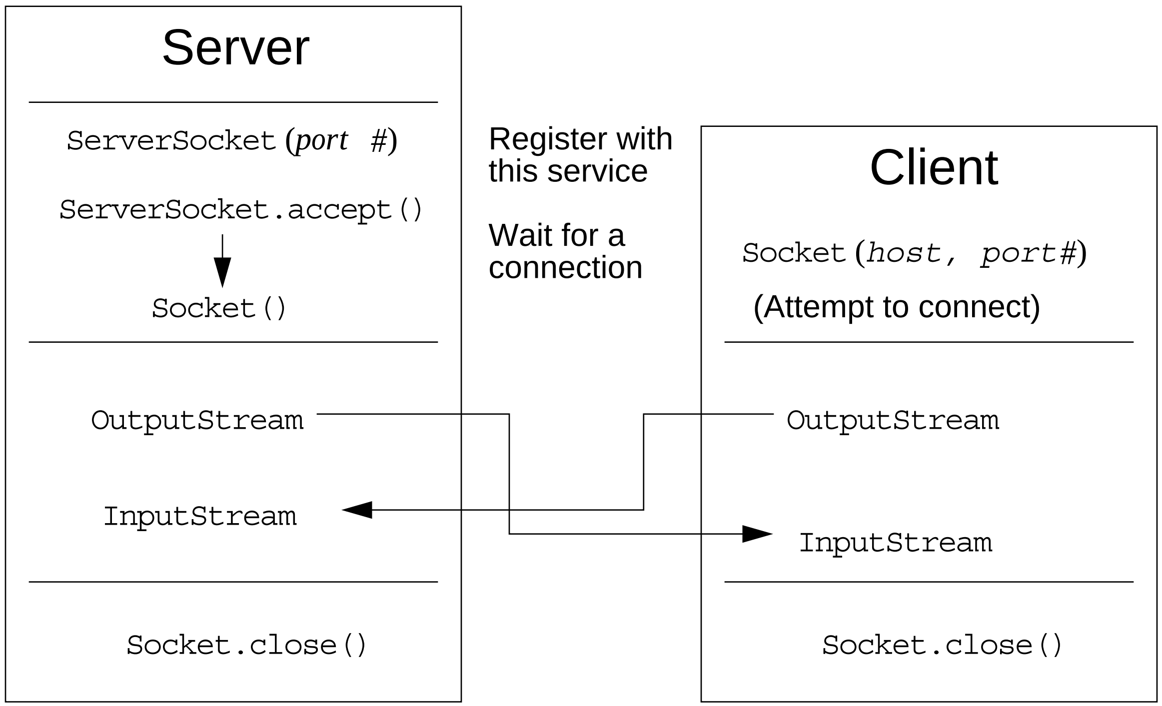
Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Networking With Java Technology

- To address the connection, include the following:
  - The address or name of remote machine
  - A port number to identify the purpose at the server
- Port numbers range from 0–65535.

Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Java Networking Model



Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Minimal TCP/IP Server

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

public class SimpleServer {
  public static void main(String args[]) {
    ServerSocket s = null;

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


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Minimal TCP/IP Server

```

    // Run the listen/accept loop forever
    while (true) {
      try {
        // Wait here and listen for a connection
        Socket s1 = s.accept();

        // Get output stream associated with the socket
        OutputStream s1out = s1.getOutputStream();
        BufferedWriter bw = new BufferedWriter(
          new OutputStreamWriter(s1out));

        // Send your string!
        bw.write("Hello Net World!\n");
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Minimal TCP/IP Server

```

            // Close the connection, but not the server socket
            bw.close();
            s1.close();

          } catch (IOException e) {
            e.printStackTrace();
          } // END of try-catch

        } // END of while(true)

      } // END of main method

    } // END of SimpleServer program
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Minimal TCP/IP Client

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

public class SimpleClient {

  public static void main(String args[]) {

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

      // Get an input stream from the socket
      InputStream is = s1.getInputStream();
      // Decorate it with a "data" input stream
      DataInputStream dis = new DataInputStream(is);
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1

Minimal TCP/IP Client

```

      // Read the input and print it to the screen
      System.out.println(dis.readUTF());

      // When done, just close the steam and connection
      dis.close();
      s1.close();

    } catch (ConnectException connExc) {
      System.err.println("Could not connect.");

    } catch (IOException e) {
      // ignore
    } // END of try-catch

  } // END of main method

} // END of SimpleClient program
```


Copyright 2006 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision F.1