

Sun Educational Services

Fundamentals of the Java™ Programming Language

SL-110



Copyright 2002 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303, 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, Java 2 Platform, Standard Edition, J2EE, J2ME, J2SE, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

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.

.THIS MANUAL IS DESIGNED TO SUPPORT AN INSTRUCTOR-LED TRAINING (ILT) COURSE AND IS INTENDED TO BE USED FOR REFERENCE PURPOSES IN CONJUNCTION WITH THE ILT COURSE. THE MANUAL IS NOT A STANDALONE TRAINING TOOL. USE OF THE MANUAL FOR SELF-STUDY WITHOUT CLASS ATTENDANCE IS NOT RECOMMENDED.

Copyright 2002 Sun Microsystems Inc., 901 San Antonio Road, Palo Alto, California 94303, Etats-Unis. Tous droits réservés.

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

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

Sun, Sun Microsystems, le logo Sun, Java, Java 2 Platform, Standard Edition, J2EE, J2ME, J2SE, et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

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.

CE MANUEL DE RÉFÉRENCE DOIT ÊTRE UTILISÉ DANS LE CADRE D'UN COURS DE FORMATION DIRIGÉ PAR UN INSTRUCTEUR (ILT). IL NE S'AGIT PAS D'UN OUTIL DE FORMATION INDÉPENDANT. NOUS VOUS DÉCONSEILLONS DE L'UTILISER DANS LE CADRE D'UNE AUTO-FORMATION.



About This Course



Course Goals

Upon completion of this course, you should be able to:

- Demonstrate knowledge of Java™ technology, the Java programming language, and the product life cycle
- Use various Java programming language constructs to create several Java technology applications
- Use decision and looping constructs and methods to dictate program flow
- Implement intermediate Java technology programming and object-oriented (OO) concepts in Java technology programs



Course Map

Introducing Java Technology Programming

Explaining Java™
Technology

Analyzing a Problem
and Designing
a Solution

Developing and
Testing a Java
Technology Program

Explaining Java Technology Programming Fundamentals

Declaring,
Initializing, and
Using Variables

Creating and
Using Objects

Dictating Program Flow

Using Operators
and Decision
Constructs

Using Loop
Constructs

Developing and
Using Methods

Describing Intermediate Java Technology and OO Concepts

Implementing
Encapsulation and
Constructors

Creating and
Using Arrays

Implementing
Inheritance



Topics Not Covered

- Advanced Java technology programming – Covered in SL-275: *Java™ Programming Language*
- Advanced OO analysis and design – Covered in OO-226: *Object-Oriented Application Analysis and Design for Java™ Technology (UML)*
- Applet programming or web page design



How Prepared Are You?

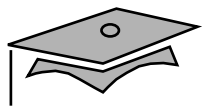
To be sure you are prepared to take this course, can you answer yes to the following questions?

- Can you create programs using a procedural language, such as C, or a scripting language, such as Perl?
- Can you create and edit text files using a text editor?
- Can you use a World Wide Web (WWW) browser?
- Can you solve logic problems?



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



Icons



Demonstration



Discussion



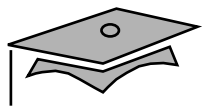
Note



Caution - Electrical



Caution - Heat



Icons



Case Study



Self-Check



Typographical Conventions

- `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.



Typographical Conventions

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



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 Environment (Solaris OE) is different from the Microsoft Windows platform, both commands are shown.



Module 1

Explaining Java™ Technology



Overview

- Objectives:
 - Describe key concepts of the Java programming language
 - List the three Java technology product groups
 - Summarize each of the seven stages in the product life cycle
- Relevance

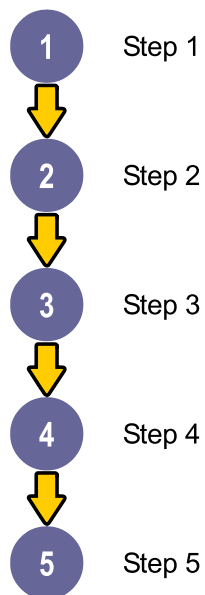


Key Concepts of the Java Programming Language

- Object-oriented
- Distributed
- Simple
- Multithreaded
- Secure
- Platform-independent

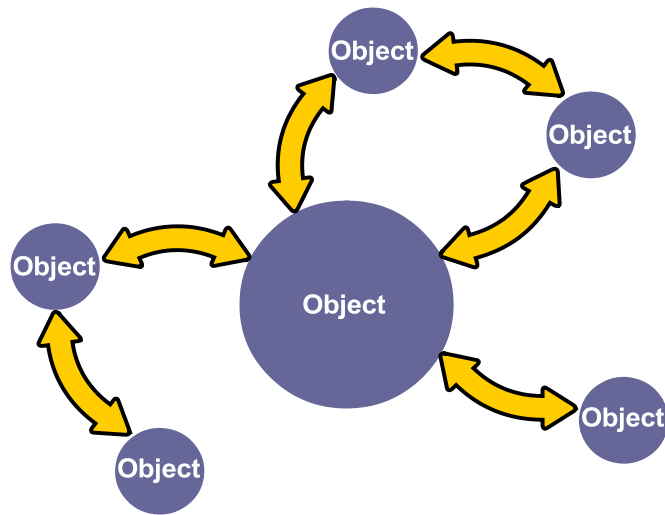


Object-Oriented



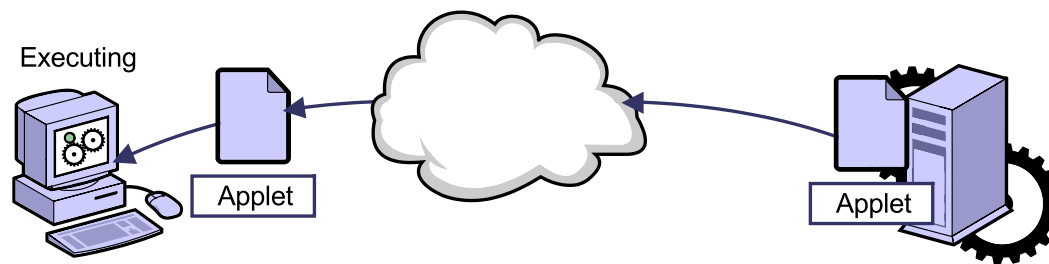


Object-Oriented





Distributed



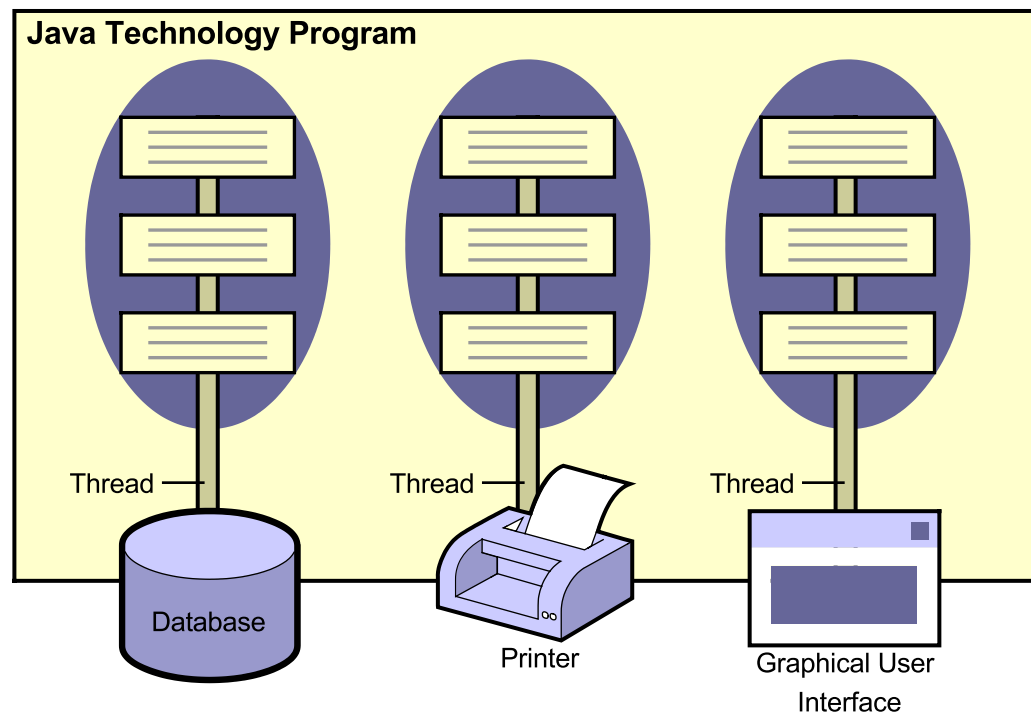


Simple

- References are used instead of pointers.
- A boolean data type can have a value of either true or false.

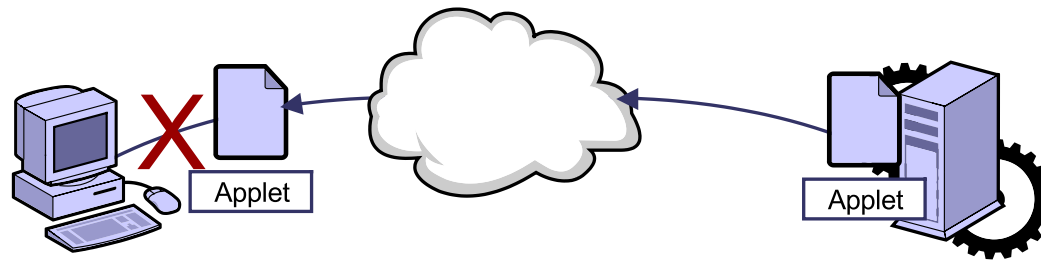


Multithreaded



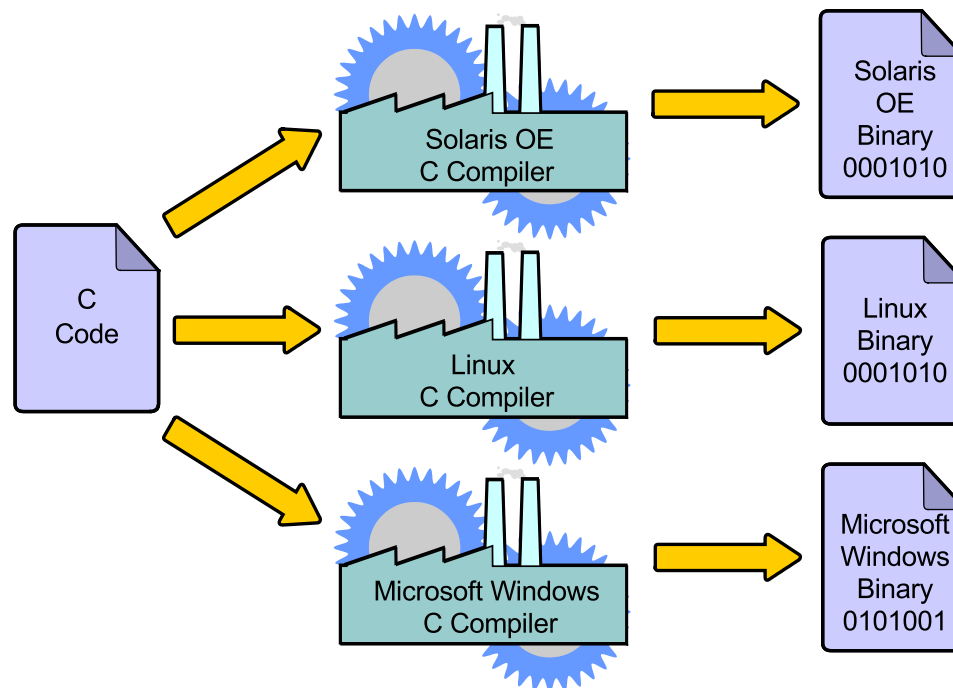


Secure



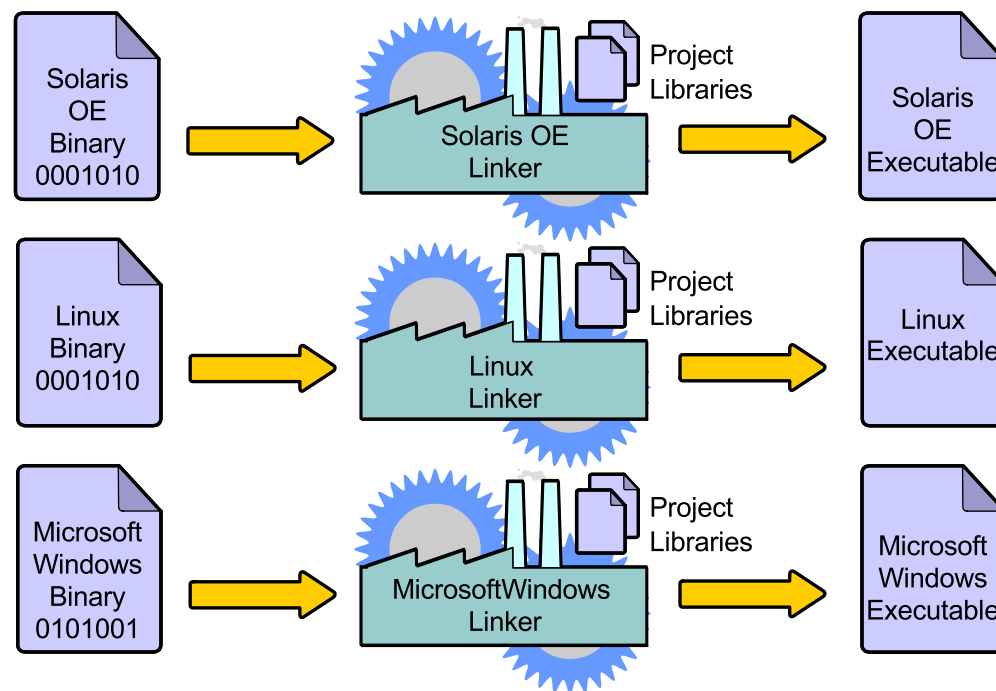


Platform-Dependent Programs



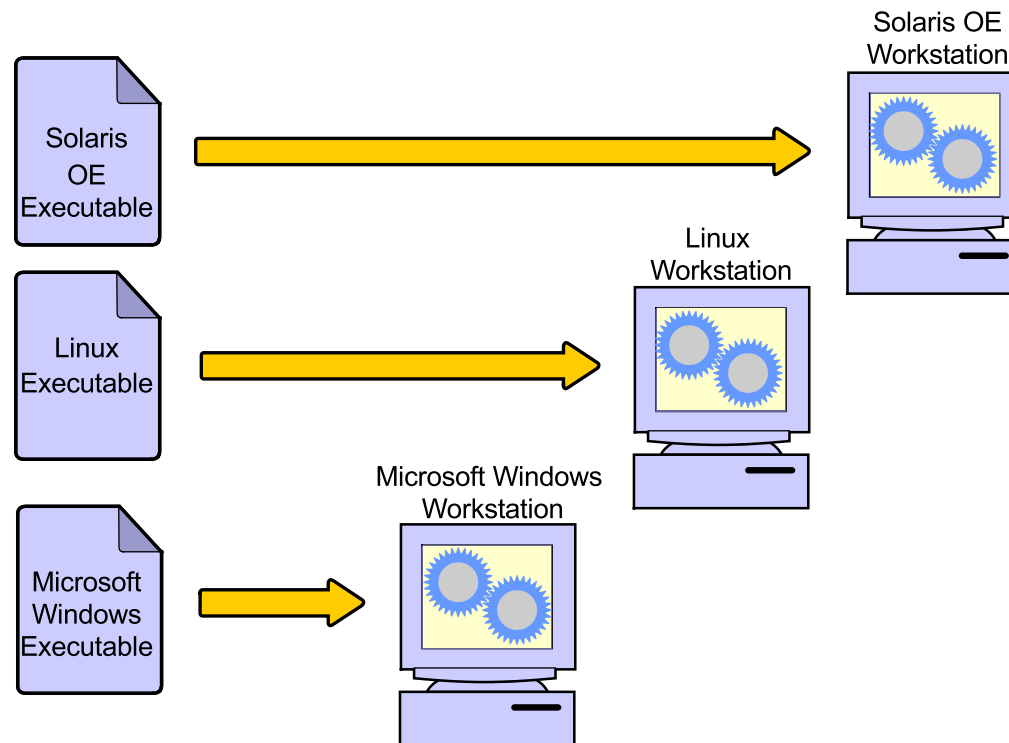


Platform-Dependent Programs



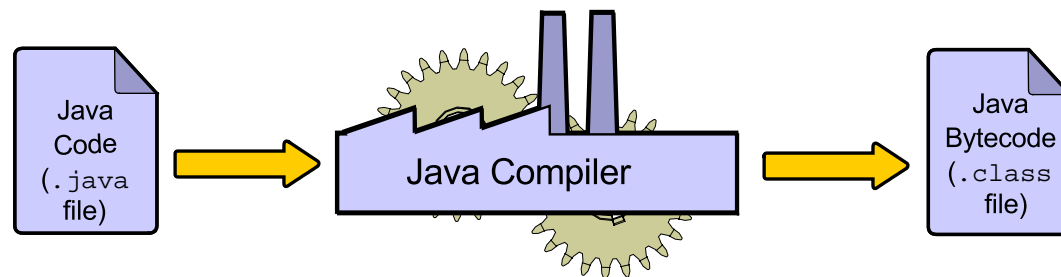


Platform-Dependent Programs



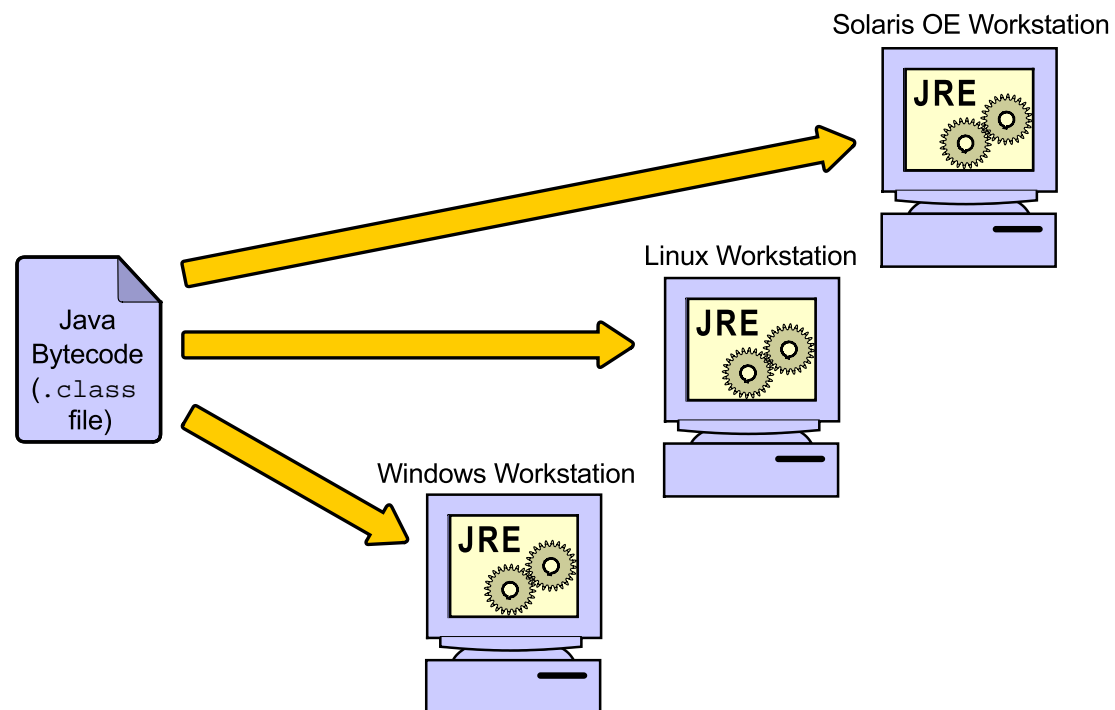


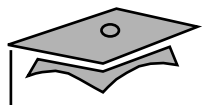
Platform-Independent Programs



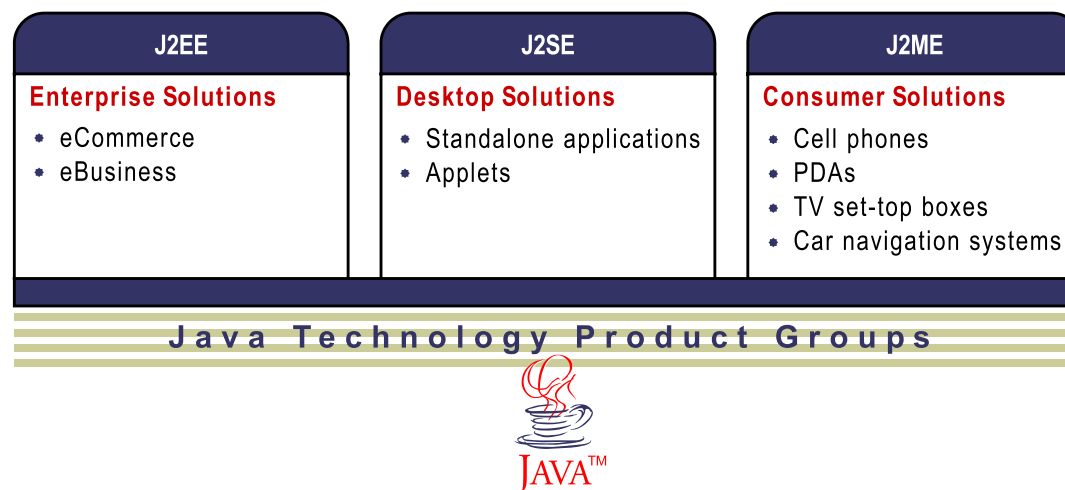


Platform-Independent





Identifying Java Technology Product Groups





Using the Java™ 2 Platform, Standard Edition SDK Components

- Java™ runtime environment:
 - A Java™ virtual machine for the platform you choose
 - Java class libraries for the platform you choose
- A Java technology compiler
- Java class library (API) documentation (as a separate download)
- Additional utilities, such as utilities for creating Java archive files (JAR files) and for debugging Java technology programs
- Examples of Java technology programs

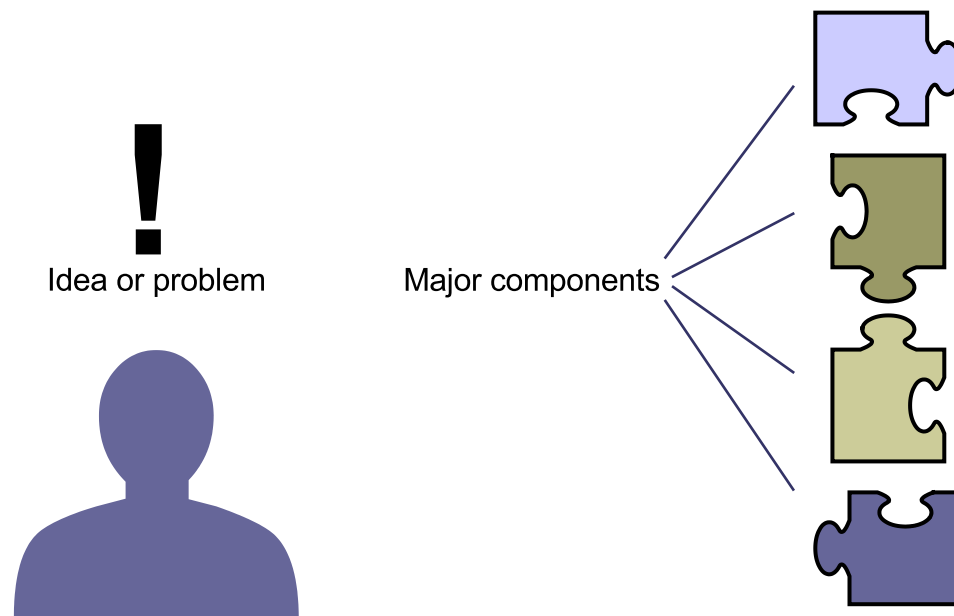


Product Life Cycle (PLC) Stages

1. Analysis
2. Design
3. Development
4. Testing
5. Implementation
6. Maintenance
7. End-of-life (EOL)

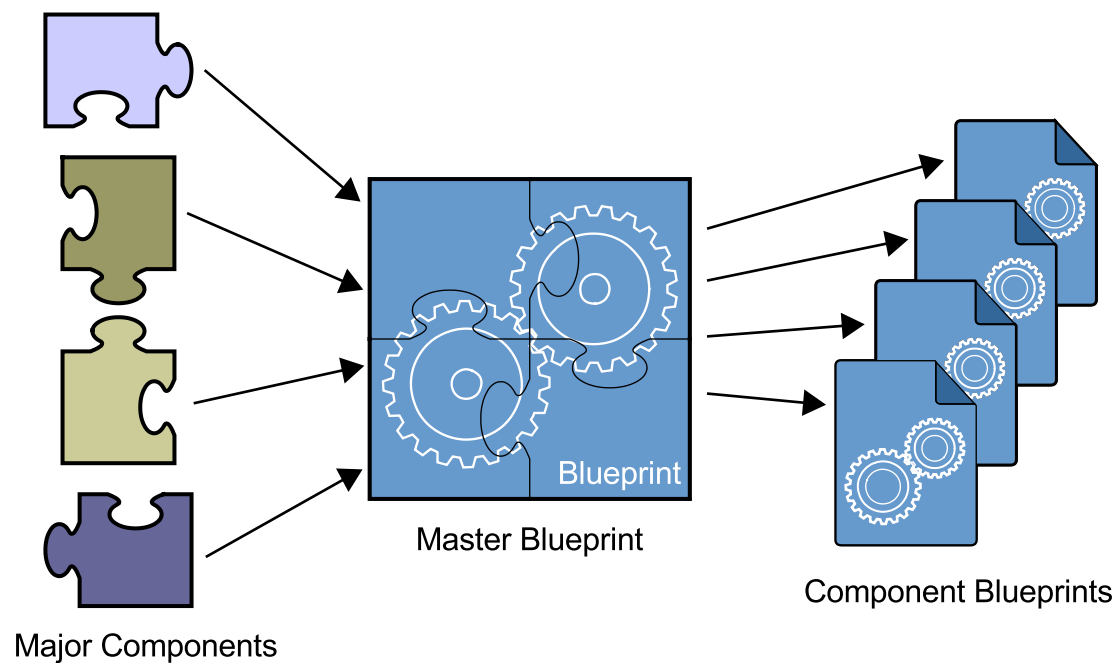


Analysis Stage



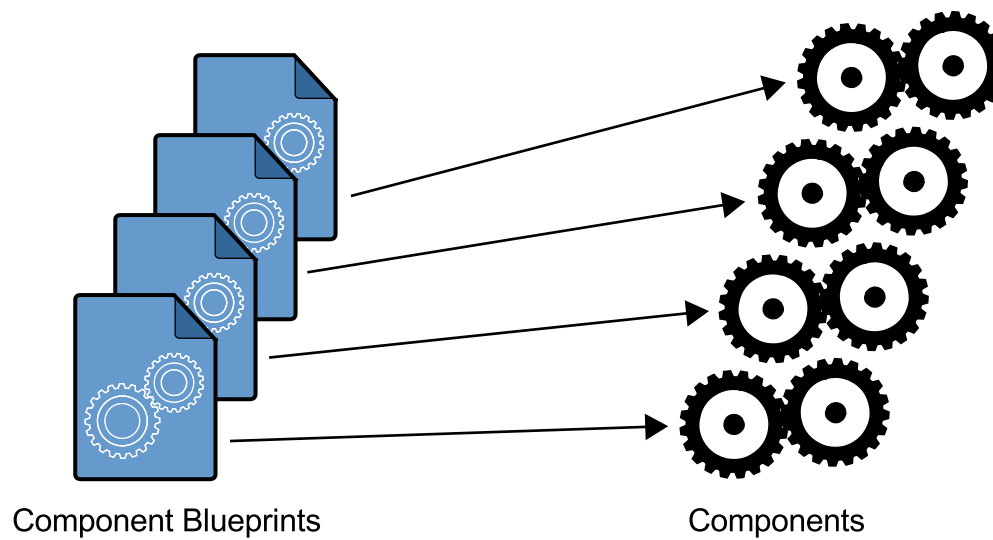


Design Stage



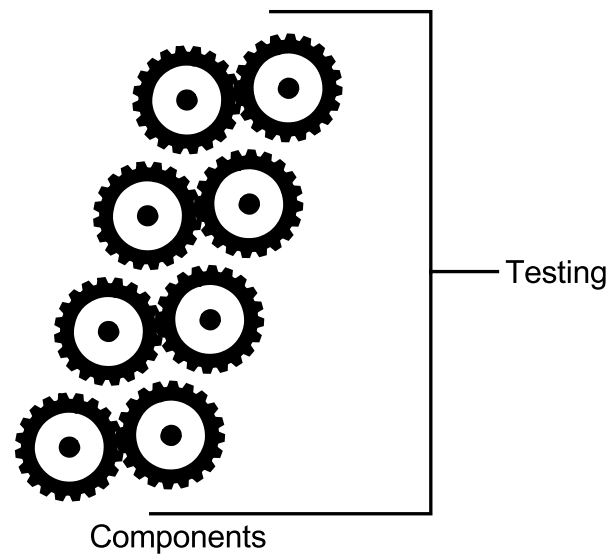


Development Stage



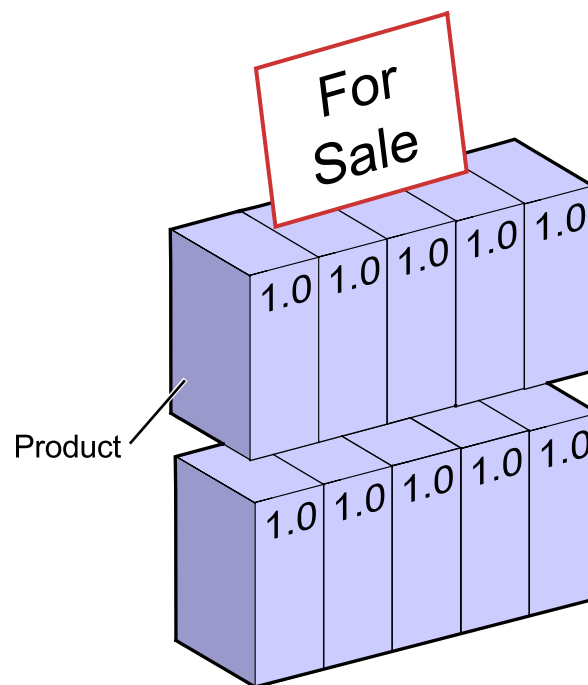


Testing Stage



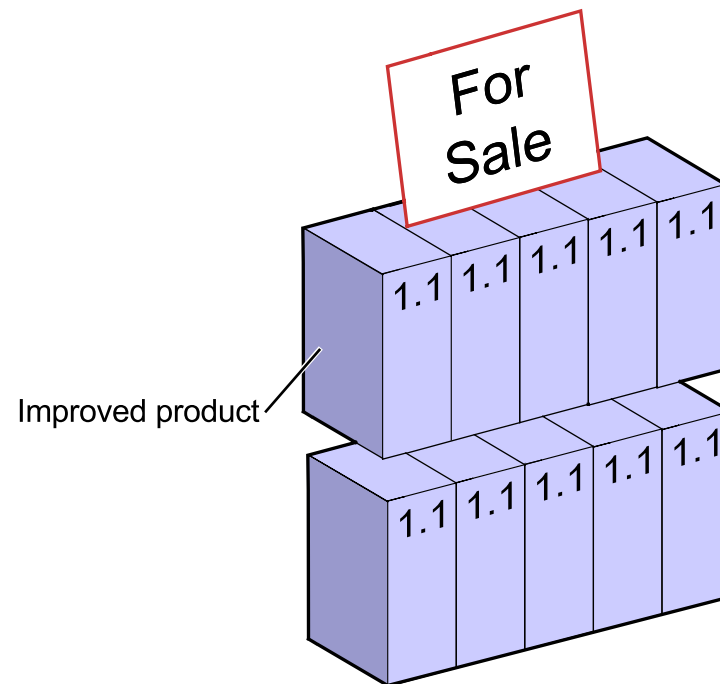


Implementation Stage



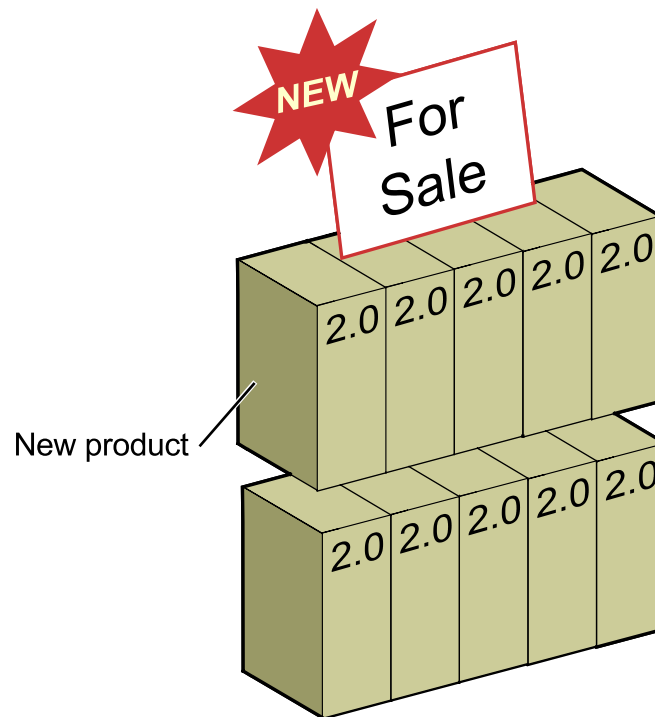


Maintenance Stage





End-of-Life (EOL) Stage





Module 2

Analyzing a Problem and Designing a Solution



Overview

- Objectives:
 - Analyze a problem using object-oriented analysis
 - Design classes from which objects will be created
- Relevance



Analyzing a Problem Using Object-Oriented Analysis

DirectClothing, Inc.
"The Shirt Company"

Name: _____ Date: _____

Address: _____

Phone #: _____

Email: _____

Shirt ID #	Size	Color Code	Price

Total Price _____

Payment: Check ☐ Credit Card ☐

Credit Card # _____

Expiration _____



Identifying a Problem Domain

- A problem domain is the scope of the problem you will solve.
- For example, “Create a system allowing order entry people to enter and accept payment for an order.”

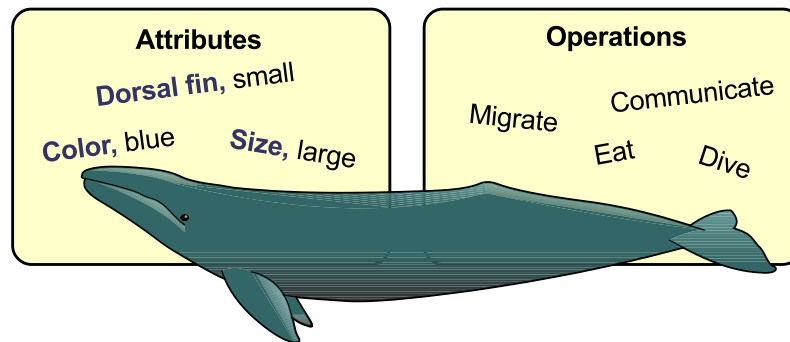


Identifying Objects

- Objects can be physical or conceptual.
- Objects have *attributes* (characteristics), such as size, name, shape, and so on.
- Objects have *operations* (the things they can do), such as setting a value, displaying a screen, or increasing speed.



Identifying Objects





Additional Criteria for Recognizing Objects

- Relevance to the problem domain:
 - Does the object exist within the boundaries of the problem domain?
 - Is the object required for the solution to be complete?
 - Is the object required as part of an interaction between a user and the system?
- Independent existence



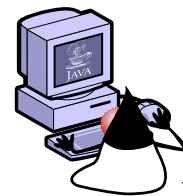
Possible Objects in the DirectClothing Case Study



Order



Shirt



Customer

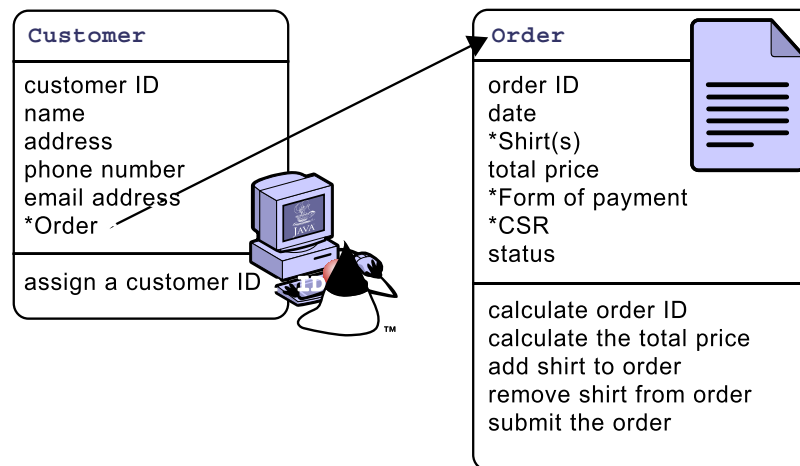


Identifying Object Attributes and Operations

- Attributes are data, such as:
 - ID
 - Order object
- Operations are actions, such as:
 - Delete item
 - Change ID

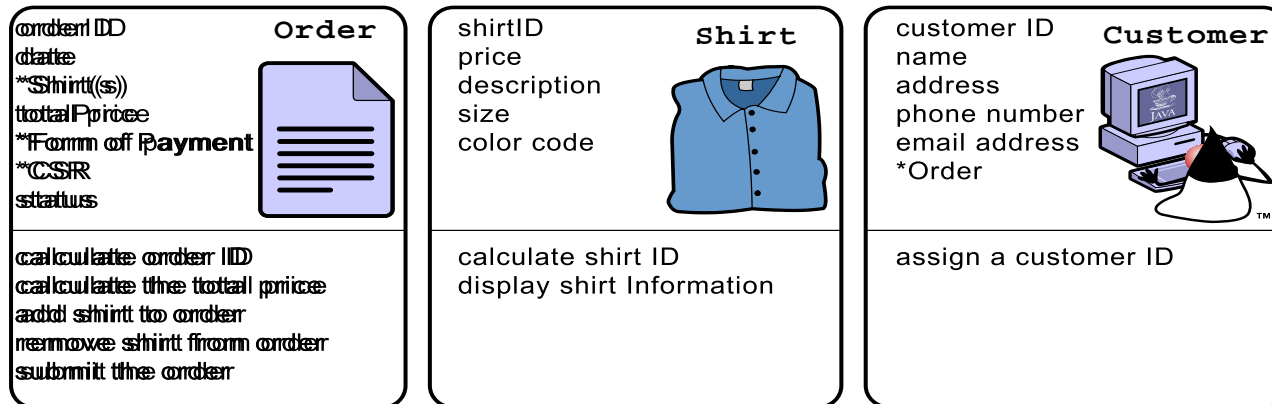


Object With Another Object as an Attribute





Possible Attributes and Operations in the DirectClothing, Inc. Case Study





Case Study Solution

Order	Shirt
order ID date *Shirt(s) total price *Form of payment *CSR status	shirt ID price description size color code
calculate order ID calculate the total price add shirt to order remove shirt from order submit the order	calculate shirt ID display shirt information



Case Study Solution

Customer	Form of Payment
customer ID name address phone number email address *Order	check number credit card number expiration date
assign a customer ID	verify credit card number verify check payment



Case Study Solution

Catalog	CSR
*Shirt(s)	name extension
add a shirt remove a shirt	



Exercise 1: Analyzing a Problem Domain

- Objective
- Preparation
- Task
- Solution
- Discussion



Designing Classes

Whale Attributes

Dorsal Fin

Color

Size

Dorsal Fin, large

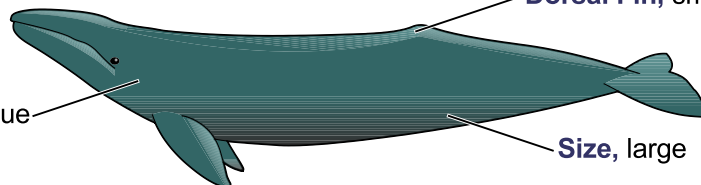
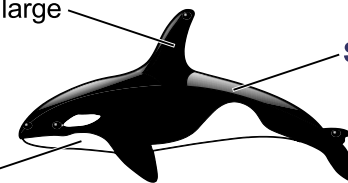
Size, medium

Color, black and white

Dorsal Fin, small

Color, blue

Size, large

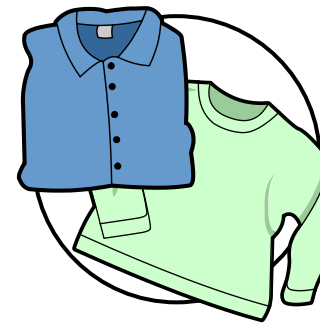




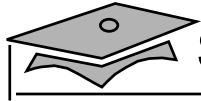
Class and Resulting Objects

Shirt
shirtID price description size colorCode R=Red, B=Blue, G=Green
calculateShirtID() displayShirtInformation()

Shirt Class



Shirt Objects



Modeling Classes

- Syntax

ClassName
attributevariableName [<i>range of values</i>] attributevariableName [<i>range of values</i>] attributevariableName [<i>range of values</i>] ...
methodName() methodName() methodName() ...

- Example

Shirt
shirtID price description size colorCode R=Red, B=Blue, G=Green
calculateShirtID() displayShirtInformation()



Exercise 2: Designing a Solution

- Objective
- Task
- Solution
- Discussion



Module 3

Developing and Testing a Java Technology Program

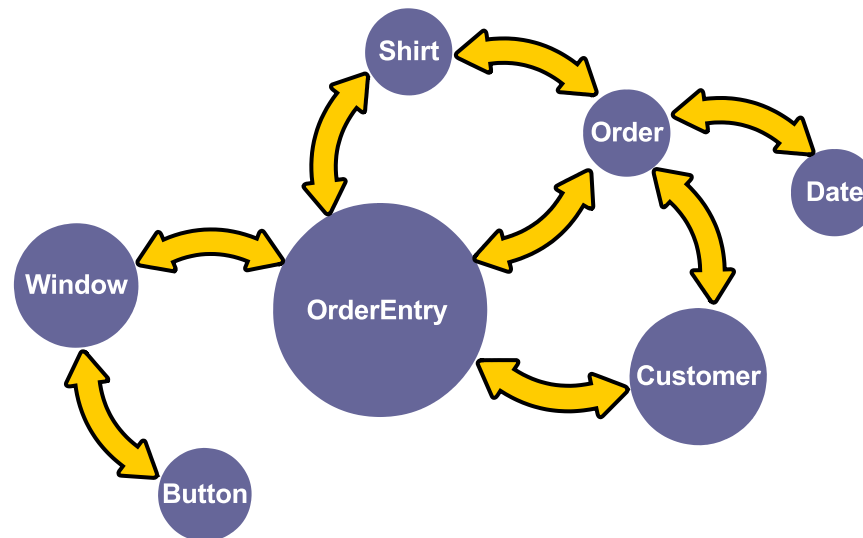


Overview

- Objectives:
 - Identify the four components of a class in the Java programming language
 - Use the `main` method in a test class to run a Java technology program from the command line
 - Compile and execute a Java technology program
- Relevance



Identifying the Components of a Class





Structuring Classes

- The class declaration
- Attribute variable declarations and initialization (optional)
- Methods (optional)
- Comments (optional)



Structuring Classes

```
1  public class Shirt {
2
3      public int shirtID = 0; // Default ID for the shirt
4      public String description = "-description required-"; // default
5
6      // The color codes are R=Red, B=Blue, G=Green, U=Unset
7      public char colorCode = 'U';
8
9      public double price = 0.0; // Default price for all shirts
10
11     public int quantityInStock = 0; // Default quantity for all shirts
12
13     // This method displays the values for an item
14     public void displayShirtInformation() {
15
16         System.out.println("Shirt ID: " + shirtID);
17         System.out.println("Shirt description:" + description);
18         System.out.println("Color Code: " + colorCode);
19         System.out.println("Shirt price: " + price);
```



```
20      System.out.println("Quantity in stock: " + quantityInStock);
21
22  } // end of display method
23 } // end of class
```



Class Declaration

- Syntax:

`[modifier] class class_identifier`

- Example:

```
public class Shirt
```




Variable Declarations and Assignments

```
public int shirtID = 0;  
public String description = "-description required-";  
public char colorCode = 'U';  
public double price = 0.0;  
public int quantityInStock = 0;
```



Comments

- Single-line:

```
public int shirtID = 0; // Default ID for the shirt
public double price = 0.0; // Default price for all shirts

// The color codes are R=Red, B=Blue, G=Green
```

- Traditional:

```
/******
 * Attribute Variable Declaration Section      *
 *****/
```



Methods

- Syntax:

```
[modifiers] return_type method_identifier ([arguments]) {  
    method_code_block  
}
```

- Example:

```
public void displayShirtInformation() {  
  
    System.out.println("Shirt ID: " + shirtID);  
    System.out.println("Shirt description:" + description);  
    System.out.println("Color Code: " + colorCode);  
    System.out.println("Shirt price: " + price);  
    System.out.println("Quantity in stock: " + quantityInStock);  
  
} // end of display method
```



Creating and Using a Test Class

Example:

```
1  public class ShirtTest {  
2  
3      public static void main (String args[]) {  
4  
5          Shirt myShirt;  
6          myShirt = new Shirt();  
7  
8          myShirt.displayShirtInformation();  
9  
10     }  
11 }
```



The main Method

Syntax:

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



Compiling a Program

1. Go the directory where the source code files are stored.
2. Enter the following command for each .java file you want to compile.

- Syntax:

```
javac filename
```

- Example:

```
javac Shirt.java
```



Executing (Testing) a Program

1. Go the directory where the class files are stored.
2. Enter the following for the class file that contains the main method.

- Syntax

```
java classname
```

- Example

```
java ShirtTest
```

- Output:

```
Shirt ID: 0  
Shirt description:-description required-  
Color Code: U  
Shirt price: 0.0  
Quantity in stock: 0
```



Debugging Tips

- Error messages state the line number where the error occurs. That line might not always be the actual source of the error.
- Be sure that you have a semicolon at the end of every line where one is required, and no others.
- Be sure that you have an even number of braces.
- Be sure that you have used consistent indentation in your program, as shown in examples in this course.



Exercise: Writing, Compiling, and Testing a Basic Program

- Objective
- Tasks
- Solutions
- Discussion



Module 4

Declaring, Initializing, and Using Variables



Overview

- Objectives:
 - Identify the uses for variables and define the syntax for a variable
 - List the eight Java programming language primitive data types
 - Declare, initialize, and use variables and constants according to Java programming language guidelines and coding standards
 - Modify variable values using operators
 - Use promotion and type casting
- Relevance



Identifying Variable Use and Syntax

Example:

```
1  public class Shirt {
2
3      public int shirtID = 0; // Default ID for the shirt
4      public String description = "-description required-"; // default
5
6      // The color codes are R=Red, B=Blue, G=Green, U=Unset
7      public char colorCode = 'U';
8
9      public double price = 0.0; // Default price for all shirts
10
11     public int quantityInStock = 0; // Default quantity for all shirts
12
13     // This method displays the values for an item
14     public void displayShirtInformation() {
15
16         System.out.println("Shirt ID: " + shirtID);
17         System.out.println("Shirt description:" + description);
```



```
18      System.out.println("Color Code: " + colorCode);
19      System.out.println("Shirt price: " + price);
20      System.out.println("Quantity in stock: " + quantityInStock);
21
22  } // end of display method
23 } // end of class
```



Uses for Variables

- Holding unique data for an object instance
- Assigning the value of one variable to another
- Representing values within a mathematical expression
- Printing the values to the screen
- Holding references to other objects



Variable Declaration and Initialization

- Syntax (attribute or instance variables):

[modifiers] type identifier = value;

- Syntax (local variables):

type identifier;

- Syntax (local variables)

type identifier = value;

- Examples:

```
public int shirtID = 0;
public String description = "-description required-";
public char colorCode = 'U';
public double price = 0.0;
public int quantityInStock = 0;
```



Describing Primitive Data Types

- Integral types (`byte`, `short`, `int`, and `long`)
- Floating point types (`float` and `double`)
- Textual type (`char`)
- Logical type (`boolean`)



Integral Primitive Types

Type	Length	Range	Examples of Allowed Literal Values
byte	8 bits	-2^7 to $2^7 - 1$ (-128 to 127, or 256 possible values)	2 -114
short	16 bits	-2^{15} to $2^{15} - 1$ (-32,768 to 32,767, or 65,535 possible values)	2 -32699
int	32 bits	-2^{31} to $2^{31} - 1$ (-2,147,483,648 to 2,147,483,647 or 4,294,967,296 possible values)	2 147334778



Integral Primitive Types

Type	Length	Range	Examples of Allowed Literal Values
long	64 bits	-2^{63} to $2^{63} - 1$ (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807, or 18,446,744,073,709,551,616 possible values)	2 -2036854775808L 1L



Integral Primitive Types

```
public int shirtID = 0; // Default ID for the shirt  
public int quantityInStock = 0; // Default quantity for all shirts
```



Floating Point Primitive Types

Type	Float Length	Examples of Allowed Literal Values
float	32 bits	99F -327456,99.01F 4.2E6F (engineering notation for $4.2 * 10^6$)
double	64 bits	-1111 2.1E12 99970132745699.999

```
public double price = 0.0; // Default price for all shirts
```



Textual Primitive Type

- The only data type is char
- Used for a single character (16 bits), such as a “y”
- Example:

```
public char colorCode = 'U';
```



Logical Primitive Type

- The only data type is `boolean`
- Can store only `true` or `false`
- Holds the result of an expression that evaluates to either `true` or `false`



Naming a Variable

- Rules:
 - Variable identifiers must start with either an uppercase or lowercase letter, an underscore (_), or a dollar sign (\$).
 - Variable identifiers cannot contain punctuation, spaces, or dashes.
 - Java technology keywords cannot be used.



Naming a Variable

- Guidelines:
 - Begin each variable with a lowercase letter; subsequent words should be capitalized, such as `myVariable`.
 - Chose names that are mnemonic and that indicate to the casual observer the intent of the variable.



Assigning a Value to a Variable

- Example:

```
double price = 12.99;
```

- Example (boolean):

```
boolean isOpen = false;
```



Declaring and Initializing Several Variables in One Line of Code

- Syntax:

```
type identifier = value [, identifier = value];
```

- Example:

```
double price = 0.0, wholesalePrice = 0.0;
```



Additional Ways to Declare Variables and Assign Values to Variables

- Assigning literal values:

```
int ID = 0;  
float pi = 3.14F;  
char myChar = 'G';  
boolean isOpen = false;
```

- Assigning the value of one variable to another variable:

```
int ID = 0;  
int saleID = ID;
```



Additional Ways to Declare Variables and Assign Values to Variables

- Assigning the result of an expression to integral, floating point, or Boolean variables

```
float numberOrdered = 908.5F;  
float casePrice = 19.99F;  
float price = (casePrice * numberOrdered);
```

```
int hour = 12;  
boolean isOpen = (hour > 8);
```

- Assigning the return value of a method call to a variable



Constants

- Variable (can change):

```
double salesTax = 6.25;
```

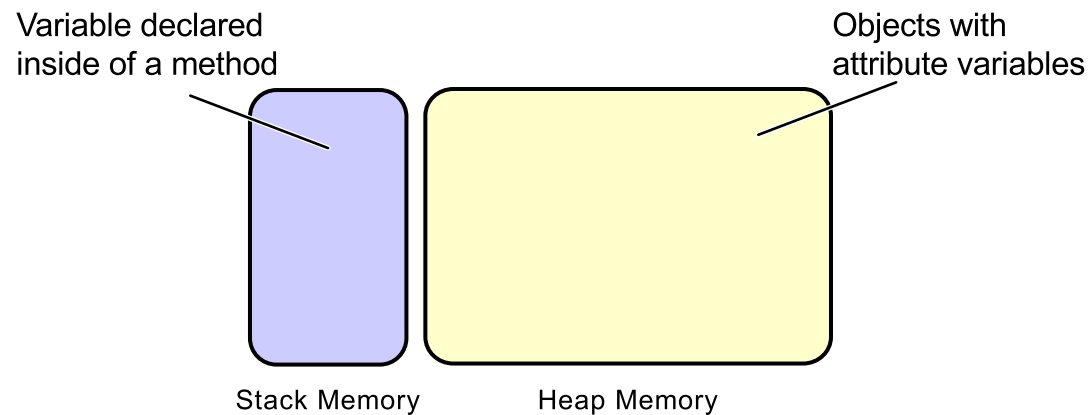
- Constant (cannot change):

```
final double SALES_TAX = 6.25;
```

- Guideline – Constants should be capitalized with words separated by an underscore (_).



Storing Primitives and Constants in Memory





Exercise 1: Using Primitive Type Variables in a Program

- Objective
- Task
- Solution
- Discussion



Standard Mathematical Operators

Purpose	Operator	Example	Comments
Addition	+	<code>sum = num1 + num2</code> If num1 is 10 and num2 is 2, sum is 12.	
Subtraction	-	<code>diff = num1 - num2</code> If num1 is 10 and num2 is 2, diff is 8.	
Multiplication	*	<code>prod = num1 * num2</code> If num1 is 10 and num2 is 2, prod is 20.	
Division	/	<code>quot = num1 / num2</code> If num1 is 31 and num2 is 6, quot is 5	Division returns an integer value (with no remainder).



Standard Mathematical Operators

Purpose	Operator	Example	Comments
Remainder	%	<code>mod = num1 % num2</code> If num1 is 31 and num2 is 6, mod is 1.	Remainder finds the remainder of the first number divided by the second number. $\begin{array}{r} 5 \text{ R } 1 \\ 6 \overline{) 31} \\ \underline{30} \\ 1 \end{array}$ Remainder always gives an answer with the same sign as the first operand.



Increment and Decrement Operators (++ and --)

- The long way:

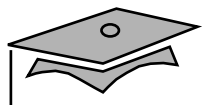
```
age = age + 1;
```



Increment and Decrement Operators (++ and --)

- The short way:

Operator	Purpose	Example	Notes
++	Pre-increment (++ <i>variable</i>)	<pre>int i = 6; int j = ++i; i is 7, j is 7</pre>	
	Post-increment (<i>variable</i> ++)	<pre>int i = 6; int j = i++; i is 7, j is 6</pre>	The value of <i>i</i> is assigned to <i>j</i> before <i>i</i> is incremented. Therefore, <i>j</i> is assigned 6.



Increment and Decrement Operators (++ and --)

Operator	Purpose	Example	Notes
--	Pre-decrement (-- <i>variable</i>)	<pre>int i = 6; int j = --i; i is 5, j is 5</pre>	
	Post-decrement (<i>variable</i> --)	<pre>int i = 6; int j = i--; i is 5, j is 6</pre>	The value <i>i</i> is assigned to <i>j</i> before <i>i</i> is decremented. Therefore, <i>j</i> is assigned 6.



Increment and Decrement Operators (++ and --)

Examples:

```
int count=15;  
int a, b, c, d;  
a = count++;  
b = count;  
c = ++count;  
d = count;  
System.out.println(a + ", " + b + ", " + c + ", " + d);
```



Operator Precedence

Rules of precedence:

1. Operators within a pair of parentheses
2. Increment and decrement operators
3. Multiplication and division operators, evaluated from left to right
4. Addition and subtraction operators, evaluated from left to right

Example of need for rules of precedence (is the answer 34 or 9?):

```
c = 25 - 5 * 4 / 2 - 10 + 4;
```



Using Parentheses

Examples:

```
c = (((25 - 5) * 4) / (2 - 10)) + 4;
```

```
c = ((20 * 4) / (2 - 10)) + 4;
```

```
c = (80 / (2 - 10)) + 4;
```

```
c = (80 / -8) + 4;
```

```
c = -10 + 4;
```

```
c = -6;
```



Using Promotion and Type Casting

- Example of potential issue:

```
int num1 = 53; // 32 bits of memory to hold the value
int num2 = 47; // 32 bits of memory to hold the value
byte num3; // 8 bits of memory reserved
num3 = (num1 + num2); // causes compiler error
```

- Example of potential solution:

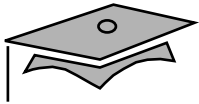
```
int num1 = 53;
int num2 = 47;
long num3;
num3 = (num1 + num2);
```




Promotion

- Automatic promotions:
 - If you assign a smaller type to a larger type
 - If you assign an integral type to a floating point type
- Examples of automatic promotions:

```
long big = 6;
```



Type Casting

- Syntax:

identifier = (target_type) value

- Example of potential issue:

```
int num1 = 53; // 32 bits of memory to hold the value
int num2 = 47; // 32 bits of memory to hold the value
byte num3; // 8 bits of memory reserved
num3 = (num1 + num2); // causes compiler error
```

- Example of potential solution:

```
int num1 = 53; // 32 bits of memory to hold the value
int num2 = 47; // 32 bits of memory to hold the value
byte num3; // 8 bits of memory reserved
num3 = (byte)(num1 + num2); // no data loss
```



Type Casting

Examples:

```
int myInt;  
long myLong = 99L;  
myInt = (int) (myLong); // No data loss, only zeroes.  
                        // A much larger number would  
                        // result in data loss.
```

```
int myInt;  
long myLong = 123987654321L;  
myInt = (int) (myLong); // Number is "chopped"
```



Compiler Assumptions for Integral and Floating Point Data Types

- Example of potential problem:

```
short a, b, c;  
a = 1 ;  
b = 2 ;  
c = a + b ;
```

- Example of potential solutions:
 - Declare c as an `int` type in the original declaration:

```
int c;
```

- Type cast the `(a+b)` result in the assignment line:

```
c = (short)(a+b);
```



Floating Point Data Types and Assignment

- Example of potential problem:

```
float float1 = 27.9;
```

- Example of potential solutions:
 - The F notifies the compiler that 27.9 is a float value:

```
float float1 = 27.9F;
```

- 27.9 is cast to a float type:

```
float float1 = (float) 27.9;
```



Example

```
1  public class Person {
2
3      public int ageYears = 32;
4
5      public void calculateAge() {
6
7          int ageDays = ageYears * 365;
8          long ageSeconds = ageYears * 365 * 24L * 60 * 60;
9
10         System.out.println("You are " + ageDays + " days old.");
11         System.out.println("You are " + ageSeconds + " seconds old.");
12
13     } // end of calculateAge method
14 } // end of class
```



Exercise 2: Using Operators and Type Casting

- Objective
- Tasks
- Solutions
- Discussion



Module 5

Creating and Using Objects

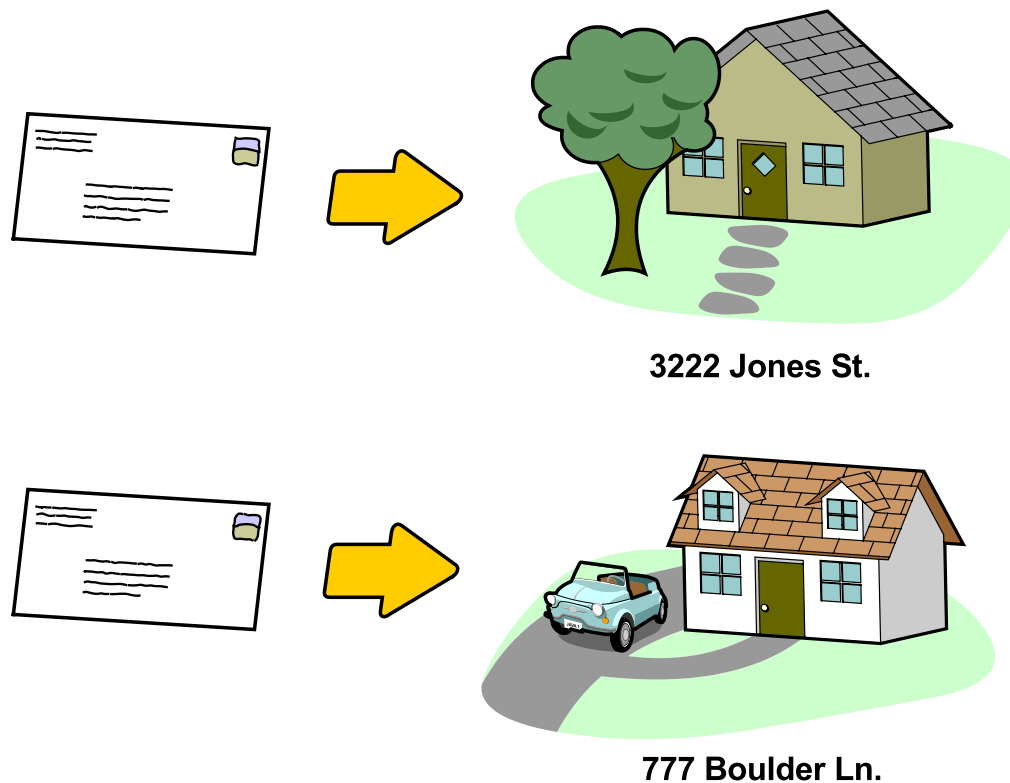


Overview

- Objectives:
 - Declare, instantiate, and initialize object reference variables
 - Compare how object reference variables are stored in relation to primitive variables
 - Use a class (the `String` class) included in the Java SDK
 - Use the Java 2 Platform, Standard Edition™ (J2SE™) application programming interface (API) documentation to learn about other classes in this API
- Relevance



Declaring Object References, Instantiating Objects, and Initializing Object References





Declaring Object References, Instantiating Objects, and Initializing Object References

- Example:

```
1  class ShirtTest {  
2  
3      public static void main (String args[]) {  
4  
5          Shirt myShirt = new Shirt();  
6  
7          myShirt.displayShirtInformation();  
8  
9      }  
10 }
```



Declaring Object Reference Variables

- Syntax:

Classname identifier;

- Example:

`Shirt myShirt;`



Instantiating an Object

Syntax:

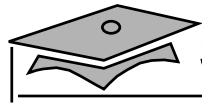
```
new Classname()
```



Initializing Object Reference Variables

- The assignment operator
- Example:

```
myShirt = new Shirt();
```



Using an Object Reference Variable to Manipulate Data

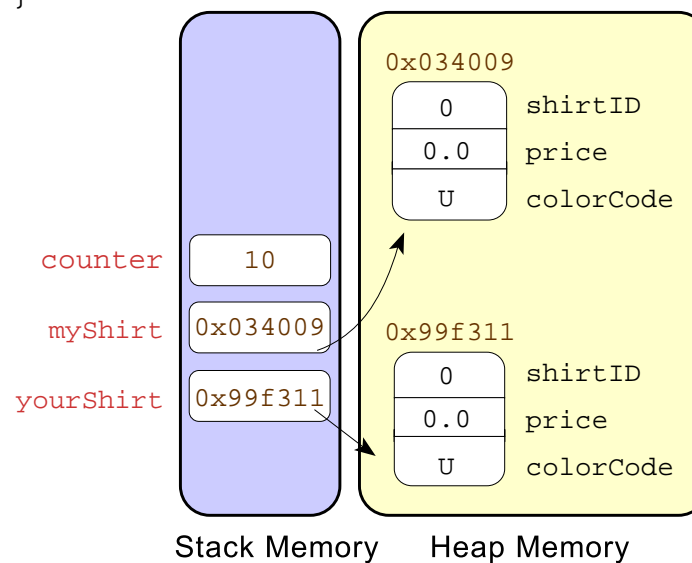
- Example:

```
1  public class ShirtTestTwo {
2
3      public static void main (String args[]) {
4
5          Shirt myShirt = new Shirt();
6          Shirt yourShirt = new Shirt();
7
8          myShirt.displayShirtInformation();
9          yourShirt.displayShirtInformation();
10
11         myShirt.colorCode='R' ;
12         yourShirt.colorCode='G' ;
13
14         myShirt.displayShirtInformation();
15         yourShirt.displayShirtInformation();
16
17     }
18 }
19
20
```



Storing Object Reference Variables in Memory

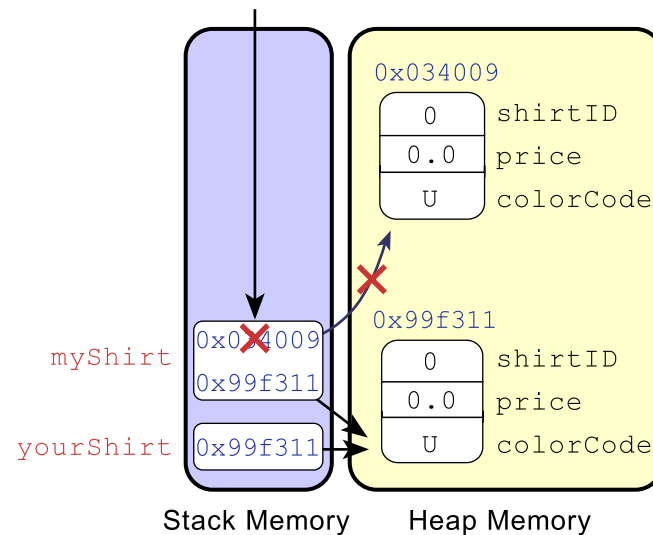
```
public static void main (String args[]) {  
  
    int counter;  
    counter = 10;  
    Shirt myShirt = new Shirt ( );  
}
```





Assigning an Object Reference From One Variable to Another

```
1  Shirt myShirt = new Shirt( );  
2  Shirt yourShirt = new Shirt( );  
3  myShirt = yourShirt;
```





Exercise 1: Using the ObjectTool to Create and Manipulate Objects

- Objective
- Tasks
- Solutions
- Discussion



Exercise 2: Creating a Test Class

- Objective
- Task
- Solution
- Discussion



Using the String Class

- Creating a String object with the new keyword:

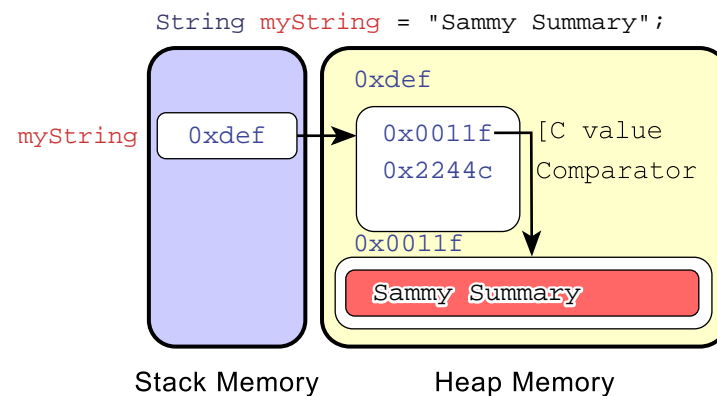
```
String myName = new String("Fred Smith");
```

- Creating a String object without the new keyword:

```
String myName = "Fred Smith";
```



Storing String Objects in Memory





Using Reference Variables for String Objects

- Example:

```
1 public class PersonTwo {  
2  
3     public String name = "Jonathan";  
4     public String job = "Ice Cream Taster";  
5  
6     public void display(){  
7         System.out.println("My name is " + name + ", I am a " + job);  
8     }  
9 } // end of class
```



Exercise 3: Using the `String` Class

- Objective
- Task
- Solution
- Discussion



Exercise 4: Examining `String` Objects With the `ObjectTool`

- Objective
- Task
- Solution
- Discussion



Investigating the Java Class Libraries

- Universal Resource Locator (URL) to view the J2SE specification:

`http://java.sun.com/j2se/version/docs/api/index.html`

- Example:

`http://java.sun.com/j2se/1.4/docs/api/index.html`



Investigating the Java Class Libraries

The screenshot shows the Java Class Library documentation for the `String` class. On the left is a navigation pane with a list of Java packages and classes. The `String` class is highlighted in red. The main content area displays the `String` class details, including its inheritance hierarchy, implemented interfaces, and a description.

Navigation Pane:

- [java.awt.print](#)
- [java.beans](#)
- [java.beans.beancontext](#)
- [java.io](#)
- [java.lang](#)**
- [java.lang.ref](#)
- [java.lang.reflect](#)
- [Float](#)
- [InheritableThreadLocal](#)
- [Integer](#)
- [Long](#)
- [Math](#)
- [Number](#)
- [Object](#)
- [Package](#)
- [Process](#)
- [Runtime](#)
- [RuntimePermission](#)
- [SecurityManager](#)
- [Short](#)
- [StackTraceElement](#)
- [StrictMath](#)
- [String](#)**

Class String

[java.lang.Object](#)

|-- **java.lang.String**

All Implemented Interfaces:

[CharSequence](#), [Comparable](#), [Serializable](#)

public final class **String**
extends [Object](#)
implements [Serializable](#), [Comparable](#), [CharSequence](#)

The string class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.



Using the Java Class Library Specification to Learn About a Method

- The `println` method:

```
System.out.println(data_to_print_to_the_screen);
```

- Example:

```
System.out.print("Carpe diem ");  
System.out.println("Seize the day");
```

prints this:

Carpe diem Seize the day



Exercise 5: Using the Class Library Specification

- Objective
- Task
- Solution
- Discussion



Module 6

Using Operators and Decision Constructs



Overview

- Objectives:
 - Identify relational and conditional operators
 - Examine `if` and `if/else` constructs
 - Use the `switch` constructs
- Relevance



Using Relational and Conditional Operators



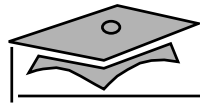


Elevator Example

```
1  public class Elevator {
2
3      public boolean doorOpen=false; // Doors are closed by
default
4      public int currentFloor = 1; // All elevators start on
first floor
5      public final int MAX_FLOORS = 10;
6      public final int MIN_FLOORS = 1;
7
8      public void openDoor() {
9          System.out.println("Opening door.");
10         doorOpen = true;
11         System.out.println("Door is open.");
12     }
13
14     public void closeDoor() {
15         System.out.println("Closing door.");
16         doorOpen = false;
17         System.out.println("Door is closed.");
```




```
18     }
19
20     public void goUp() {
21         System.out.println("Going up one floor.");
22         currentFloor++;
23         System.out.println("Floor: " + currentFloor);
24     }
25
26     public void goDown() {
27         System.out.println("Going down one floor.");
28         currentFloor--;
29         System.out.println("Floor: " + currentFloor);
30     }
31
32     public int getFloor() {
33         return currentFloor;
34     }
35
36     public boolean checkDoorStatus() {
37         return doorOpen;
38     }
39 }
```



Elevator Test Class

```
1  public class ElevatorTest {
2
3      public static void main(String args[]) {
4
5          Elevator myElevator = new Elevator();
6
7          myElevator.openDoor();
8          myElevator.closeDoor();
9          myElevator.goDown();
10         myElevator.goUp();
11         myElevator.goUp();
12         myElevator.goUp();
13         myElevator.openDoor();
14         myElevator.closeDoor();
15         myElevator.goDown();
16         myElevator.openDoor();
17         myElevator.closeDoor();
18         myElevator.goDown();
19         myElevator.openDoor();
20     }
21 }
```



Relational Operators

Condition	Operator	Example
Is equal to	<code>==</code>	<pre>int i==1; (i == 1)</pre>
Is not equal to	<code>!=</code>	<pre>int i=2; (i != 1)</pre>
Is less than	<code><</code>	<pre>int i=0; (i < 1)</pre>
Is less than or equal to	<code><=</code>	<pre>int i=1; (i <= 1)</pre>
Is greater than	<code>></code>	<pre>int i=2; (i > 1)</pre>
Is greater than or equal to	<code>>=</code>	<pre>int i=1; (i >= 1)</pre>



Testing Equality Between Strings

Example:

```
1  public class Employees {
2
3      public String name1 = "Fred Smith";
4      public String name2 = "Joseph Smith";
5
6      public void areNamesEqual() {
7
8          if (name1.equals(name2)) {
9              System.out.println("Same name.");
10         }
11         else {
12             System.out.println("Different name.");
13         }
14     }
15 }
```



Conditional Operators

Operation	Operator	Example
If one condition AND another condition	&&	<pre>int i = 2; int j = 8; ((i < 1) && (j > 6))</pre>
If either one condition OR another condition		<pre>int i = 2; int j = 8; ((i < 1) (j > 10))</pre>
NOT	!	<pre>int i = 2; (!(i < 3))</pre>



The `if` Construct

- Syntax:

```
if (boolean_expression) {  
    code_block;  
} // end of if construct  
// program continues here
```

- Example of potential output:

```
Opening door.  
Door is open.  
Closing door.  
Door is closed.  
Going down one floor.  
Floor: 0 <--- this is a error in logic  
Going up one floor.  
Floor: 1  
Going up one floor.  
Floor: 2  
...
```



The `if` Construct

Example of potential solution:

```
1  public class IfElevator {
2
3      public boolean doorOpen=false; // Doors are closed by default
4      public int currentFloor = 1; // All elevators start on first floor
5      public final int MAX_FLOORS = 10;
6      public final int MIN_FLOORS = 1;
7
8      public void openDoor() {
9          System.out.println("Opening door.");
10         doorOpen = true;
11         System.out.println("Door is open.");
12     }
13
14     public void closeDoor() {
15         System.out.println("Closing door.");
16         doorOpen = false;
17         System.out.println("Door is closed.");
18     }
19 }
```



```
18     }
19
20     public void goUp() {
21         System.out.println("Going up one floor.");
22         currentFloor++;
23         System.out.println("Floor: " + currentFloor);
24     }
25
26     public void goDown() {
27
28         if (currentFloor == MIN_FLOORS) {
29             System.out.println("Cannot Go down");
30         }
31
32         if (currentFloor > MIN_FLOORS) {
33             System.out.println("Going down one floor.");
34             currentFloor--;
35             System.out.println("Floor: " + currentFloor);
36         }
37     }
38
39     public int getFloor() {
```




```
40     return currentFloor;
41 }
42
43 public boolean checkDoorStatus() {
44     return doorOpen;
45 }
46 }
```



The `if` Construct

Example potential output:

```
Opening door.  
Door is open.  
Closing door.  
Door is closed.  
Cannot Go down <--- elevator logic prevents problem  
Going up one floor.  
Floor: 2  
Going up one floor.  
Floor: 3  
...
```



Nested `if` Statements

Example:

```
1  public class NestedIfElevator {
2
3      public boolean doorOpen=false; // Doors are closed by default
4      public int currentFloor = 1; // All elevators start on first floor
5      public final int MAX_FLOORS = 10;
6      public final int MIN_FLOORS = 1;
7
8      public void openDoor() {
9          System.out.println("Opening door.");
10         doorOpen = true;
11         System.out.println("Door is open.");
12     }
13
14     public void closeDoor() {
15         System.out.println("Closing door.");
16         doorOpen = false;
17         System.out.println("Door is closed.");
18     }
19 }
```



```
18     }
19
20     public void goUp() {
21         System.out.println("Going up one floor.");
22         currentFloor++;
23         System.out.println("Floor: " + currentFloor);
24     }
25
26     public void goDown() {
27
28         if (currentFloor == MIN_FLOORS) {
29             System.out.println("Cannot Go down");
30         }
31
32         if (currentFloor > MIN_FLOORS) {
33
34             if (!doorOpen) {
35
36                 System.out.println("Going down one floor.");
37                 currentFloor--;
38                 System.out.println("Floor: " + currentFloor);
39             }
```



```
40     }
41 }
42
43 public int getFloor() {
44     return currentFloor;
45 }
46
47 public boolean checkDoorStatus() {
48     return doorOpen;
49 }
50 }
```



The if/else Construct

Syntax:

```
if (boolean_expression) {  
    code_block;  
}  
// end of if construct  
  
else {  
    code_block;  
}  
// end of else construct  
  
// program continues here
```



The if/else Construct

Example:

```
1  public class IfElseElevator {
2
3      public boolean doorOpen=false; // Doors are closed by default
4      public int currentFloor = 1; // All elevators start on first floor
5      public final int MAX_FLOORS = 10;
6      public final int MIN_FLOORS = 1;
7
8      public void openDoor() {
9          System.out.println("Opening door.");
10         doorOpen = true;
11         System.out.println("Door is open.");
12     }
13
14     public void closeDoor() {
15         System.out.println("Closing door.");
16         doorOpen = false;
17         System.out.println("Door is closed.");
```



```
18     }
19
20     public void goUp() {
21         System.out.println("Going up one floor.");
22         currentFloor++;
23         System.out.println("Floor: " + currentFloor);
24     }
25
26     public void goDown() {
27
28         if (currentFloor == MIN_FLOORS) {
29             System.out.println("Cannot Go down");
30         }
31
32         else {
33             System.out.println("Going down one floor.");
34             currentFloor--;
35             System.out.println("Floor: " + currentFloor);
36         }
37     }
38
39     public int getFloor() {
```




```
40     return currentFloor;
41 }
42
43 public boolean checkDoorStatus() {
44     return doorOpen;
45 }
46 }
```



The `if/else` Construct

Example potential output:

```
Opening door.  
Door is open.  
Closing door.  
Door is closed.  
Cannot Go down <--- elevator logic prevents problem  
Going up one floor.  
Floor: 2  
Going up one floor.  
Floor: 3  
...
```



Chaining `if/else` Constructs

Syntax:

```
if (boolean_expression) {  
    code_block;  
}  
// end of if construct  
  
else if (boolean_expression) {  
    code_block;  
}  
// end of else if construct  
  
else {  
    code_block;  
}  
// program continues here
```



Chaining `if/else` Constructs

Example:

```
1  public class IfElseDate {
2
3      public int month = 10;
4
5      public void calculateNumDays() {
6
7          if (month == 1 || month == 3 || month == 5 || month == 7 ||
8              month == 8 || month == 10 || month == 12) {
9
10             System.out.println("There are 31 days in that month.");
11         }
12
13         else if (month == 2) {
14             System.out.println("There are 28 days in that month.");
15         }
16
17         else if (month == 4 || month == 6 || month == 9 || month == 11) {
```



```
18         System.out.println("There are 30 days in that month.");
19     }
20
21     else {
22         System.out.println("Invalid month.");
23     }
24 }
25 }
```



Exercise 1: Using `if` and `if/else` Constructs

- Objective
- Tasks
- Solutions
- Discussion



Using the `switch` Construct

Syntax:

```
switch (variable) {  
    case literal_value:  
        code_block;  
        [break;]  
    case another_literal_value:  
        code_block;  
        [break;]  
    [default:]  
        code_block;  
}
```



Using the switch Construct

Example:

```
1  public class SwitchDate {
2
3      public int month = 10;
4
5      public void calculateNumDays() {
6
7          switch(month) {
8              case 1:
9              case 3:
10             case 5:
11             case 7:
12             case 8:
13             case 10:
14             case 12:
15                 System.out.println("There are 31 days in that month.");
16                 break;
17             case 2:
```




```
18         System.out.println("There are 28 days in that month.");
19         break;
20     case 4:
21     case 6:
22     case 9:
23     case 11:
24         System.out.println("There are 30 days in that month.");
25         break;
26     default:
27         System.out.println("Invalid month.");
28         break;
29     }
30 }
31 }
```



When to Use `switch` Constructs

- Equality tests
- Tests against a *single* variable, such as `customerStatus`
- Tests against the value of an `int`, `short`, `byte`, or `char` type



Exercise 2: Using the `switch` Construct

- Objective
- Task
- Solution
- Discussion



Module 7

Using Loop Constructs



Overview

- Objectives:
 - Create `while` loops
 - Develop `for` loops
 - Create `do/while` loops
- Relevance



Creating `while` Loops

- Syntax:

```
while (boolean_expression) {  
  
    code_block;  
  
}    // end of while construct  
  
// program continues here
```



Creating while Loops

Example:

```
1  public class WhileElevator {
2
3      public boolean doorOpen=false;
4      public int currentFloor = 1;
5      public int weight = 0;
6
7      public final int CAPACITY = 1000;
8      public final int TOP_FLOOR = 5;
9      public final int BOTTOM_FLOOR = 1;
10
11     public void openDoor() {
12         System.out.println("Opening door.");
13         doorOpen = true;
14         System.out.println("Door is open.");
15     }
16
17     public void closeDoor() {
```



```
18     System.out.println("Closing door.");
19     doorOpen = false;
20     System.out.println("Door is closed.");
21 }
22
23 public void goUp() {
24     System.out.println("Going up one floor.");
25     currentFloor++;
26     System.out.println("Floor: " + currentFloor);
27 }
28
29 public void goDown() {
30     System.out.println("Going down one floor.");
31     currentFloor--;
32     System.out.println("Floor: " + currentFloor);
33 }
34
35 public void setFloor() {
36
37     // Normally you would pass the desiredFloor as an argument to the
38     // setFloor method. However, because you have not learned how to
39     // do this yet, desiredFloor is set to a specific number (5)
```




```
40      // below.
41
42      int desiredFloor = 5;
43
44      while (currentFloor != desiredFloor)
45          if (currentFloor < desiredFloor) {
46              goUp();
47          }
48          else {
49              goDown();
50          }
51      }
52
53      public int getFloor() {
54          return currentFloor;
55      }
56
57      public boolean checkDoorStatus() {
58          return doorOpen;
59      }
60  }
```



Nested while Loops

- Problem:

```
@@@@@@@@@@@
@@@@@@@@@@@
@@@@@@@@@@@
1  public class WhileElevator {
2
3      public boolean doorOpen=false;
4      public int currentFloor = 1;
5      public int weight = 0;
6
7      public final int CAPACITY = 1000;
8      public final int TOP_FLOOR = 5;
9      public final int BOTTOM_FLOOR = 1;
10
11     public void openDoor() {
12         System.out.println("Opening door.");
13         doorOpen = true;
14         System.out.println("Door is open.");
15     }
```



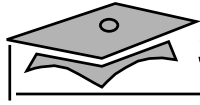
```
16
17     public void closeDoor() {
18         System.out.println("Closing door.");
19         doorOpen = false;
20         System.out.println("Door is closed.");
21     }
22
23     public void goUp() {
24         System.out.println("Going up one floor.");
25         currentFloor++;
26         System.out.println("Floor: " + currentFloor);
27     }
28
29     public void goDown() {
30         System.out.println("Going down one floor.");
31         currentFloor--;
32         System.out.println("Floor: " + currentFloor);
33     }
34
35     public void setFloor() {
36
37         // Normally you would pass the desiredFloor as an argument to the
```



```
38      // setFloor method. However, because you have not learned how to
39      // do this yet, desiredFloor is set to a specific number (5)
40      // below.
41
42      int desiredFloor = 5;
43
44      while (currentFloor != desiredFloor)
45          if (currentFloor < desiredFloor) {
46              goUp();
47          }
48          else {
49              goDown();
50          }
51      }
52
53      public int getFloor() {
54          return currentFloor;
55      }
56
57      public boolean checkDoorStatus() {
58          return doorOpen;
59      }
```



60 }



Nested while Loops

- Example potential solution:

```
1  public class WhileRectangle {
2
3      public int height = 3;
4      public int width = 10;
5
6      public void displayRectangle() {
7
8          int colCount = 0;
9          int rowCount = 0;
10
11         while (rowCount < height) {
12             colCount=0;
13
14             while (colCount < width) {
15                 System.out.print("@");
16                 colCount++;
17             }
18
19             System.out.println();
20             rowCount++;
21         }
22     }
23 }
```



Exercise 1: Using the `while` Loop

- Objective
- Tasks
- Solutions
- Discussion



Developing a for Loop

Syntax:

```
for (initialize[,initialize]; boolean_expression;  
    update[,update]) {  
  
    code_block;  
}
```




Developing a for Loop

Example:

```
1  public class ForElevator {
2
3      public boolean doorOpen=false;
4      public int currentFloor = 1;
5      public int weight = 0;
6
7      public final int CAPACITY = 1000;
8      public final int TOP_FLOOR = 5;
9      public final int BOTTOM_FLOOR = 1;
10
11     public void openDoor() {
12         System.out.println("Opening door.");
13         doorOpen = true;
14         System.out.println("Door is open.");
15     }
16
17     public void closeDoor() {
```



```
18     System.out.println("Closing door.");
19     doorOpen = false;
20     System.out.println("Door is closed.");
21 }
22
23 public void goUp() {
24     System.out.println("Going up one floor.");
25     currentFloor++;
26     System.out.println("Floor: " + currentFloor);
27 }
28
29 public void goDown() {
30     System.out.println("Going down one floor.");
31     currentFloor--;
32     System.out.println("Floor: " + currentFloor);
33 }
34
35 public void setFloor() {
36
37     // Normally you would pass the desiredFloor as an argument to the
38     // setFloor method. However, because you have not learned how to
39     // do this yet, desiredFloor is set to a specific number (5)
```



```
40      // below.
41
42      int desiredFloor = 5;
43
44      if (currentFloor > desiredFloor) {
45          for (int down = currentFloor; down != desiredFloor; --down) {
46              goDown();
47          }
48      }
49
50      else {
51          for (int up = currentFloor; up != desiredFloor; ++up) {
52              goUp();
53          }
54      }
55  }
56
57  public int getFloor() {
58      return currentFloor;
59  }
60
61  public boolean checkDoorStatus() {
```



```
62      return doorOpen;  
63  }  
64 }
```



Nested for Loops

Example:

```
1  public class ForRectangle {
2
3      public int height = 3;
4      public int width = 10;
5
6      public void displayRectangle() {
7
8          for (int rowCount = 0; rowCount < height; rowCount++) {
9              for (int colCount = 0; colCount < width; colCount++) {
10                 System.out.print("@");
11             }
12             System.out.println();
13         }
14     }
15 }
```



Exercise 2: Using the `for` Loop

- Objective
- Tasks
- Solutions
- Discussion



Coding a do/while Loop

Syntax:

```
do {  
    code_block;  
}  
while (boolean_expression); // Semicolon is mandatory.
```



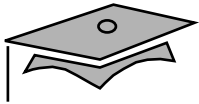
Coding a do/while Loop

Example:

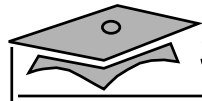
```
1  public class DoWhileElevator {
2
3      public boolean doorOpen=false;
4      public int currentFloor = 1;
5      public int weight = 0;
6
7      public final int CAPACITY = 1000;
8      public final int TOP_FLOOR = 5;
9      public final int BOTTOM_FLOOR = 1;
10
11     public void openDoor() {
12         System.out.println("Opening door.");
13         doorOpen = true;
14         System.out.println("Door is open.");
15     }
16
17     public void closeDoor() {
```




```
18     System.out.println("Closing door.");
19     doorOpen = false;
20     System.out.println("Door is closed.");
21 }
22
23 public void goUp() {
24     System.out.println("Going up one floor.");
25     currentFloor++;
26     System.out.println("Floor: " + currentFloor);
27 }
28
29 public void goDown() {
30     System.out.println("Going down one floor.");
31     currentFloor--;
32     System.out.println("Floor: " + currentFloor);
33 }
34
35 public void setFloor() {
36
37     // Normally you would pass the desiredFloor as an argument to the
38     // setFloor method. However, because you have not learned how to
39     // do this yet, desiredFloor is set to a specific number (5)
40     // below.
```



```
41
42     int desiredFloor = 5;
43
44     do {
45         if (currentFloor < desiredFloor) {
46             goUp();
47         }
48         if (currentFloor > desiredFloor) {
49             goDown();
50         }
51     }
52     while (currentFloor != desiredFloor);
53 }
54
55 public int getFloor() {
56     return currentFloor;
57 }
58
59 public boolean checkDoorStatus() {
60     return doorOpen;
61 }
62 }
63
```



Nested do/while Loops

Example:

```
1  public class DoWhileRectangle {
2
3      public int height = 3;
4      public int width = 10;
5
6      public void displayRectangle() {
7
8          int rowCount = 0;
9          int colCount = 0;
10
11         do {
12             colCount = 0;
13
14             do {
15                 System.out.print("@");
16                 colCount++;
17             }
18             while (colCount < width);
19
20             System.out.println();
21             rowCount++;
22         }
23         while (rowCount < height);
24     }
25 }
```



Comparing Loop Constructs

- Use the `while` loop to iterate indefinitely through statements and to perform the statements zero or more times.
- Use the `do/while` loop to iterate indefinitely through statements and to perform the statements *one* or more times.
- Use the `for` loop to step through statements a predefined number of times.



Exercise 3: Using the `do/while` Loop

- Objective
- Tasks
- Solutions
- Discussion



Module 8

Developing and Using Methods



Overview

- Objectives:
 - Describe the advantages of methods and define worker and calling methods
 - Declare and invoke a method
 - Compare object and static methods
 - Use overloaded methods
- Relevance



Creating and Invoking Methods

- Syntax:

```
[modifiers] return_type method_identifier ([arguments]) {  
    method_code_block  
}
```




Basic Form of a Method

Example:

```
public void displayShirtInformation() {  
    System.out.println("Shirt ID: " + shirtID);  
    System.out.println("Shirt description:" + description);  
    System.out.println("Color Code: " + colorCode);  
    System.out.println("Shirt price: " + price);  
    System.out.println("Quantity in stock: " + quantityInStock);  
} // end of display method
```



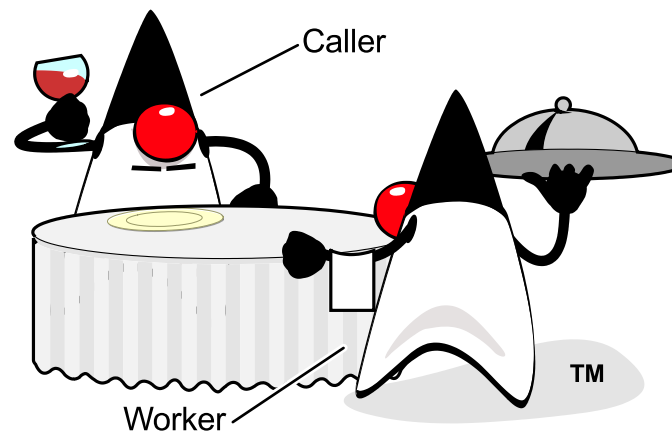
Invoking a Method From a Different Class

Example:

```
1  public class ShirtTest {  
2  
3      public static void main (String args[]) {  
4  
5          Shirt myShirt;  
6          myShirt = new Shirt();  
7  
8          myShirt.displayShirtInformation();  
9  
10     }  
11 }
```



Calling and Worker Methods





Invoking a Method in the Same Class

- Example

```
1  public class Elevator {
2
3      public boolean doorOpen=false;
4      public int currentFloor = 1;
5      public int weight = 0;
6
7      public final int CAPACITY = 1000;
8      public final int TOP_FLOOR = 5;
9      public final int BOTTOM_FLOOR = 1;
10
11     public void openDoor() {
12         System.out.println("Opening door.");
13         doorOpen = true;
14         System.out.println("Door is open.");
15     }
16
17     public void closeDoor() {
18         System.out.println("Closing door.");
```



```
19     doorOpen = false;
20     System.out.println("Door is closed.");
21 }
22
23 public void goUp() {
24     System.out.println("Going up one floor.");
25     currentFloor++;
26     System.out.println("Floor: " + currentFloor);
27 }
28
29 public void goDown() {
30     System.out.println("Going down one floor.");
31     currentFloor--;
32     System.out.println("Floor: " + currentFloor);
33 }
34
35 public void setFloor(int desiredFloor) {
36     while (currentFloor != desiredFloor)
37         if (currentFloor < desiredFloor) {
38             goUp();
39         }
40     else {
```



```
41         goDown( );
42     }
43 }
44
45 public int getFloor() {
46     return currentFloor;
47 }
48
49 public boolean checkDoorStatus() {
50     return doorOpen;
51 }
52 }
```

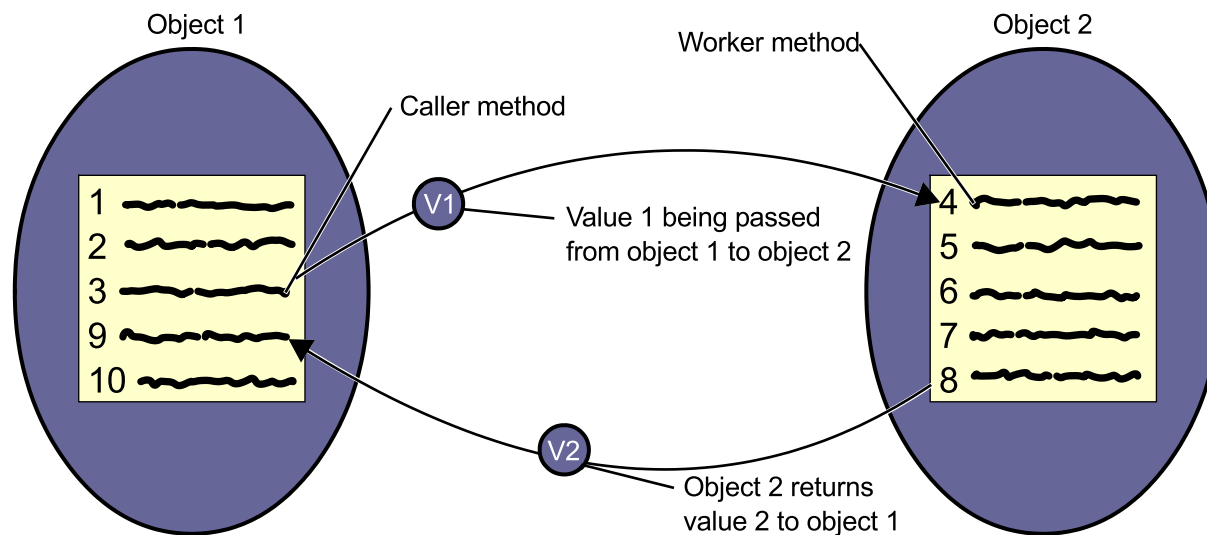


Guidelines for Invoking Methods

- There is no limit to the number of method calls that a calling method can make.
- The calling method and the worker method can be in the same class or in different classes.
- The way you invoke the worker method is different, depending on whether it is in the same class or in a different class from the calling method.
- You can invoke methods in any order. Methods do not need to be completed in the order in which they are listed in the class where they are declared (the class containing the worker methods).



Passing Arguments and Returning Values





Declaring Methods With Arguments

- Example:

```
public void setFloor(int desiredFloor) {  
    while (currentFloor != desiredFloor) {  
        if (currentFloor < desiredFloor) {  
            goUp();  
        }  
        else {  
            goDown();  
        }  
    }  
}
```

- Example:

```
public void multiply(int NumberOne, int NumberTwo)
```



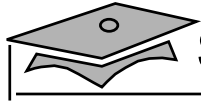
The main Method

- Example:

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

- Example (invocation):

```
java ShirtTest 12.99 R
```



Invoking Methods With Arguments

Example:

```
1  public class ElevatorTest {
2
3      public static void main(String args[]) {
4
5          Elevator myElevator = new Elevator();
6
7          myElevator.openDoor();
8          myElevator.closeDoor();
9          myElevator.goUp();
10         myElevator.goUp();
11         myElevator.goUp();
12         myElevator.openDoor();
13         myElevator.closeDoor();
14         myElevator.goDown();
15         myElevator.openDoor();
16         myElevator.closeDoor();
17         myElevator.goDown();
18
19         myElevator.setFloor(myElevator.TOP_FLOOR);
20
21         myElevator.openDoor();
22     }
23 }
```



Declaring Methods With Return Values

Declaration:

```
public int sum(int numberOne, int numberTwo)
```



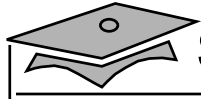
Returning a Value

Example:

```
public int sum(int numberOne, int numberTwo) {  
    int sum = numberOne + numberTwo;  
  
    return sum;  
}
```

Example:

```
public int getFloor() {  
    return currentFloor;  
}
```



Receiving Return Values

Example:

```
1  public class ElevatorTestTwo {
2
3      public static void main(String args[]) {
4
5          Elevator myElevator = new Elevator();
6
7          myElevator.openDoor();
8          myElevator.closeDoor();
9          myElevator.goUp();
10         myElevator.goUp();
11         myElevator.goUp();
12         myElevator.openDoor();
13         myElevator.closeDoor();
14         myElevator.goDown();
15         myElevator.openDoor();
16         myElevator.closeDoor();
17         myElevator.goDown();
18
19         int curFloor = myElevator.getFloor();
20         System.out.println("Current Floor: " +
curFloor);
21
22         myElevator.setFloor(curFloor+1);
23
24         myElevator.openDoor();
25     }
26 }
```



Advantages of Method Use

- Methods make programs more readable and easier to maintain.
- Methods make development and maintenance quicker.
- Methods are central to reusable software.
- Methods allow separate objects to communicate and to distribute the work performed by the program.



Exercise 1: Using Arguments and Return Values

- Objective
- Task
- Solution
- Discussion



Creating `static` Methods and Variables

- Comparing instance and `static` methods and variables
- Declaring `static` methods:

```
static Properties getProperties()
```

- Invoking `static` methods:

```
Classname.method();
```



Creating `static` Methods and Variables

Example:

```
public static char convertShirtSize(int numericalSize) {  
  
    if (numericalSize < 10) {  
        return 'S';  
    }  
  
    else if (numericalSize < 14) {  
        return 'M';  
    }  
  
    else if (numericalSize < 18) {  
        return 'L';  
    }  
  
    else {  
        return 'X';  
    }  
}
```



Creating `static` Methods and Variables

- Declaring `static` variables:

```
static double salesTAX = 8.25;
```

- Accessing `static` variables:

```
Classname.variable;
```

- Example:

```
double myPI;  
myPI = Math.PI;
```



Static Methods and Variables in the Java API

- Examples:
 - The `Math` class
 - The `System` class



Static Methods and Variables in the Java API

- When to declare a `static` method or variable:
 - Performing the operation on an individual object or associating the variable with a specific object type is not important.
 - Accessing the variable or method before instantiating an object is important.
 - The method or variable does not logically belong to an object, but possibly belongs to a utility class, such as the `Math` class, included in the Java API.



Using Method Overloading

Example overloaded methods:

```
1  public class Calculator {
2
3      public int sum(int numberOne, int numberTwo){
4
5          System.out.println("Method One");
6
7          return numberOne + numberTwo;
8      }
9
10     public float sum(float numberOne, float numberTwo) {
11
12         System.out.println("Method Two");
13
14         return numberOne + numberTwo;
15     }
16
17     public float sum(int numberOne, float numberTwo) {
```



```
18
19     System.out.println( "Method Three" );
20
21     return numberOne + numberTwo;
22 }
23 }
```



Using Method Overloading

Example method invocation:

```
1  public class CalculatorTest {  
2  
3      public static void main(String [] args) {  
4  
5          Calculator myCalculator = new Calculator();  
6  
7          int totalOne = myCalculator.sum(2,3);  
8          System.out.println(totalOne);  
9  
10         float totalTwo = myCalculator.sum(15.99F, 12.85F);  
11         System.out.println(totalTwo);  
12  
13         float totalThree = myCalculator.sum(2, 12.85F);  
14         System.out.println(totalThree);  
15     }  
16 }  
17
```




Method Overloading and the Java API

Method	Use
<code>void println()</code>	Terminates the current line by writing the line separator string
<code>void println(boolean x)</code>	Prints a boolean value and then terminates the line
<code>void println(char x)</code>	Prints a character and then terminates the line
<code>void println(char[] x)</code>	Prints an array of characters and then terminates the line
<code>void println(double x)</code>	Prints a double and then terminates the line
<code>void println(float x)</code>	Prints a float and then terminates the line
<code>void println(int x)</code>	Prints an int and then terminates the line
<code>void println(long x)</code>	Prints a long and then terminates the line
<code>void println(Object x)</code>	Prints an object and then terminates the line
<code>void println(String x)</code>	Prints a string and then terminates the line



Uses for Method Overloading

Examples:

```
public int sum(int numberOne, int numberTwo)
public int sum(int numberOne, int numberTwo, int numberThree)
public int sum(int numberOne, int numberTwo, int numberThree, int
numberFour)
```



Uses for Method Overloading

Example:

```
1  public class ShirtTwo {
2
3      public int shirtID = 0; // Default ID for the shirt
4      public String description = "-description required-"; // default
5
6      // The color codes are R=Red, B=Blue, G=Green, U=Unset
7      public char colorCode = 'U';
8      public double price = 0.0; // Default price for all items
9      public int quantityInStock = 0; // Default quantity for all items
10
11     public void setShirtInfo(int ID, String desc, double cost){
12         shirtID = ID;
13         description = desc;
14         price = cost;
15     }
16
17     public void setShirtInfo(int ID, String desc, double cost, char
```



```
color){
18     shirtID = ID;
19     description = desc;
20     price = cost;
21     colorCode = color;
22 }
23
24     public void setShirtInfo(int ID, String desc, double cost, char color,
int quantity){
25         shirtID = ID;
26         description = desc;
27         price = cost;
28         colorCode = color;
29         quantityInStock = quantity;
30     }
31
32     // This method displays the values for an item
33     public void display() {
34
35         System.out.println("Item ID: " + shirtID);
36         System.out.println("Item description:" + description);
37         System.out.println("Color Code: " + colorCode);
```



```
38      System.out.println("Item price: " + price);
39      System.out.println("Quantity in stock: " + quantityInStock);
40
41  } // end of display method
42 } // end of class
```



Uses for Method Overloading

Example:

```
1  class ShirtTwoTest {
2
3      public static void main (String args[]) {
4          ShirtTwo shirtOne = new ShirtTwo();
5          ShirtTwo shirtTwo = new ShirtTwo();
6          ShirtTwo shirtThree = new ShirtTwo();
7
8          shirtOne.setShirtInfo(100, "Button Down", 12.99);
9          shirtTwo.setShirtInfo(101, "Long Sleeve Oxford", 27.99, 'G');
10         shirtThree.setShirtInfo(102, "Shirt Sleeve T-Shirt", 9.99, 'B', 50);
11
12         shirtOne.display();
13         shirtTwo.display();
14         shirtThree.display();
15     }
16 }
```



Exercise 2: Using Overloaded Methods

- Objective
- Task
- Solution
- Discussion



Module 9

Implementing Encapsulation and Constructors



Overview

- Objectives:
 - Use encapsulation to protect data
 - Create constructors to initialize objects
- Relevance

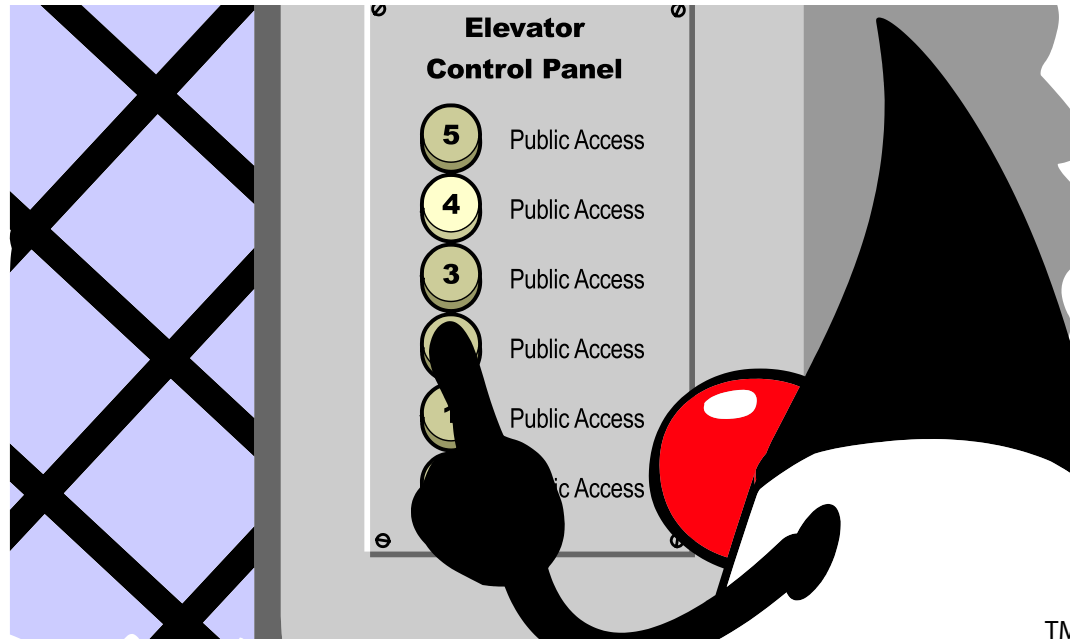


Using Encapsulation





The public Modifier



TM

```
public int currentFloor=1;

public void setFloor(int desiredFloor) {
    ...
}
```



The `public` Modifier

Example:

```
1  public class PublicElevator {  
2  
3      public boolean doorOpen=false;  
4      public int currentFloor = 1;  
5  
6      public final int TOP_FLOOR = 5;  
7      public final int BOTTOM_FLOOR = 1;  
8  }
```



The public Modifier

Example:

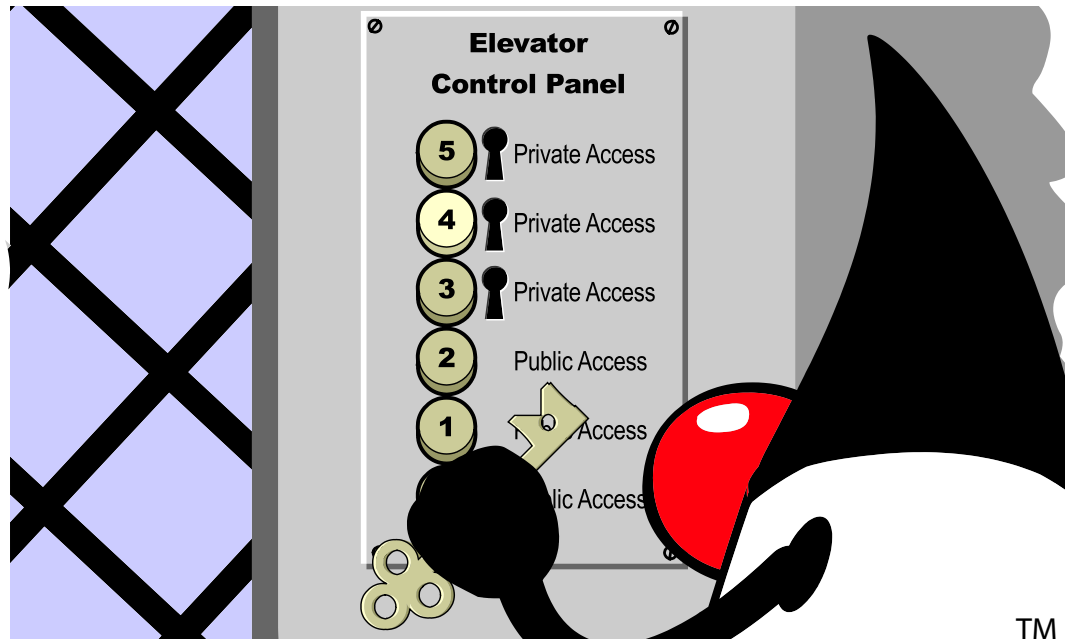
```
1  public class PublicElevatorTest {
2
3      public static void main(String args[]) {
4
5          PublicElevator pubElevator = new PublicElevator();
6
7          pubElevator.doorOpen = true;  //passengers get on
8          pubElevator.doorOpen = false; //doors close
9          //go down to floor 0 (below bottom of building)
10         pubElevator.currentFloor--;
11         pubElevator.currentFloor++;
12
13         //jump to floor 7 (only 5 floors in building)
14         pubElevator.currentFloor = 7;
15         pubElevator.doorOpen = true;  //passengers get on/off
16         pubElevator.doorOpen = false;
17         pubElevator.currentFloor = 1; //go to the first floor
```



```
18     pubElevator.doorOpen = true;    //passengers get on/off
19     pubElevator.currentFloor++;    //elevator moves with door open
20     pubElevator.doorOpen = false;
21     pubElevator.currentFloor--;
22     pubElevator.currentFloor--;
23 }
24 }
```



The private Modifier



```
private int currentFloor=1;

private void calculateCapacity() {
    ...
}
```



The private Modifier

Example:

```
1  public class PrivateElevator1 {  
2  
3      private boolean doorOpen=false;  
4      private int currentFloor = 1;  
5  
6      private final int TOP_FLOOR = 5;  
7      private final int BOTTOM_FLOOR = 1;  
8  }
```




The private Modifier

Example:

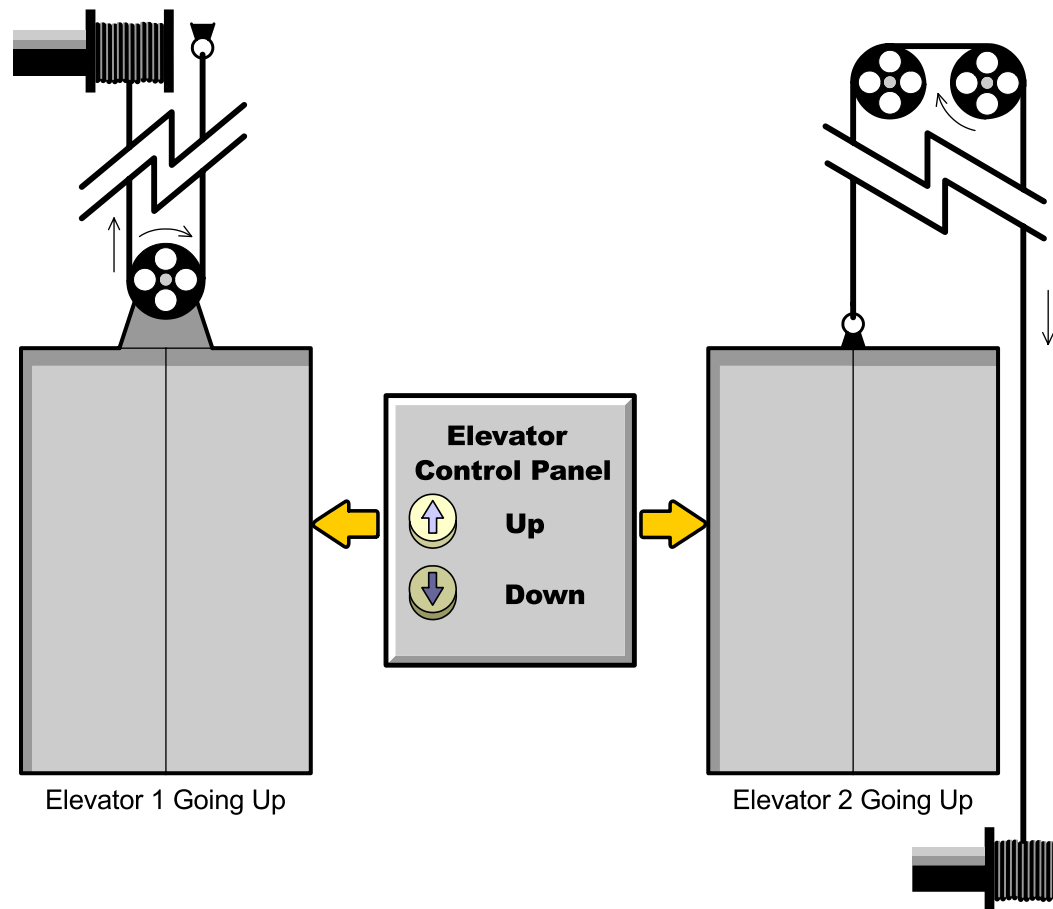
```
1  public class PrivateElevator1Test {
2
3      public static void main(String args[]) {
4
5          PrivateElevator1 privElevator = new PrivateElevator1();
6
7          /*****
8           * The following lines of code will not compile      *
9           * because they attempt to access private            *
10          * variables.                                          *
11          *****/
12
13          privElevator.doorOpen = true; //passengers get on
14          privElevator.doorOpen = false; //doors close
15          //go down to currentFloor 0 (below bottom of building)
16          privElevator.currentFloor--;
17          privElevator.currentFloor++;
```



```
18
19     //jump to currentFloor 7 (only 5 floors in building)
20     privElevator.currentFloor = 7;
21     privElevator.doorOpen = true;  //passengers get on/off
22     privElevator.doorOpen = false;
23     privElevator.currentFloor = 1; //go to the first floor
24     privElevator.doorOpen = true;  //passengers get on/off
25     privElevator.currentFloor++;    //elevator moves with door open
26     privElevator.doorOpen = false;
27     privElevator.currentFloor--;
28     privElevator.currentFloor--;
29 }
30 }
```



Interface and Implementation





Interface and Implementation

Example:

```
1  public class PrivateShirt1 {
2
3      private int shirtID = 0; // Default ID for the shirt
4      private String description = "-description required-"; // default
5
6      // The color codes are R=Red, B=Blue, G=Green, U=Unset
7      private char colorCode = 'U';
8      private double price = 0.0; // Default price for all items
9      private int quantityInStock = 0; // Default quantity for all items
10
11     public char getColorCode() {
12         return colorCode;
13     }
14
15     public void setColorCode(char newCode) {
16         colorCode = newCode;
17     }
```



```
18
19    // Additional get and set methods for shirtID, description,
20    // price, and quantityInStock would follow
21
22 } // end of class
```



Interface and Implementation

Example:

```
1  public class PrivateShirt1Test {
2
3      public static void main (String args[]) {
4
5          PrivateShirt1 privShirt = new PrivateShirt1();
6          char colorCode;
7
8          // Set a valid colorCode
9          privShirt.setColorCode('R');
10         colorCode = privShirt.getColorCode();
11
12         // The PrivateShirtTest1 class can set a valid colorCode
13         System.out.println("Color Code: " + colorCode);
14
15         // Set an invalid color code
16         privShirt.setColorCode('Z');
17         colorCode = privShirt.getColorCode();
```



```
18
19    // The PrivateShirtTest1 class can set an invalid colorCode
20    System.out.println("Color Code: " + colorCode);
21    }
22 }
```



Interface and Implementation

Example:

```
1  public class PrivateShirt2 {
2
3      private int shirtID = 0; // Default ID for the shirt
4      private String description = "-description required-"; // default
5
6      // The color codes are R=Red, B=Blue, G=Green, U=Unset
7      private char colorCode = 'U';
8      private double price = 0.0; // Default price for all items
9      private int quantityInStock = 0; // Default quantity for all items
10
11     public char getColorCode() {
12         return colorCode;
13     }
14
15     public void setColorCode(char newCode) {
16
17         switch (newCode) {
```




```
18     case 'R':
19     case 'G':
20     case 'B':
21         colorCode = newCode;
22         break;
23     default:
24         System.out.println("Invalid colorCode. Use R, G, or B");
25     }
26 }
27
28 // Additional get and set methods for shirtID, description,
29 // price, and quantityInStock would follow
30
31 } // end of class
```



Interface and Implementation

Example:

```
1  public class PrivateShirt2Test {
2
3      public static void main (String args[]) {
4
5          PrivateShirt2 privShirt = new PrivateShirt2();
6          char colorCode;
7
8          // Set a valid colorCode
9          privShirt.setColorCode('R');
10         colorCode = privShirt.getColorCode();
11
12         // The PrivateShirtTest2 class can set a valid colorCode
13         System.out.println("Color Code: " + colorCode);
14
15         // Set an invalid color code
16         privShirt.setColorCode('Z');
17         colorCode = privShirt.getColorCode();
```



```
18
19    // The PrivateShirtTest2 class cannot set an invalid colorCode.
20    // Color code is still R
21    System.out.println("Color Code: " + colorCode);
22    }
23 }
```



Encapsulated Elevator

Example:

```
1  public class PrivateElevator2 {  
2  
3      private boolean doorOpen=false;  
4      private int currentFloor = 1;  
5      private int weight = 0;  
6  
7      final int CAPACITY = 1000;  
8      final int TOP_FLOOR = 5;  
9      final int BOTTOM_FLOOR = 1;  
10  
11     public void openDoor() {  
12         doorOpen = true;  
13     }  
14  
15     public void closeDoor() {  
16         calculateCapacity();  
17     }
```



```
18     if (weight <= CAPACITY) {
19         doorOpen = false;
20     }
21     else {
22         System.out.println("The elevator has exceeded capacity.");
23         System.out.println("Doors will remain open until someone exits!");
24     }
25 }
26
27 // In reality, the elevator would have weight sensors to
28 // check the actual weight in the elevator, but for the sake
29 // of simplicity we just pick a random number to represent the
30 // weight in the elevator
31
32 private void calculateCapacity() {
33     weight = (int) (Math.random() * 1500);
34     System.out.println("The weight is " + weight);
35 }
36
37 public void goUp() {
38     if (!doorOpen) {
39         if (currentFloor < TOP_FLOOR) {
```



```
40     currentFloor++;
41     System.out.println(currentFloor);
42     }
43     else {
44     System.out.println("Already on top floor.");
45     }
46     }
47     else {
48     System.out.println("Doors still open!");
49     }
50     }
51
52     public void goDown() {
53     if (!doorOpen) {
54     if (currentFloor > BOTTOM_FLOOR) {
55     currentFloor--;
56     System.out.println(currentFloor);
57     }
58     else {
59     System.out.println("Already on bottom floor.");
60     }
61     }
```



```
62     else {
63         System.out.println("Doors still open!");
64     }
65 }
66
67 public void setFloor(int desiredFloor) {
68     if ((desiredFloor >= BOTTOM_FLOOR) && (desiredFloor <= TOP_FLOOR)) {
69
70         while (currentFloor != desiredFloor) {
71             if (currentFloor < desiredFloor) {
72                 goUp();
73             }
74
75             else {
76                 goDown();
77             }
78         }
79     }
80     else {
81         System.out.println("Invalid Floor");
82     }
83 }
```



```
84
85     public int getFloor() {
86         return currentFloor;
87     }
88
89     public boolean getDoorStatus() {
90         return doorOpen;
91     }
92 }
93
```




Encapsulated Elevator

Example:

```
1  public class PrivateElevator2Test {  
2  
3      public static void main(String args[]) {  
4  
5          PrivateElevator2 privElevator = new PrivateElevator2();  
6  
7          privElevator.openDoor();  
8          privElevator.closeDoor();  
9          privElevator.goDown();  
10         privElevator.goUp();  
11         privElevator.goUp();  
12         privElevator.openDoor();  
13         privElevator.closeDoor();  
14         privElevator.goDown();  
15         privElevator.openDoor();  
16         privElevator.goDown();  
17         privElevator.closeDoor();  
}
```



```
18     privElevator.goDown();
19     privElevator.goDown();
20
21     int curFloor = privElevator.getFloor();
22
23     if (curFloor != 5 && ! privElevator.getDoorStatus()) {
24         privElevator.setFloor(5);
25     }
26
27     privElevator.setFloor(10);
28     privElevator.openDoor();
29 }
30 }
```



Sample Output

```
The weight is 453
Already on bottom floor.
2
3
The weight is 899
2
Doors still open!
The weight is 974
1
Already on bottom floor.
2
3
4
5
```



Exercise 1: Writing Encapsulated Classes

- Objective
- Tasks
- Solutions
- Discussion



Describing Variable Scope

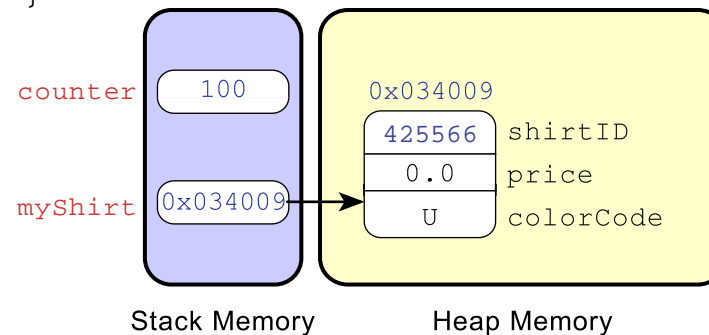
Example:

```
1  public class Person2 {
2
3      // begin scope of int age
4      private int age = 34;
5
6      public void displayName() {
7
8          // begin scope of String name
9          String name = "Peter Simmons";
10         System.out.println("My name is " + name + " and I am " + age );
11
12     }    // end scope of String name
13
14     public String getName () {
15
16         return name; // this causes an error
17     }
18 }    // end scope of int age
```



How Instance Variables and Local Variables Appear in Memory

```
public static void main (String args[]) {  
  
    int counter = 100;  
    Shirt myShirt = new Shirt ( );  
    myShirt.shirtID = 425566 ;  
}
```





Creating Constructors

- Syntax:

```
[modifiers] class ClassName {  
  
    [modifiers] ConstructorName([arguments]) {  
        code_block  
    }  
}
```



Creating Constructors

- Example:

```
1 public class ConstructorShirt1 {
2
3     private int shirtID = 0; // Default ID for the shirt
4     private String description = "-description required-"; // default
5
6     // The color codes are R=Red, B=Blue, G=Green, U=Unset
7     private char colorCode = 'U';
8     private double price = 0.0; // Default price for all items
9     private int quantityInStock = 0; // Default quantity for all items
10
11     public ConstructorShirt1(char startingCode) {
12
13         switch (startingCode) {
14             case 'R':
15             case 'G':
16             case 'B':
17                 colorCode = startingCode;
18                 break;
```




```
19      default:
20          System.out.println("Invalid colorCode. Use R, G, or B");
21      }
22  }
23
24      public char getColorCode() {
25          return colorCode;
26      }
27  } // end of class
28
29
```



Creating Constructors

Example:

```
1  public class ConstructorShirt1Test {  
2  
3      public static void main (String args[]) {  
4  
5          ConstructorShirt1 constShirt = new ConstructorShirt1('R');  
6          char colorCode;  
7  
8          colorCode = constShirt.getColorCode();  
9  
10         System.out.println("Color Code: " + colorCode);  
11     }  
12 }
```



Default Constructor

- Example:

```
ConstructorShirt constShirt = new ConstructorShirt();
```



Default Constructor

- Example:

```
1 public class DefaultShirt {
2
3     private int shirtID = 0; // Default ID for the shirt
4     private String description = "-description required-"; // default
5
6     // The color codes are R=Red, B=Blue, G=Green, U=Unset
7     private char colorCode = 'U';
8     private double price = 0.0; // Default price for all items
9     private int quantityInStock = 0; // Default quantity for all items
10
11     public DefaultShirt() {
12         colorCode = 'R';
13     }
14
15     public char getColorCode() {
16         return colorCode;
17     }
18 } // end of class
19
```



Overloading Constructors

Example:

```
1  public class ConstructorShirt2 {
2
3      private int shirtID = 0; // Default ID for the shirt
4      private String description = "-description required-"; // default
5
6      // The color codes are R=Red, B=Blue, G=Green, U=Unset
7      private char colorCode = 'U';
8      private double price = 0.0; // Default price for all items
9      private int quantityInStock = 0; // Default quantity for all items
10
11     public ConstructorShirt2() {
12         colorCode = 'R';
13     }
14
15     public ConstructorShirt2 (char startingCode) {
16
17         switch (startingCode) {
```



```
18     case 'R':
19     case 'G':
20     case 'B':
21         colorCode = startingCode;
22         break;
23     default:
24         System.out.println("Invalid colorCode. Use R, G, or B");
25     }
26 }
27
28 public ConstructorShirt2 (char startingCode, int startingQuantity) {
29
30     switch (startingCode) {
31     case 'R':
32         colorCode = startingCode;
33         break;
34     case 'G':
35         colorCode = startingCode;
36         break;
37     case 'B':
38         colorCode = startingCode;
39         break;
```



```
40     default:
41         System.out.println("Invalid colorCode. Use R, G, or B");
42     }
43
44     if (startingQuantity > 0 || startingQuantity < 2000) {
45         quantityInStock = startingQuantity;
46     }
47
48     else {
49         System.out.println("Invalid quantity. Must be > 0 or < 2000");
50     }
51 }
52
53 public char getColorCode() {
54     return colorCode;
55 }
56
57
58 public int getQuantityInStock() {
59     return quantityInStock;
60 }
61
62 } // end of class
63
```



Overloading Constructors

Example:

```
1  public class ConstructorShirt2Test {
2
3      public static void main (String args[]) {
4
5          ConstructorShirt2 constShirtFirst = new ConstructorShirt2();
6          ConstructorShirt2 constShirtSecond = new ConstructorShirt2('G');
7          ConstructorShirt2 constShirtThird = new ConstructorShirt2('B',
1000);
8
9          char colorCode;
10         int quantity;
11
12         colorCode = constShirtFirst.getColorCode();
13         System.out.println("Object 1 Color Code: " + colorCode);
14
15         colorCode = constShirtSecond.getColorCode();
16         System.out.println("Object 2 Color Code: " + colorCode);
```




```
17
18     colorCode = constShirtThird.getColorCode();
19     quantity = constShirtThird.getQuantityInStock();
20     System.out.println("Object 3 Color Code: " + colorCode);
21     System.out.println("Object 3 Quantity on Hand: " + quantity);
22 }
23 }
```



Exercise 2: Using Constructors

- Objective
- Tasks
- Solutions
- Discussion



Module 10

Creating and Using Arrays



Overview

- Objectives:
 - Code one-dimensional arrays
 - Set array values using the length attribute and a loop
 - Pass arguments to the main method for use in a program
 - Create two-dimensional arrays
- Relevance



Creating One-Dimensional Arrays

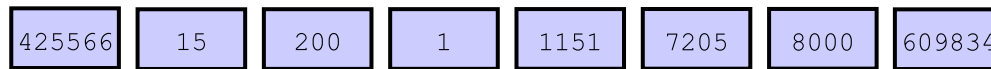
Example:

```
int ageOne = 27;  
int ageTwo = 12;  
int ageThree = 82;  
int ageFour = 70;  
int ageFive = 54;  
int ageSix = 6;  
int ageSeven = 1;  
int ageEight = 30;  
int ageNine = 34;  
int ageTen = 42;
```



Creating One-Dimensional Arrays

Array of `int`



Array of `Shirts`



Array of `Strings`

Hugh Mongus
Aaron Datires
Stan Ding
Albert Kerkie
Carrie DeKeys
Walter Mellon
Hugh Morris
Moe DeLawn



Declaring a One-Dimensional Array

- Syntax:

```
type [] array_identifier;
```

- Examples:

```
char [] status;  
int [] ages;
```

```
Shirt [] shirts;  
String [] names;
```



Instantiating a One-Dimensional Array

- Syntax:

```
array_identifier = new type [length];
```

- Examples:

```
status = new char [20];  
ages = new int [5];
```

```
names = new String [7];  
shirts = new Shirt [3];
```




Initializing a One-Dimensional Array

- Syntax:

```
array_identifier[index] = value;
```

- Examples:

```
ages[0] = 19;
```

```
ages[1] = 42;
```

```
ages[2] = 92;
```

```
ages[3] = 33;
```

```
ages[4] = 46;
```

```
shirts[0] = new Shirt();
```

```
shirts[1] = new Shirt('G');
```

```
shirts[2] = new Shirt('G', 1000);
```



Declaring, Instantiating, and Initializing One-Dimensional Arrays

- Syntax:

```
type [] array_identifier = {comma-separated list of values or expressions};
```

- Examples:

```
int [] ages = {19, 42, 92, 33, 46};
```

```
Shirt [] shirts = {new Shirt(), new Shirt(121,"Work Shirt", 'B', 12.95),  
    new Shirt(122,"Flannel Shirt", 'G', 22.95)};
```

```
int [] ages;  
ages = {19, 42, 92, 33, 46};
```



Accessing a Value Within an Array

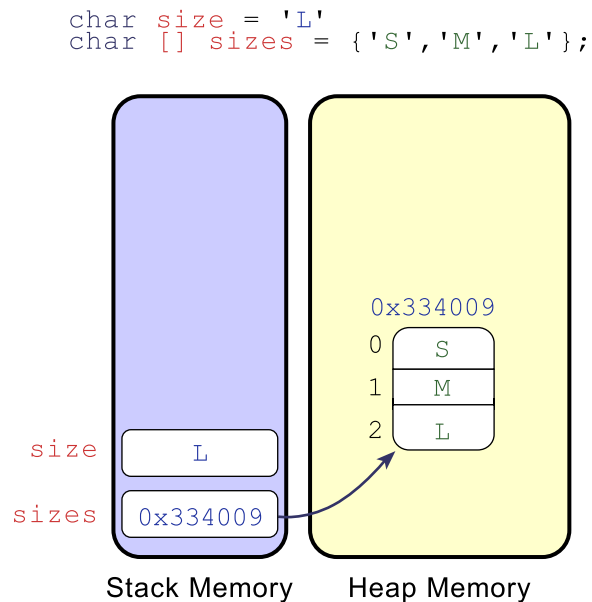
Examples:

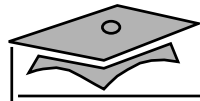
```
status[0] = '3';  
names[1] = "Fred Smith";  
ages[1] = 19;  
prices[2] = 9.99F;
```

```
char s = status[0];  
String name = names [1];  
int age = ages[1];  
double price = prices[2];
```



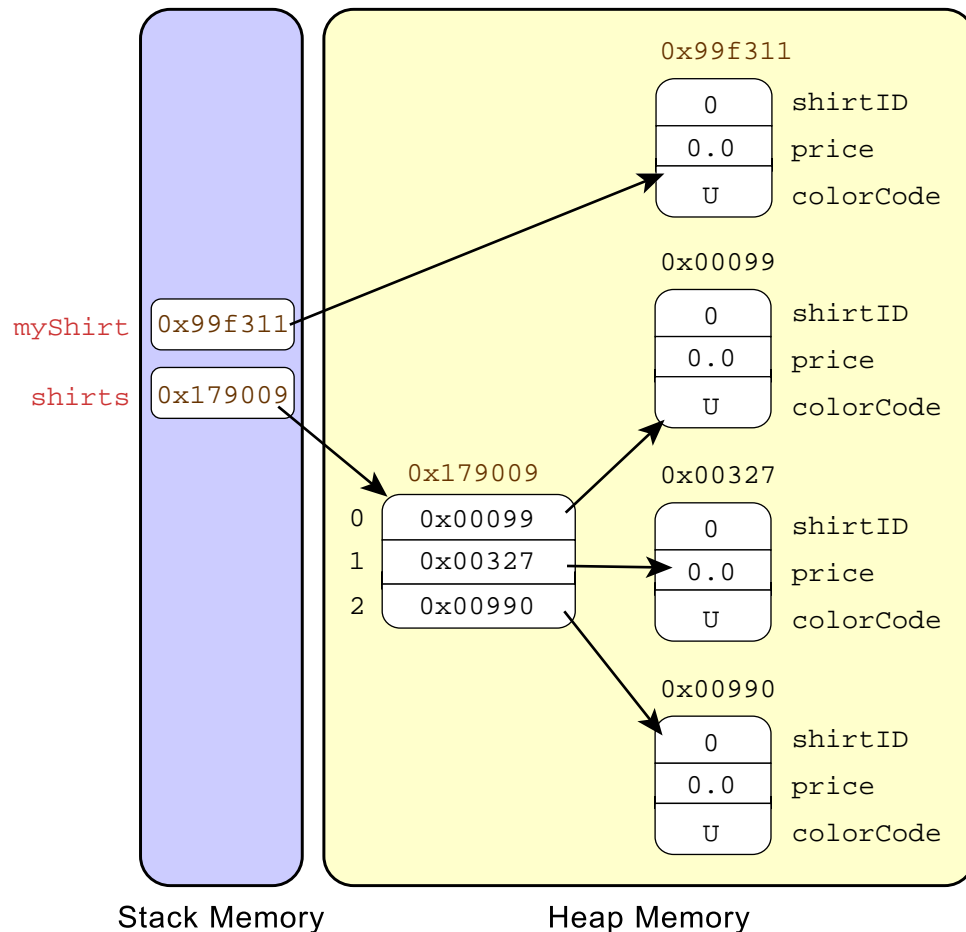
Storing Primitive Variables and Arrays of Primitives in Memory





Storing Reference Variables and Arrays of References in Memory

```
1 Shirt myShirt = new Shirt();  
2 Shirt [] shirts = {new Shirt(),  
                     new Shirt(),  
                     new Shirt()};
```





Exercise 1: Creating and Using One-Dimensional Arrays

- Objective
- Tasks
- Solutions
- Discussion



Exercise 2: Viewing Arrays Using the ObjectTool

- Objective
- Tasks
- Solutions
- Discussion



Setting Array Values Using the `length` Attribute and a Loop

Example:

```
int [] myArray;  
myArray = new int[100];  
  
for (int count = 0; count < myArray.length; count++) {  
    myArray[count] = count;  
}
```




Exercise 3: Using Loops and Arrays

- Objective
- Task
- Solution
- Discussion



Using the args Array in the main Method

- Example:

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

- Example:

```
1  public class ArgsTest {  
2  
3      public static void main (String args[]) {  
4  
5          System.out.println("args[0] is " + args[0]);  
6          System.out.println("args[1] is " + args[1]);  
7      }  
8  }
```



Converting `String` Arguments to Other Types

Example:

```
int ID = Integer.parseInt(args[0]);
```



Exercise 4: Parsing the `args[]` Array

- Objective
- Task
- Solution
- Discussion



Describing Two-Dimensional Arrays

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Week 1							
Week 2							
Week 3							
Week 4							



Declaring a Two-Dimensional Array

- Syntax:

```
type [][] array_identifier;
```

- Example:

```
int [][] yearlySales;
```



Instantiating a Two-Dimensional Array

- Syntax:

```
array_identifier = new type [number_of_arrays] [length];
```

- Example:

```
// Instantiates a two-dimensional array: 5 arrays of 4 elements each  
YearlySales = new int[5][4];
```

	Quarter 1	Quarter 2	Quarter 3	Quarter 4
Year 1				
Year 2				
Year 3				
Year 4				
Year 5				



Initializing a Two-Dimensional Array

Example:

```
yearlySales[0][0] = 1000;  
yearlySales[0][1] = 1500;  
yearlySales[0][2] = 1800;  
yearlySales[1][0] = 1000;  
yearlySales[2][0] = 1400;  
yearlySales[3][3] = 2000;
```

	Quarter 1	Quarter 2	Quarter 3	Quarter 4
Year 1	1000	1500	1800	
Year 2	1000			
Year 3	1400			
Year 4				2000
Year 5				



Exercise 5: Creating and Using Two-Dimensional Arrays

- Objective
- Task
- Solution
- Discussion



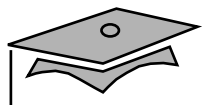
Module 11

Implementing Inheritance



Overview

- Objectives:
 - Define and test your use of inheritance
 - Explain abstraction
 - Explicitly identify class libraries used in your code
- Relevance



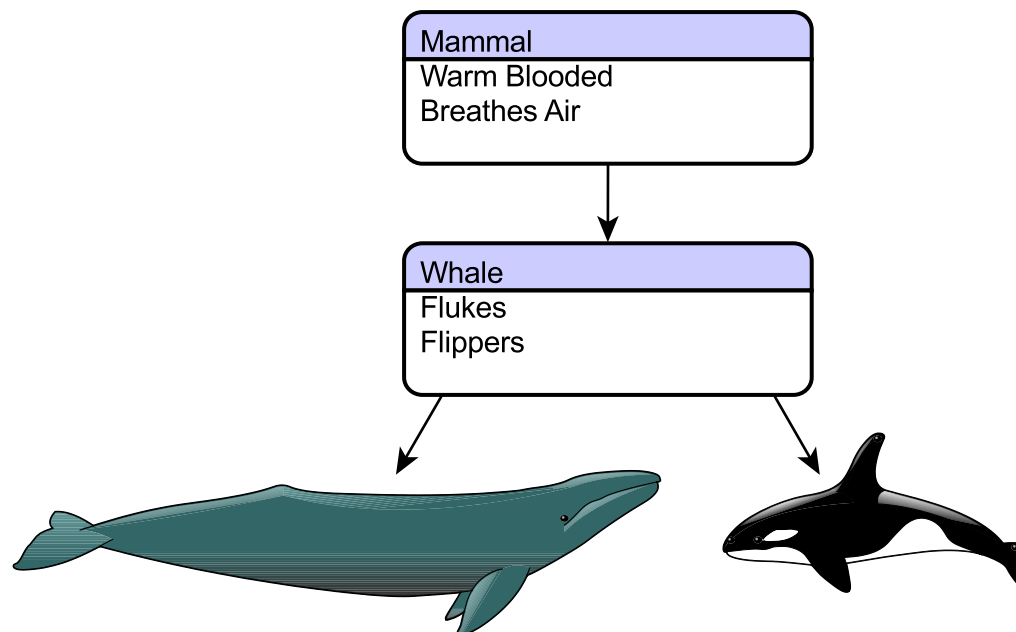
Inheritance

Hat	Sock
hatID price description colorCode R=Red, B=Blue, G=Green quantityInStock	sockID price description colorCode R=Red, B=Blue, G=Green quantityInStock
calculateHatID() displayHatInformation()	calculateSockID() displaySocksInformation()

Pant	Shirt
pantID price size gender M=Male, F=Female description colorCode B=Blue, T=Tan quantityInStock	shirtID price description colorCode R=Red, B=Blue, G=Green quantityInStock
calculatePantID() displayPantInformation()	calculateShirtID() displayShirtInformation()

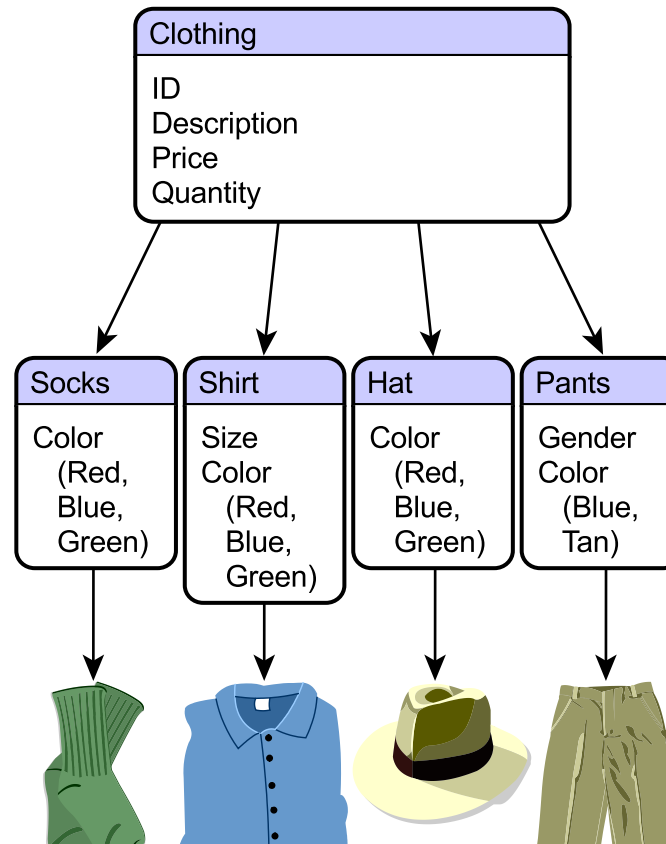


Superclasses and Subclasses





Testing Superclass and Subclass Relationships





Modeling Superclasses and Subclasses

Hat:Clothing	Sock:Clothing
colorCode R=Red, B=Blue, G=Green	colorCode R=Red, B=Blue, G=Green
displayHatInformation()	displaySockInformation()

Pant:Clothing	Shirt:Clothing
size gender M=Male, F=Female colorCode B=Blue, T=Tan	size colorCode R=Red, B=Blue, G=Green
displayClothingInformation()	displayShirtInformation()



Modeling Superclasses and Subclasses

Clothing
ID price description quantityInStock
calculateID()



Declaring a Subclass

- Syntax:

[class_modifier] class class_identifier extends superclass_identifier



Declaring a Subclass

- Example:

```
1 public class Shirt extends Clothing {
2
3     // The color codes are R=Red, B=Blue, G=Green, U=Unset
4     public char colorCode = 'U';
5
6     // This method displays the values for an item
7     public void displayShirtInformation() {
8
9         System.out.println("Shirt ID: " + getID());
10        System.out.println("Shirt description:" + getDescription());
11        System.out.println("Color Code: " + colorCode);
12        System.out.println("Shirt price: " + getPrice());
13        System.out.println("Quantity in stock: " + getQuantityInStock());
14
15    } // end of display method
16 } // end of class
```



Declaring a Subclass

Example:

```
1  public class Clothing {
2
3      private int ID = 0; // Default ID for all clothing
4      private String description = "-description required-"; // default
5
6      private double price = 0.0; // Default price for all clothing
7      private int quantityInStock = 0; // Default quantity for all clothing
8
9      public void calculateID() {
10         int uniqueID;
11
12         uniqueID = ((int)( Math.random()*10000)+1);
13
14         ID = uniqueID;
15     }
16
17     public int getID() {
```



```
18     return ID;
19 }
20
21 public void setDescription(String d) {
22     description = d;
23 }
24
25 public String getDescription() {
26     return description;
27 }
28
29 public void setPrice(double p) {
30     price = p;
31 }
32
33 public double getPrice() {
34     return price;
35 }
36
37 public void setQuantityInStock(int q) {
38     quantityInStock = q;
39 }
```



```
40
41     public int getQuantityInStock() {
42         return quantityInStock;
43     }
44
45 } // end of class
```



Exercise 1: Creating Superclasses and Subclasses

- Objective
- Tasks
- Solutions
- Discussion



Exercise 2: Viewing Class Hierarchies Using the ObjectTool

- Objective
- Tasks
- Solutions
- Discussion



Abstraction

- What is abstraction?
- Abstraction in the DirectClothing, Inc. case study



Classes in the Java API

- Implicitly available classes: the `java.lang` package
- Importing and qualifying classes:
 - The `java.awt` package
 - The `java.applet` package
 - The `java.net` package
 - The `java.io` package
 - The `java.util` package



The `import` Statement

- Syntax:

```
import package_name.class_name;  
import package_name.*;
```

- Example:

```
import java.awt.*;  
public class MyPushButton1 extends Button {  
    // class statements  
  
}
```



Specifying the Fully Qualified Name

- Syntax:

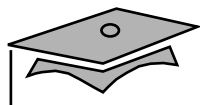
package_name.class_name

- Example:

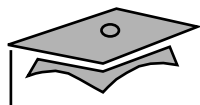
```
public class MyPushButton2 extends java.awt.Button {  
    // class statements  
}
```

Course Contents

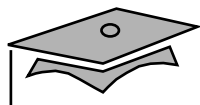
About This Course	Preface-iv
Course Goals	Preface-v
Course Map	Preface-vi
Topics Not Covered	Preface-vii
How Prepared Are You?	Preface-viii
Introductions	Preface-ix
Icons	Preface-x
Typographical Conventions	Preface-xii
Additional Conventions	Preface-xiv
 Explaining Java™ Technology	 1-1
Overview	1-2
Key Concepts of the Java Programming Language	1-3
Object-Oriented	1-4
Distributed	1-6
Simple	1-7
Multithreaded	1-8
Secure	1-9
Platform-Dependent Programs	1-10
Platform-Independent	1-14
Identifying Java Technology Product Groups	1-15
Using the Java™ 2 Platform, Standard Edition SDK Components	1-16
Product Life Cycle (PLC) Stages	1-17
Analysis Stage	1-18
Design Stage	1-19
Development Stage	1-20
Testing Stage	1-21



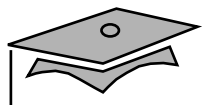
Implementation Stage	1-22
Maintenance Stage	1-23
End-of-Life (EOL) Stage	1-24
Analyzing a Problem and Designing a Solution	2-1
Overview	2-2
Analyzing a Problem Using Object-Oriented Analysis	2-3
Identifying a Problem Domain	2-4
Identifying Objects	2-5
Additional Criteria for Recognizing Objects	2-7
Possible Objects in the DirectClothing Case Study	2-8
Identifying Object Attributes and Operations	2-9
Object With Another Object as an Attribute	2-10
Possible Attributes and Operations in the DirectClothing, Inc. Case Study	2-11
Case Study Solution	2-12
Exercise 1: Analyzing a Problem Domain	2-15
Designing Classes	2-16
Class and Resulting Objects	2-17
Modeling Classes	2-18
Exercise 2: Designing a Solution	2-19
Developing and Testing a Java Technology Program	3-1
Overview	3-2
Identifying the Components of a Class	3-3
Structuring Classes	3-4
Class Declaration	3-7
Variable Declarations and Assignments	3-8
Comments	3-9
Methods	3-10
Creating and Using a Test Class	3-11



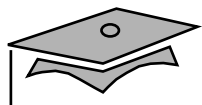
The main Method	3-12
Compiling a Program	3-13
Executing (Testing) a Program	3-14
Debugging Tips	3-15
Exercise: Writing, Compiling, and Testing a Basic Program	3-16
Declaring, Initializing, and Using Variables	4-1
Overview	4-2
Identifying Variable Use and Syntax	4-3
Uses for Variables	4-5
Variable Declaration and Initialization	4-6
Describing Primitive Data Types	4-7
Integral Primitive Types	4-8
Floating Point Primitive Types	4-11
Textual Primitive Type	4-12
Logical Primitive Type	4-13
Naming a Variable	4-14
Assigning a Value to a Variable	4-16
Declaring and Initializing Several Variables in One Line of Code	4-17
Additional Ways to Declare Variables and Assign Values to Variables	4-18
Constants	4-20
Storing Primitives and Constants in Memory	4-21
Exercise 1: Using Primitive Type Variables in a Program	4-22
Standard Mathematical Operators	4-23
Increment and Decrement Operators (++ and --)	4-25
Operator Precedence	4-29
Using Parentheses	4-30
Using Promotion and Type Casting	4-31
Promotion	4-32
Type Casting	4-33



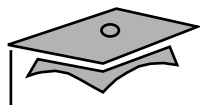
Compiler Assumptions for Integral and Floating Point Data Types	4-35
Floating Point Data Types and Assignment	4-36
Example	4-37
Exercise 2: Using Operators and Type Casting	4-38
 Creating and Using Objects	5-1
Overview	5-2
Declaring Object References, Instantiating Objects, and Initializing Object References	5-3
Declaring Object Reference Variables	5-5
Instantiating an Object	5-6
Initializing Object Reference Variables	5-7
Using an Object Reference Variable to Manipulate Data	5-8
Storing Object Reference Variables in Memory	5-9
Assigning an Object Reference From One Variable to Another	5-10
Exercise 1: Using the ObjectTool to Create and Manipulate Objects	5-11
Exercise 2: Creating a Test Class	5-12
Using the String Class	5-13
Storing String Objects in Memory	5-14
Using Reference Variables for String Objects	5-15
Exercise 3: Using the String Class	5-16
Exercise 4: Examining String Objects With the ObjectTool	5-17
Investigating the Java Class Libraries	5-18
Using the Java Class Library Specification to Learn About a Method	5-20
Exercise 5: Using the Class Library Specification	5-21
 Using Operators and Decision Constructs	6-1
Overview	6-2
Using Relational and Conditional Operators	6-3
Elevator Example	6-4
Elevator Test Class	6-6



Relational Operators	6-7
Testing Equality Between Strings	6-8
Conditional Operators	6-9
The if Construct	6-10
Nested if Statements	6-15
The if/else Construct	6-18
Chaining if/else Constructs	6-23
Exercise 1: Using if and if/else Constructs	6-26
Using the switch Construct	6-27
When to Use switch Constructs	6-30
Exercise 2: Using the switch Construct	6-31
 Using Loop Constructs	 7-1
Overview	7-2
Creating while Loops	7-3
Nested while Loops	7-7
Exercise 1: Using the while Loop	7-12
Developing a for Loop	7-13
Nested for Loops	7-18
Exercise 2: Using the for Loop	7-19
Coding a do/while Loop	7-20
Nested do/while Loops	7-24
Comparing Loop Constructs	7-25
Exercise 3: Using the do/while Loop	7-26
 Developing and Using Methods	 8-1
Overview	8-2
Creating and Invoking Methods	8-3
Basic Form of a Method	8-4
Invoking a Method From a Different Class	8-5



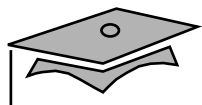
Calling and Worker Methods	8-6
Invoking a Method in the Same Class	8-7
Guidelines for Invoking Methods	8-10
Passing Arguments and Returning Values	8-11
Declaring Methods With Arguments	8-12
The <code>main</code> Method	8-13
Invoking Methods With Arguments	8-14
Declaring Methods With Return Values	8-15
Returning a Value	8-16
Receiving Return Values	8-17
Advantages of Method Use	8-18
Exercise 1: Using Arguments and Return Values	8-19
Creating <code>static</code> Methods and Variables	8-20
Static Methods and Variables in the Java API	8-23
Using Method Overloading	8-25
Method Overloading and the Java API	8-28
Uses for Method Overloading	8-29
Exercise 2: Using Overloaded Methods	8-34
 Implementing Encapsulation and Constructors	9-1
Overview	9-2
Using Encapsulation	9-3
The <code>public</code> Modifier	9-4
The <code>private</code> Modifier	9-8
Interface and Implementation	9-12
Encapsulated Elevator	9-21
Sample Output	9-28
Exercise 1: Writing Encapsulated Classes	9-29
Describing Variable Scope	9-30
How Instance Variables and Local Variables Appear in Memory	9-31



Creating Constructors	9-32
Default Constructor	9-36
Overloading Constructors	9-38
Exercise 2: Using Constructors	9-43

Creating and Using Arrays 10-1

Overview	10-2
Creating One-Dimensional Arrays	10-3
Declaring a One-Dimensional Array	10-5
Instantiating a One-Dimensional Array	10-6
Initializing a One-Dimensional Array	10-7
Declaring, Instantiating, and Initializing One-Dimensional Arrays	10-8
Accessing a Value Within an Array	10-9
Storing Primitive Variables and Arrays of Primitives in Memory	10-10
Storing Reference Variables and Arrays of References in Memory	10-11
Exercise 1: Creating and Using One-Dimensional Arrays	10-12
Exercise 2: Viewing Arrays Using the ObjectTool	10-13
Setting Array Values Using the length Attribute and a Loop	10-14
Exercise 3: Using Loops and Arrays	10-15
Using the args Array in the main Method	10-16
Converting String Arguments to Other Types	10-17
Exercise 4: Parsing the args[] Array	10-18
Describing Two-Dimensional Arrays	10-19
Declaring a Two-Dimensional Array	10-20
Instantiating a Two-Dimensional Array	10-21
Initializing a Two-Dimensional Array	10-22
Exercise 5: Creating and Using Two-Dimensional Arrays	10-23



Implementing Inheritance	11-1
Overview	11-2
Inheritance	11-3
Superclasses and Subclasses	11-4
Testing Superclass and Subclass Relationships	11-5
Modeling Superclasses and Subclasses	11-6
Declaring a Subclass	11-8
Exercise 1: Creating Superclasses and Subclasses	11-13
Exercise 2: Viewing Class Hierarchies Using the ObjectTool	11-14
Abstraction	11-15
Classes in the Java API	11-16
The <code>import</code> Statement	11-17
Specifying the Fully Qualified Name	11-18