

Sun Services

---

# Fundamentals of the Java™ Programming Language

## SL-110-SE6



Copyright 2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decomposition. 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, the Duke logo, Forte, Java, Java Database Connectivity, JDBC, NetBeans, JDK, J2EE, J2ME, J2SE, JVM, JavaStation, Javadoc, Solaris, and Write Once, Run Anywhere 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.**

Export Commodity Classification Number (ECCN) assigned: 25 March 2002

Copyright 2007 Sun Microsystems Inc. 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

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

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

Sun, Sun Microsystems, le logo Sun, le logo Duke, Forte, Java, Java Database Connectivity, JDBC, NetBeans, JDK, J2EE, J2ME, J2SE, JVM, JavaStation, Javadoc, Solaris, et Write Once, Run Anywhere 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.

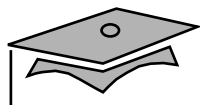
LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION 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.

# Course Contents

---

<b>About This Course .....</b>	<b>Preface-xii</b>
Course Goals .....	Preface-xiii
Course Map .....	Preface-xiv
Topics Not Covered .....	Preface-xv
How Prepared Are You? .....	Preface-xvi
Introductions .....	Preface-xvii
Icons .....	Preface-xviii
Typographical Conventions .....	Preface-xx
Additional Conventions .....	Preface-xxii
 <b>Explaining Java™ Technology .....</b>	 <b>1-1</b>
Objectives .....	1-2
Relevance .....	1-3
Key Concepts of the Java Programming Language .....	1-4
Procedural Programming .....	1-5
Distributed .....	1-7
Simple .....	1-8
Multithreaded .....	1-9
Secure .....	1-10
Platform-Dependent Programs .....	1-11
Platform-Independent Programs .....	1-15
Identifying Java Technology Product Groups .....	1-16
Using the Java Platform, Standard Edition SDK Components .....	1-17
Product Life Cycle (PLC) Stages .....	1-18
Analysis Stage .....	1-19
Design Stage .....	1-20



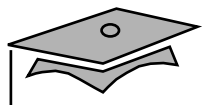
Development Stage .....	1-21
Testing Stage .....	1-22
Implementation Stage .....	1-23
Maintenance Stage .....	1-24
End-of-Life (EOL) Stage .....	1-25

## **Analyzing a Problem and Designing**

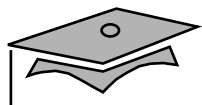
<b>a Solution .....</b>	<b>2-1</b>
Objectives .....	2-2
Relevance .....	2-3
Analyzing a Problem Using OOA .....	2-4
Identifying a Problem Domain .....	2-5
Identifying Objects .....	2-6
Additional Criteria for Recognizing Objects .....	2-8
Possible Objects in the DirectClothing, Inc. Case Study .....	2-9
Identifying Object Attributes and Operations .....	2-10
Object With Another Object as an Attribute .....	2-11
Possible Attributes and Operations for Objects in the DirectClothing, Inc. Case Study .....	2-12
Case Study Solution .....	2-13
Designing Classes .....	2-17
Class and Resulting Objects .....	2-18
Modeling Classes .....	2-19

## **Developing and Testing a Java Technology Program .....**

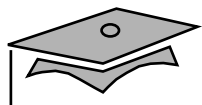
<b>Objectives .....</b>	<b>3-2</b>
<b>Relevance .....</b>	<b>3-3</b>
<b>Identifying the Components of a Class .....</b>	<b>3-4</b>
<b>Structuring Classes .....</b>	<b>3-5</b>
<b>Class Declaration .....</b>	<b>3-7</b>
<b>Variable Declarations and Assignments .....</b>	<b>3-8</b>



Comments .....	3-9
Methods .....	3-10
Creating and Using a Test Class .....	3-11
The <code>main</code> Method .....	3-12
Compiling a Program .....	3-13
Executing (Testing) a Program .....	3-14
Debugging Tips .....	3-15
 <b>Declaring, Initializing, and Using Variables .....</b>	<b>4-1</b>
Objectives .....	4-2
Relevance .....	4-3
Identifying Variable Use and Syntax .....	4-4
Identifying Variable Use and Syntax .....	4-5
Uses for Variables .....	4-6
Variable Declaration and Initialization .....	4-7
Describing Primitive Data Types .....	4-8
Integral Primitive Types .....	4-9
Floating Point Primitive Types .....	4-12
Textual Primitive Type .....	4-13
Logical Primitive Type .....	4-14
Naming a Variable .....	4-15
Assigning a Value to a Variable .....	4-17
Declaring and Initializing Several Variables in One Line of Code .....	4-18
Additional Ways to Declare Variables and Assign Values to Variables .....	4-19
Constants .....	4-21
Storing Primitives and Constants in Memory .....	4-22
Standard Mathematical Operators .....	4-23
Increment and Decrement Operators ( <code>++</code> and <code>--</code> ) .....	4-25
Operator Precedence .....	4-29
Using Parentheses .....	4-30

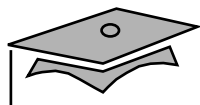


Using Promotion and Type Casting .....	4-31
Promotion .....	4-32
Type Casting .....	4-33
Compiler Assumptions for Integral and Floating Point Data Types .....	4-35
Floating Point Data Types and Assignment .....	4-36
Example .....	4-37
 <b>Creating and Using Objects .....</b>	<b>5-1</b>
Objectives .....	5-2
Relevance .....	5-3
Declaring Object References, Instantiating Objects, and Initializing Object References .....	5-4
Declaring Object Reference Variables .....	5-6
Instantiating an Object .....	5-7
Initializing Object Reference Variables .....	5-8
Using an Object Reference Variable to Manipulate Data .....	5-9
Storing Object Reference Variables in Memory .....	5-10
Assigning an Object Reference From One Variable to Another .....	5-11
Using the String Class .....	5-12
Storing String Objects in Memory .....	5-13
Using Reference Variables for String Objects .....	5-14
Investigating the Java Class Libraries .....	5-15
Using the Java Class Library Specification to Learn About a Method .....	5-17
 <b>Using Operators and Decision Constructs .....</b>	<b>6-1</b>
Objectives .....	6-2
Relevance .....	6-3
Using Relational and Conditional Operators .....	6-4
Elevator Example .....	6-5
The ElevatorTest.java File .....	6-7
Relational Operators .....	6-8

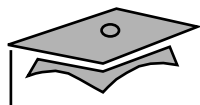


Testing Equality Between Strings .....	6-9
Common Conditional Operators .....	6-10
The if Construct .....	6-11
Nested if Statements .....	6-15
The if/else Construct .....	6-18
The if/else Construct .....	6-18
Chaining if/else Constructs .....	6-22
Using the switch Construct .....	6-25
When to Use switch Constructs .....	6-28
 <b>Using Loop Constructs .....</b>	 <b>7-1</b>
Objectives .....	7-2
Relevance .....	7-3
Creating while Loops .....	7-4
Nested while Loops .....	7-8
Developing a for Loop .....	7-9
Nested for Loops .....	7-13
Coding a do/while Loop .....	7-14
Nested do/while Loops .....	7-18
Comparing Loop Constructs .....	7-20
 <b>Developing and Using Methods .....</b>	 <b>8-1</b>
Overview .....	8-2
Relevance .....	8-3
Creating and Invoking Methods .....	8-4
Basic Form of a Method .....	8-5
Invoking a Method From a Different Class .....	8-6
Calling and Worker Methods .....	8-7
Invoking a Method in the Same Class .....	8-8
Guidelines for Invoking Methods .....	8-11





Passing Arguments and Returning Values .....	8-12
Declaring Methods With Arguments .....	8-13
The <code>main</code> Method .....	8-14
Invoking Methods With Arguments .....	8-15
Declaring Methods With Return Values .....	8-17
Returning a Value .....	8-18
Receiving Return Values .....	8-19
Advantages of Method Use .....	8-21
Creating <code>static</code> Methods and Variables .....	8-22
Static Methods and Variables in the Java API .....	8-26
Using Method Overloading .....	8-28
Method Overloading and the Java API .....	8-31
Uses for Method Overloading .....	8-32
 <b>Implementing Encapsulation and Constructors .....</b>	<b>9-1</b>
Objectives .....	9-2
Relevance .....	9-3
Using Encapsulation .....	9-4
The <code>public</code> Modifier .....	9-5
The <code>private</code> Modifier .....	9-9
Interface and Implementation .....	9-13
Encapsulated Elevator .....	9-22
Sample Output .....	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



<b>Creating and Using Arrays .....</b>	<b>10-1</b>
Objectives .....	10-2
Relevance .....	10-3
Creating One-Dimensional Arrays .....	10-4
Declaring a One-Dimensional Array .....	10-6
Instantiating a One-Dimensional Array .....	10-7
Initializing a One-Dimensional Array .....	10-8
Declaring, Instantiating, and Initializing One-Dimensional Arrays .....	10-9
Accessing a Value Within an Array .....	10-10
Storing Primitive Variables and Arrays of Primitives in Memory .....	10-11
Storing Arrays of References in Memory .....	10-12
Setting Array Values Using the <code>length</code> Attribute and a Loop .....	10-13
Enhanced For Loop .....	10-14
Using the <code>args</code> Array in the <code>main</code> Method .....	10-15
Converting <code>String</code> Arguments to Other Types .....	10-16
The <code>Varargs</code> Feature .....	10-17
Describing Two-Dimensional Arrays .....	10-18
Declaring a Two-Dimensional Array .....	10-19
Instantiating a Two-Dimensional Array .....	10-20
Initializing a Two-Dimensional Array .....	10-21
 <b>Implementing Inheritance .....</b>	 <b>11-1</b>
Objectives .....	11-2
Relevance .....	11-3
Inheritance .....	11-4
Superclasses and Subclasses .....	11-6
Testing Superclass and Subclass Relationships .....	11-7
Modeling Superclasses and Subclasses .....	11-8
Declaring a Superclass .....	11-10
Abstraction .....	11-15

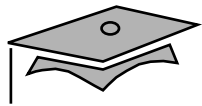


Classes in the Java API .....	11-16
The <code>import</code> Statement .....	11-17
Specifying the Fully Qualified Name .....	11-18



# Preface

## 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

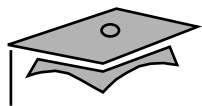
Developing and  
Using Methods

## 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?

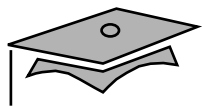
- Can you create and edit text files using a text editor?
- Can you use a World Wide Web (WWW) browser?
- Can you solve logic problems?





# 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



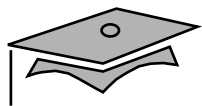
Note



Caution - Electrical



Caution - Heat



# Icons



Case Study



Self-Check



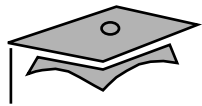
## Typographical Conventions

- `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.



## Typographical Conventions

- *Courier italics bold* is used to represent variables whose values are to be entered by the student as part of an activity.
- *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<sup>™</sup> Operating System (Solaris OS) is different from the Microsoft Windows platform, both commands are shown.

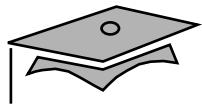


Sun Services

---

# **Module 1**

## Explaining Java™ Technology



## 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

- What is the definition for the following words:
  - Secure
  - Object oriented
  - Independent
  - Dependent
  - Distributed
- What are the stages involved in building something, such as a house or piece of furniture?



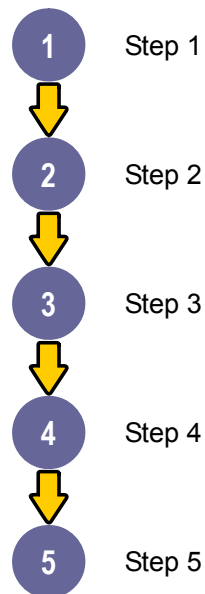
# Key Concepts of the Java Programming Language

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



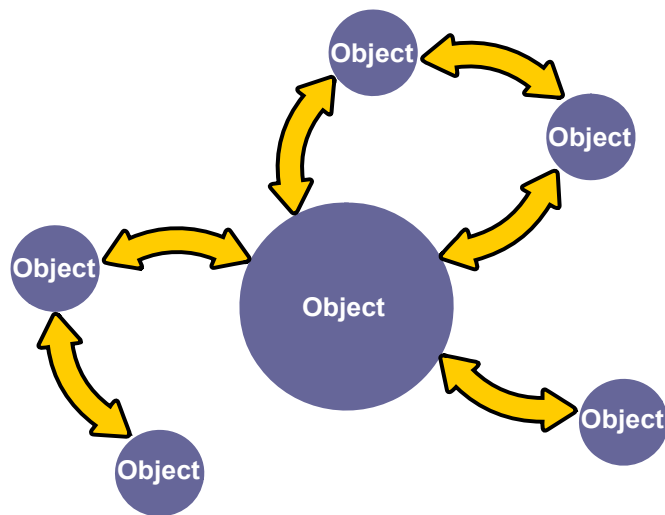
# Procedural Programming

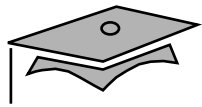
Procedural programming focuses on sequence.



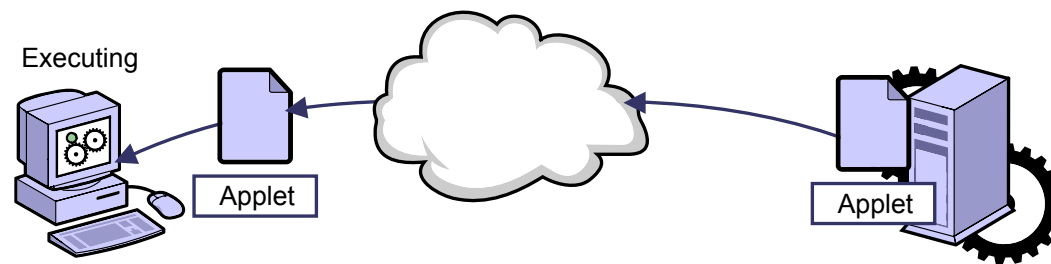


# Object-Oriented





# Distributed



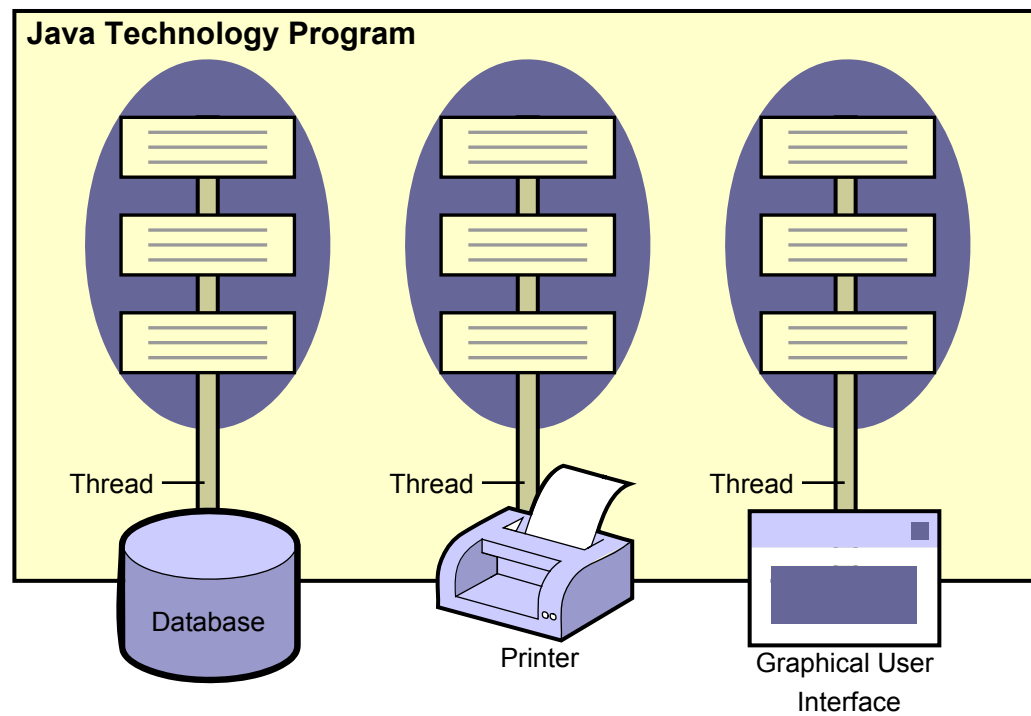


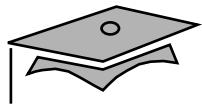
## Simple

- References are used instead of memory pointers.
- A boolean data type can have a value of either `true` or `false`.
- Memory management is automatic.

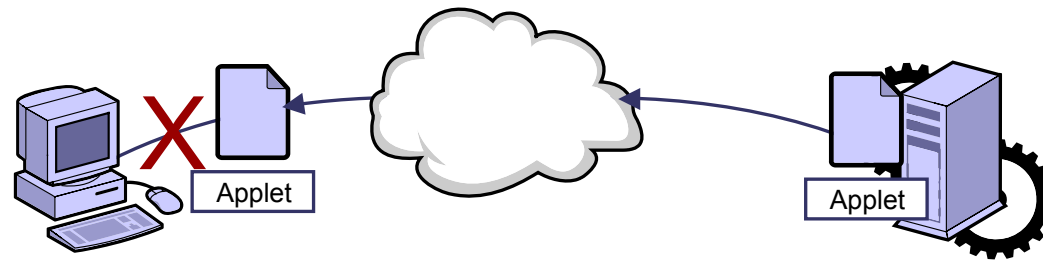


# Multithreaded





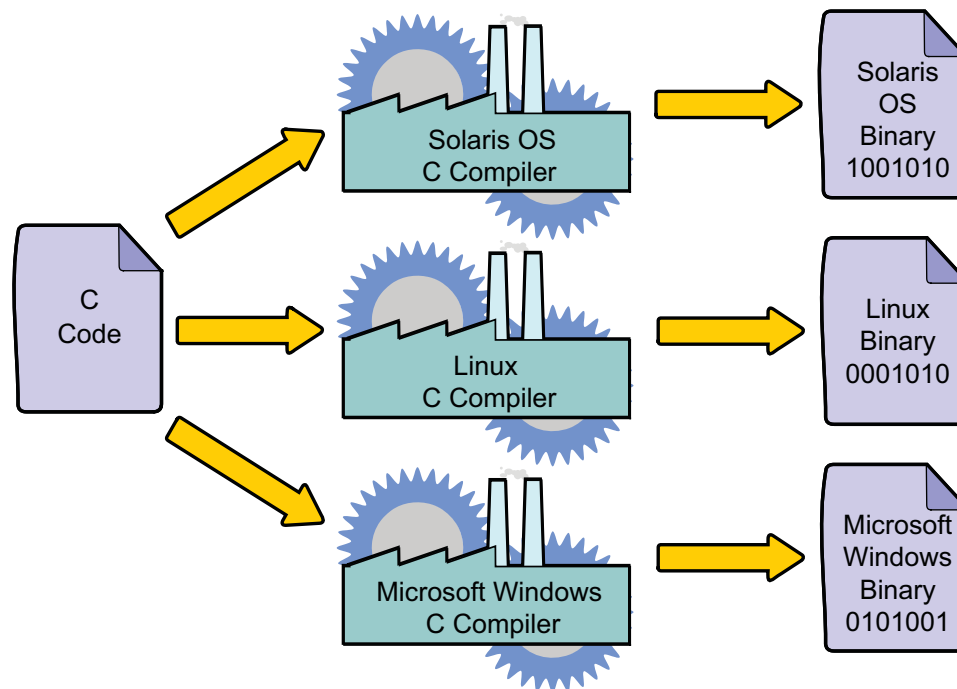
# Secure





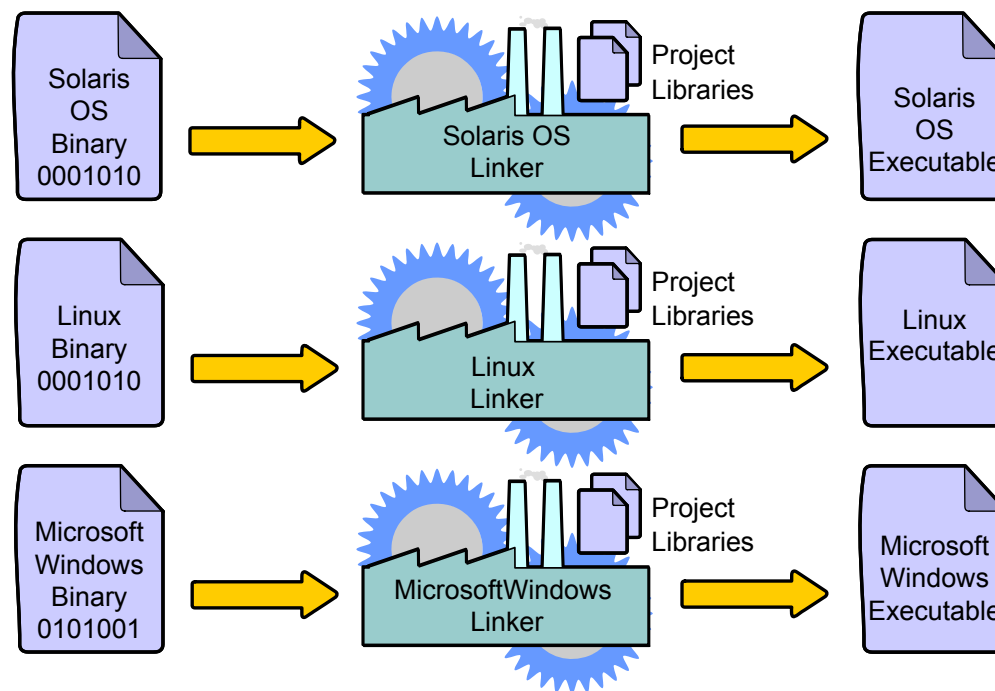


# Platform-Dependent Programs



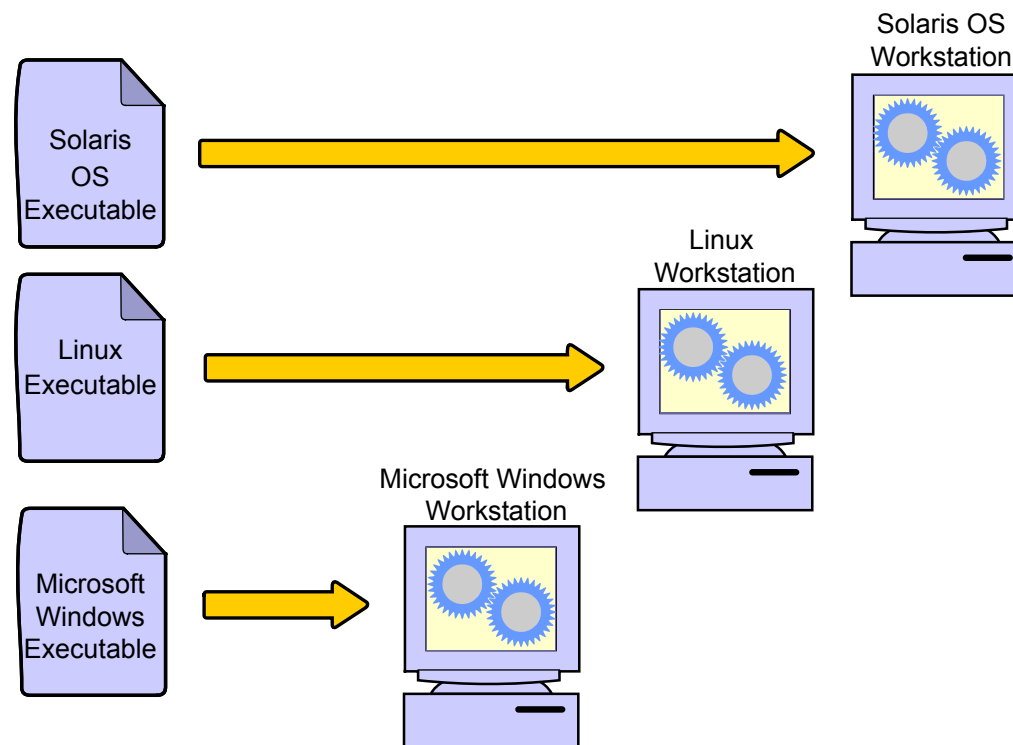


# Platform-Dependent Programs



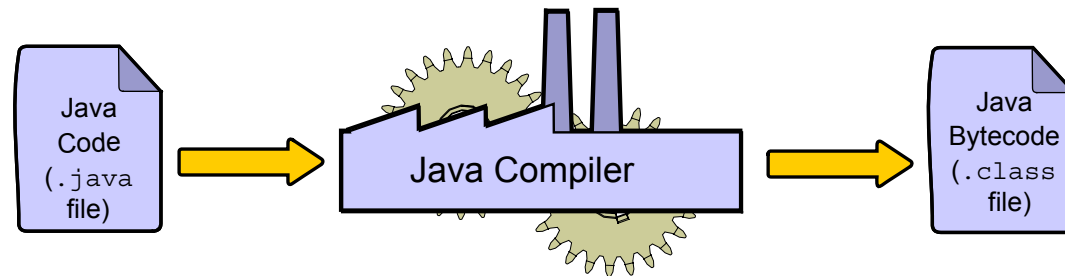


# Platform-Dependent Programs



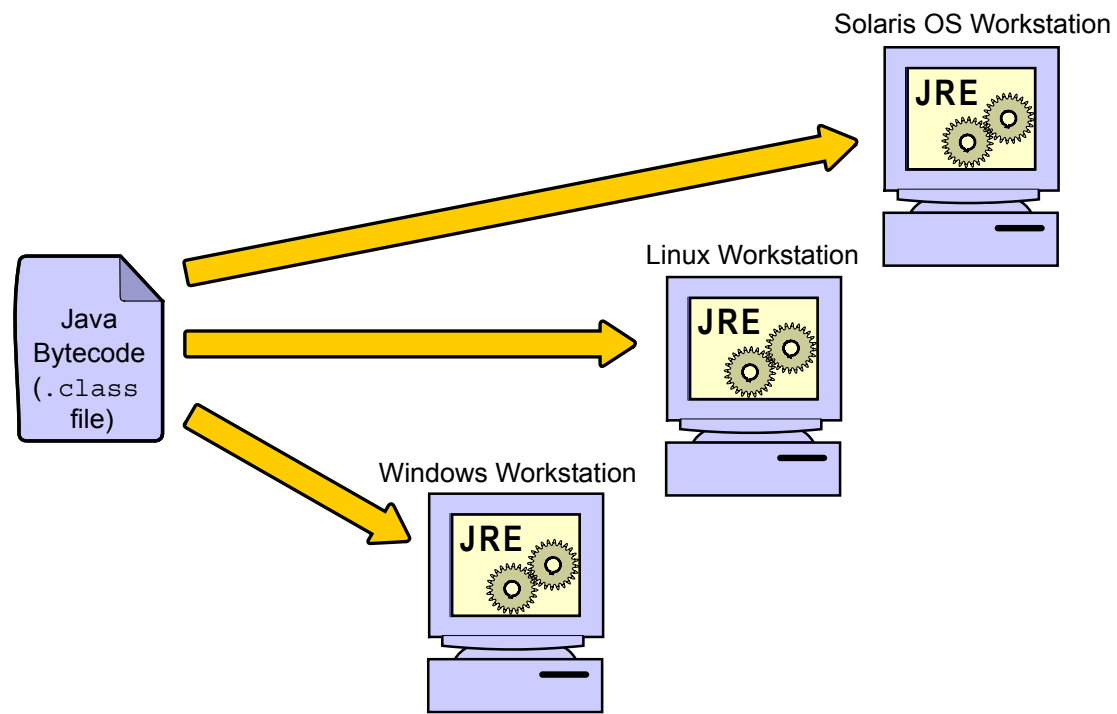


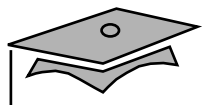
# Platform-Independent Programs



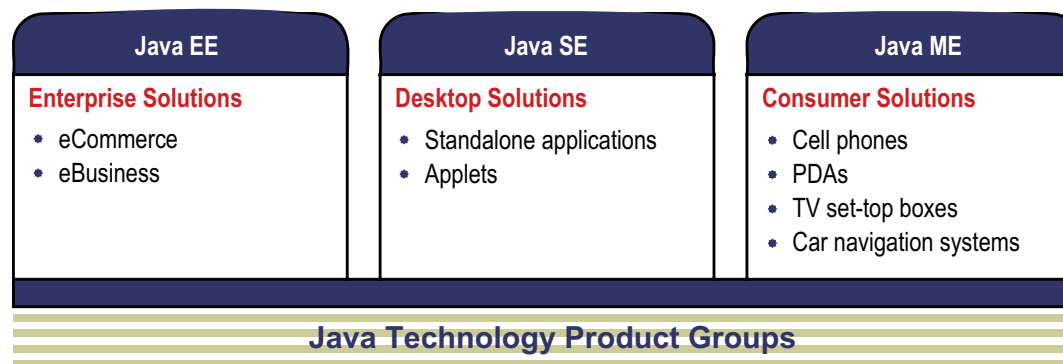


# Platform-Independent Programs





# Identifying Java Technology Product Groups





# Using the Java Platform, Standard Edition SDK Components

- Java runtime environment (JRE):
  - A Java Virtual Machine (JVM™) 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



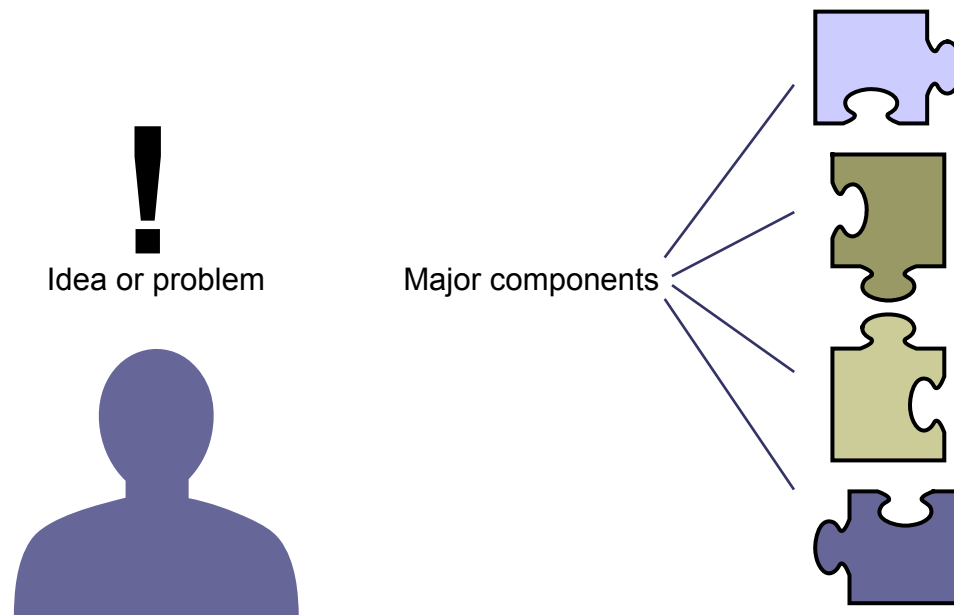
## Product Life Cycle (PLC) Stages

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



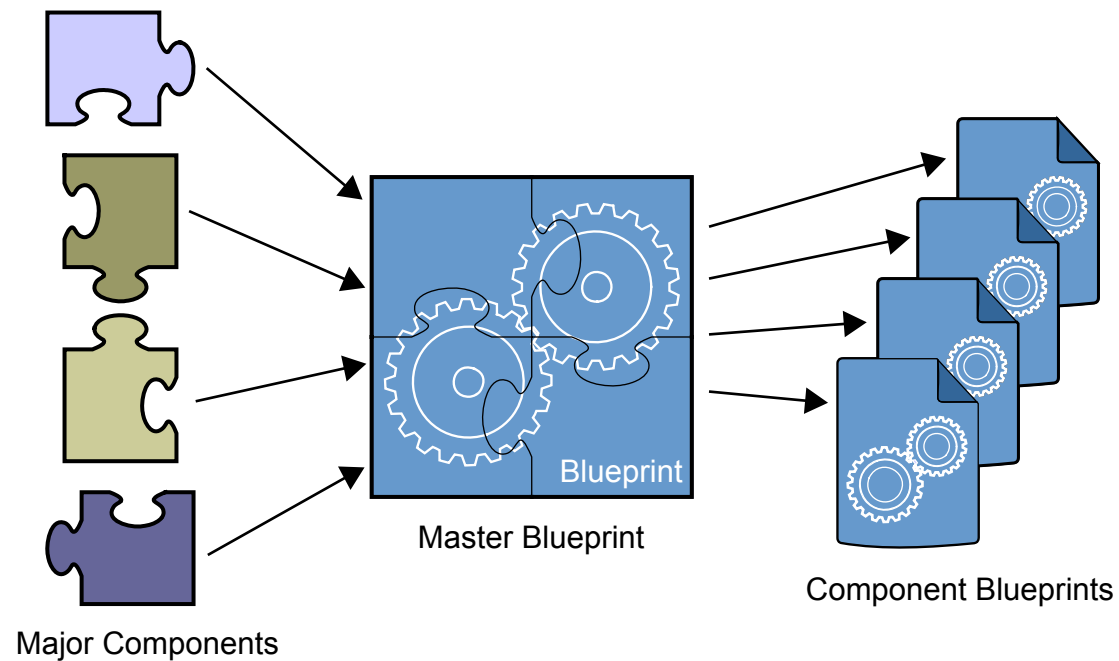


# Analysis Stage



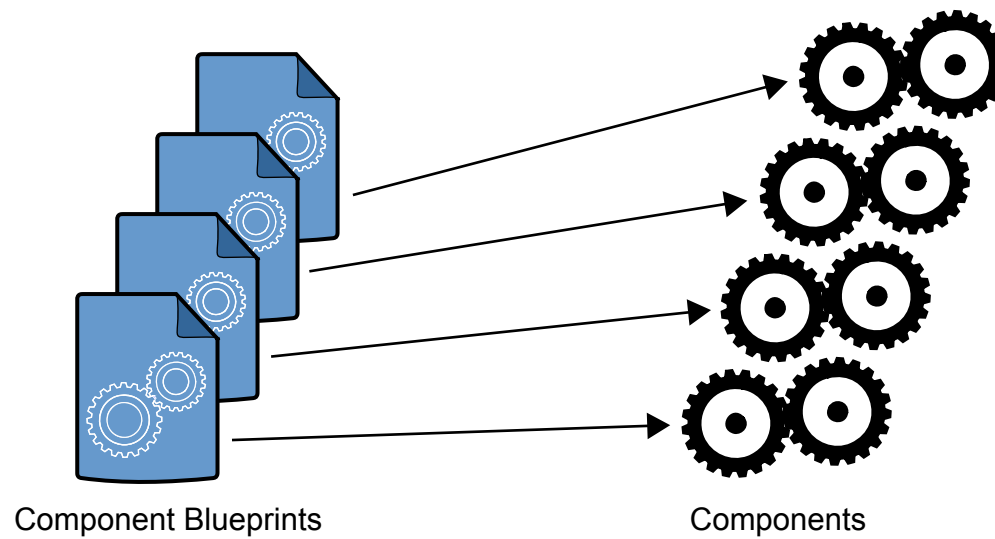


# Design Stage



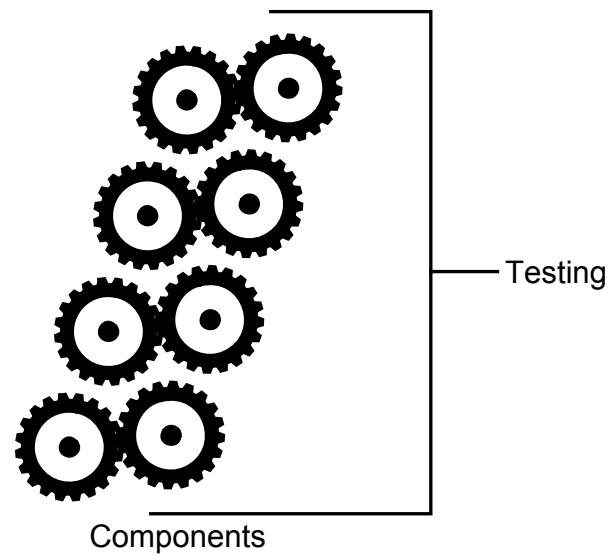


# Development Stage



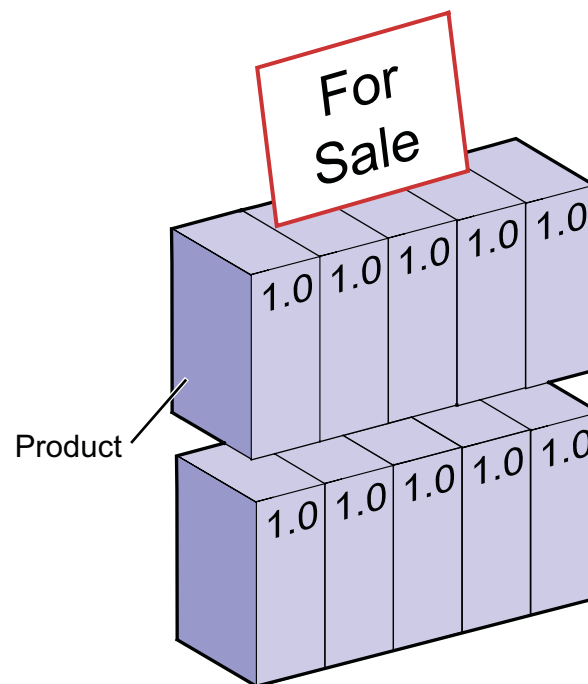


# Testing Stage





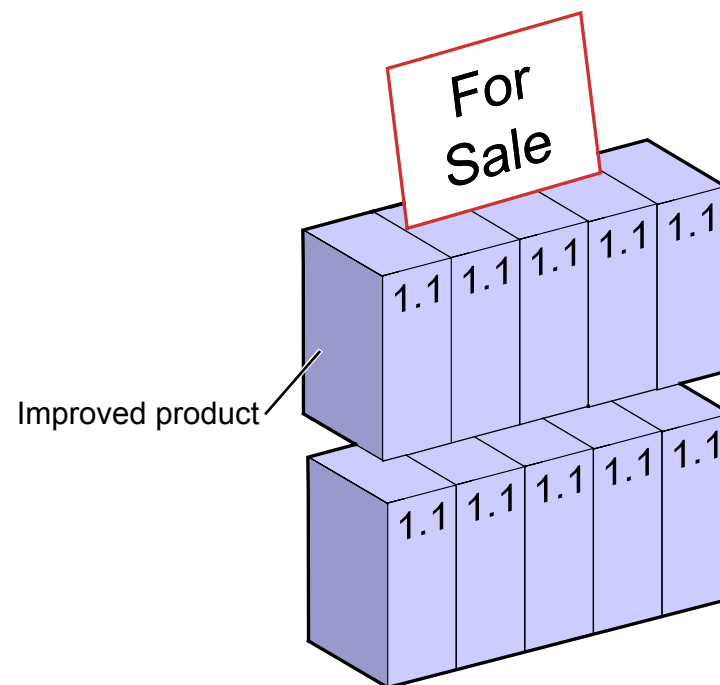
# Implementation Stage



Implementation refers to shipping a product such that customers can purchase it.

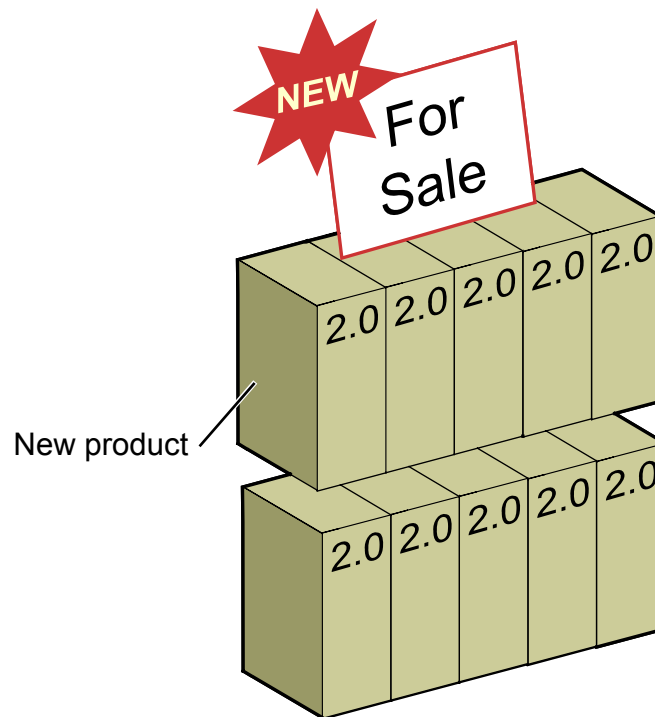


# Maintenance Stage





# End-of-Life (EOL) Stage

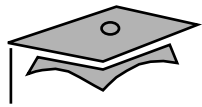




# **Module 2**

## Analyzing a Problem and Designing a Solution





## Objectives

- Analyze a problem using object-oriented analysis (OOA)
- Design classes from which objects will be created



## Relevance

- How do you decide what components are needed for something you are going to build, such as a house or a piece of furniture?
- What is a taxonomy?
- How are organisms in a taxonomy related?
- What is the difference between attributes and values?



## Analyzing a Problem Using OOA

DirectClothing, Inc. sells shirts from their catalog. Business is growing 30 percent per year, and they need a new order entry system.

- DirectClothing produces a catalog of clothing every six months and mails it to subscribers. Each shirt in the catalog has an item identifier (ID), one or more colors (each with a color code), one or more sizes, a description, and a price.
- DirectClothing accepts checks and all credit cards.
- Customers can call DirectClothing to order directly from a customer service representative (CSR), or customers can mail or fax an order form to DirectClothing.



## 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.”

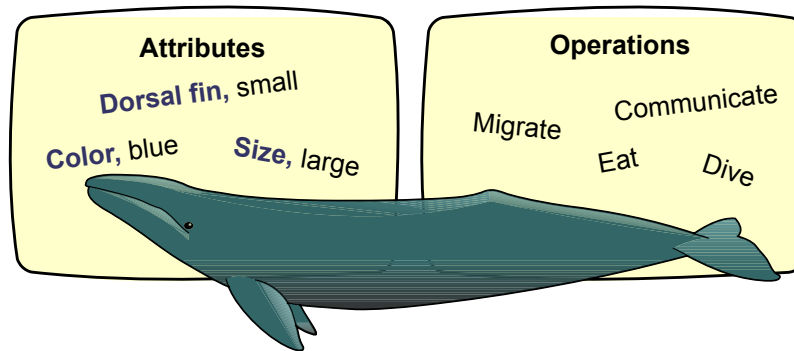


## Identifying Objects

- 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



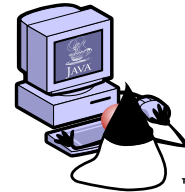
# Possible Objects in the DirectClothing, Inc. Case Study



Order

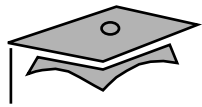


Shirt



Customer



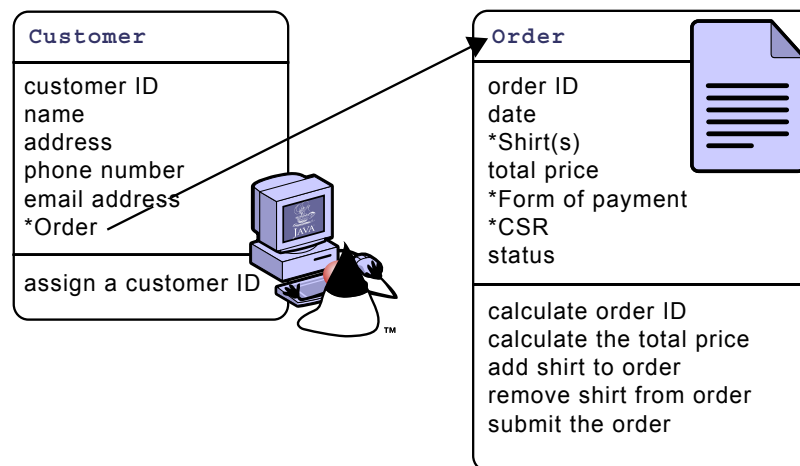


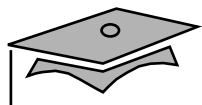
# Identifying Object Attributes and Operations

- 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 for Objects in the DirectClothing, Inc. Case Study

<div>order ID date *Shirt(s) total price *Form of payment *CSR status</div> <div>calculate order ID calculate the total price add shirt to order remove shirt from order submit the order</div>	<div>Order</div> <div></div>	<div>shirtID price description size color code</div> <div>calculate shirt ID display shirt Information</div>	<div>Shirt</div> <div></div>	<div>customer ID name address phone number email address *Order</div> <div>assign a customer ID</div>	<div>Customer</div> <div></div>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------	----------------------------------------------------------------------------------------------------------------------------------	------------------------------	---------------------------------------------------------------------------------------------------------------------------	---------------------------------



## Case Study Solution

Order	Shirt
order ID date *Shirt(s) total price *Form of payment *CSR status	shirt ID price description size color code



Order	Shirt
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

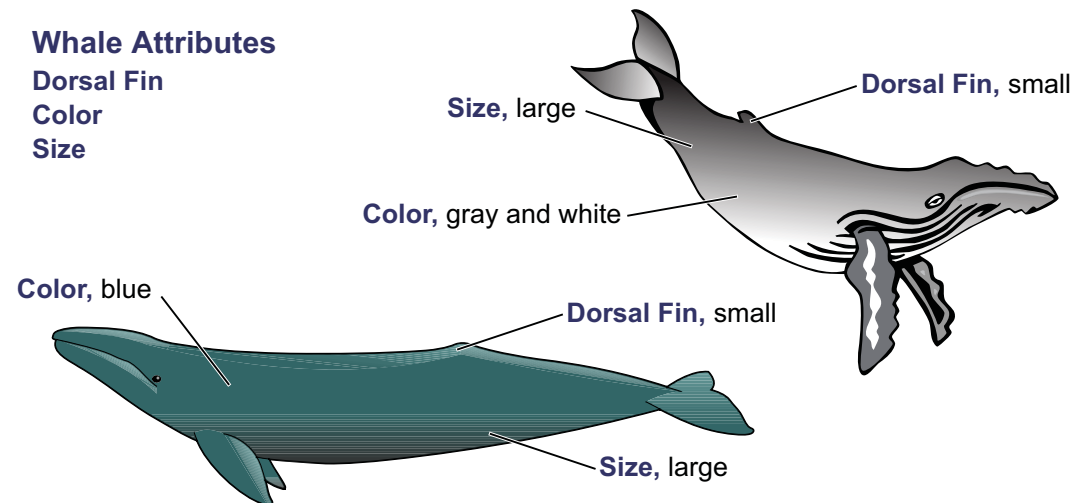


## Case Study Solution

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



# Designing Classes



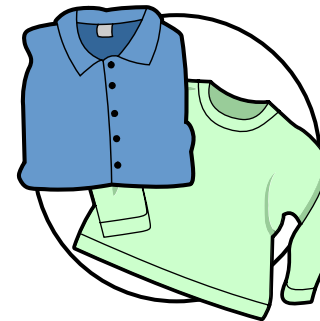




# Class and Resulting Objects

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

Shirt Class



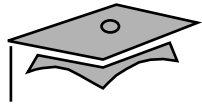
Shirt Objects



# Modeling Classes

- Syntax

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



- Example

<b>Shirt</b>
shirtID price description size colorCode R=Red, B=Blue, G=Green
calculateShirtID() displayInformation()



# **Module 3**

## **Developing and Testing a Java Technology Program**



## 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

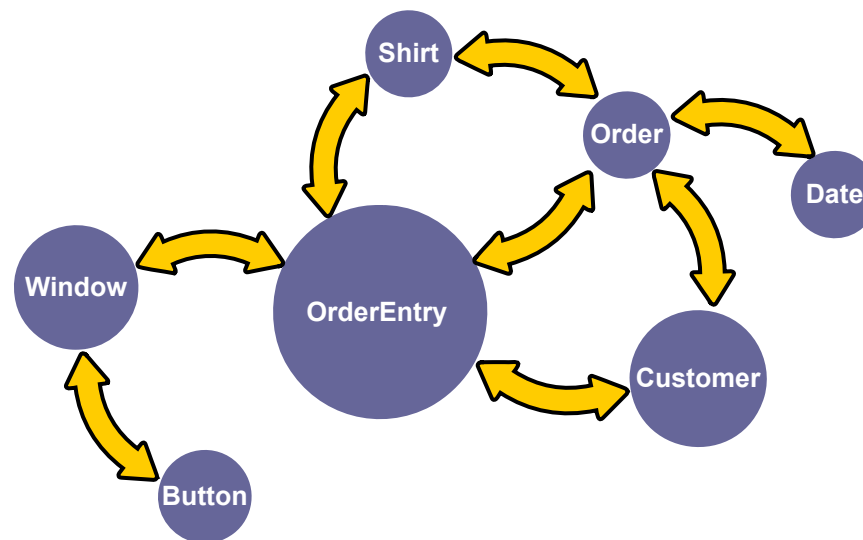


## Relevance

- How do you test something you have built, such as a house, a piece of furniture, or a program?



# 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  // The color codes are R=Red, B=Blue, G=Green, U=Unset
6      public char colorCode = 'U';
7      public double price = 0.0; // Default price for all shirts
8      public int quantityInStock = 0; // Default quantity for all shirts
9
10 // This method displays the values for an item
11     public void displayInformation() {
12         System.out.println("Shirt ID: " + shirtID);
13         System.out.println("Shirt description:" + description);
14         System.out.println("Color Code: " + colorCode);
15         System.out.println("Shirt price: " + price);
16         System.out.println("Quantity in stock: " + quantityInStock);
17
18     } // end of display method
19 } // end of class
```



## Class Declaration

- **Syntax:**  
`[modifiers] 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 displayInformation() {  
  
    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  class ShirtTest {  
2  
3      public static void main (String args[]) {  
4  
5          Shirt myShirt;  
6          myShirt= new Shirt();  
7  
8          myShirt.displayInformation();  
9  
10  
11      }  
12  }  
13
```



## The main Method

- A special method that the JVM recognizes as the starting point for every Java technology program that runs from a command line
- 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.



# **Module 4**

## **Declaring, Initializing, and Using Variables**



## 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

- A variable refers to something that can change. Variables can contain one of a set of values. Where have you seen variables before?
- What types of data do you think variables can hold?



# Identifying Variable Use and Syntax

## Example:

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



# Identifying Variable Use and Syntax

## Example (continued)

```
18      System.out.println("Shirt description:" + description);
19      System.out.println("Color Code: " + colorCode);
20      System.out.println("Shirt price: " + price);
21      System.out.println("Quantity in stock: " + quantityInStock);
22
23  } // end of display method
24
25  } // 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)
- 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`.
- Choose 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

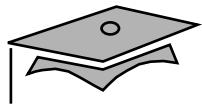
- Variable (can change):

```
double salesTax = 6.25;
```

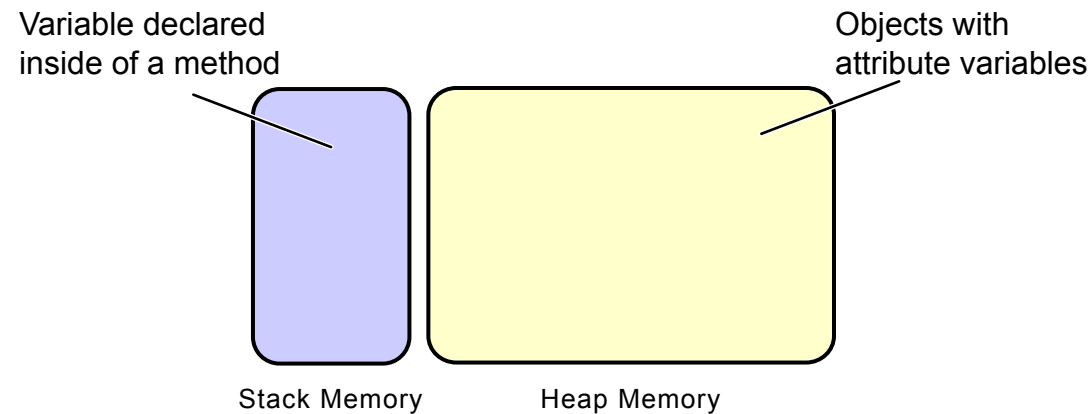
- Constant (cannot change):

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

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



# Storing Primitives and Constants in Memory







# 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 ; //compiler error
```

- 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;//compiler error
```

- 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
```



# **Module 5**

## **Creating and Using Objects**





## 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 Platform, Standard Edition (Java SE™) class library specification to learn about other classes in this application programming interface (API)

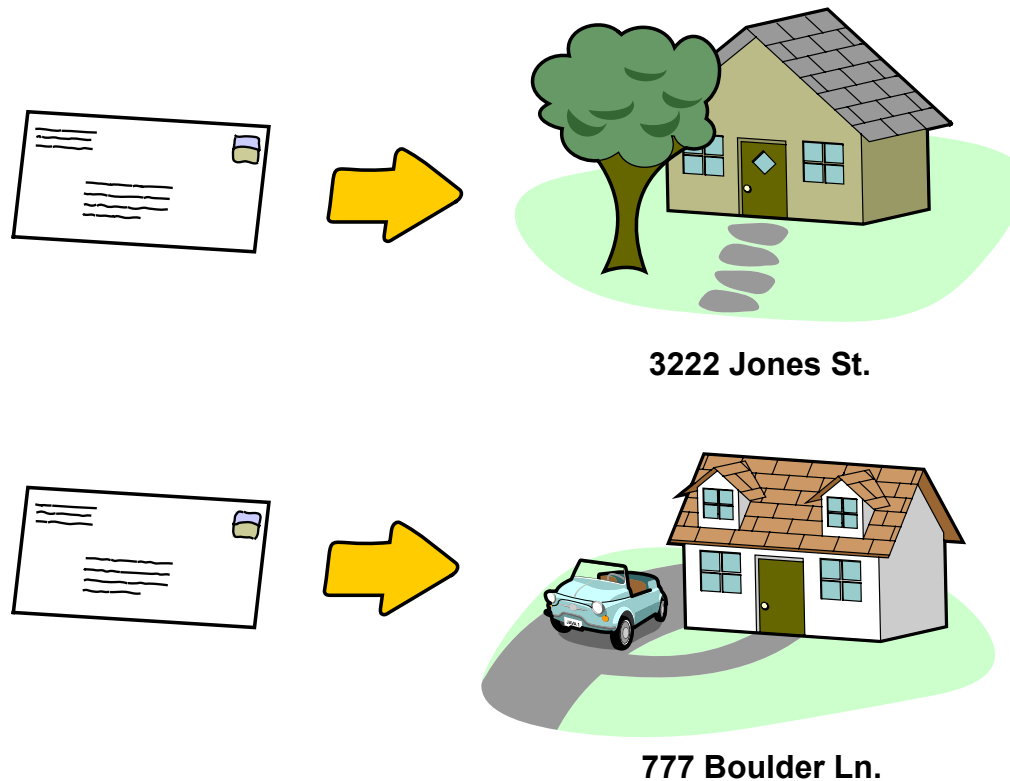


## Relevance

- What does it mean to create an instance of a blueprint for a house?
- How do you refer to different houses on the same street?
- When a builder builds a house, does the builder build every component of the house, including the windows, doors, and cabinets?



# 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.displayInformation();  
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

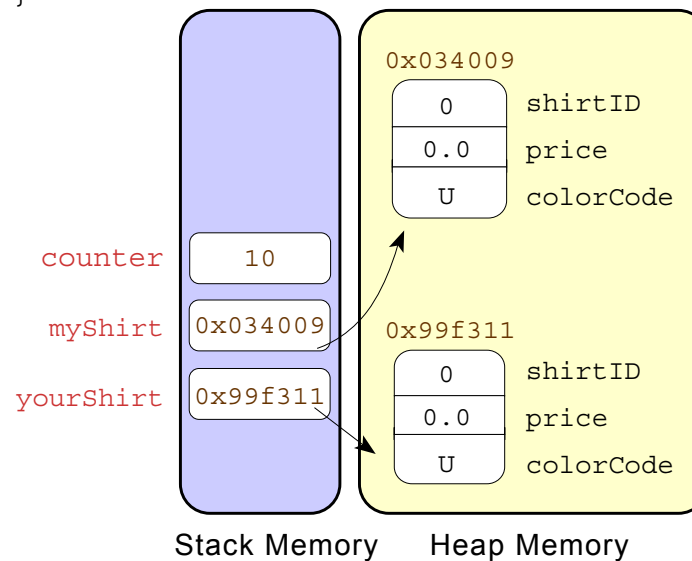
```
1  public class ShirtTestTwo {  
2  
3      public static void main (String args[]) {  
4  
5          Shirt myShirt = new Shirt();  
6          Shirt yourShirt = new Shirt();  
7  
8          myShirt.displayInformation();  
9          yourShirt.displayInformation();  
10  
11         myShirt.colorCode='R';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





# Storing Object Reference Variables in Memory

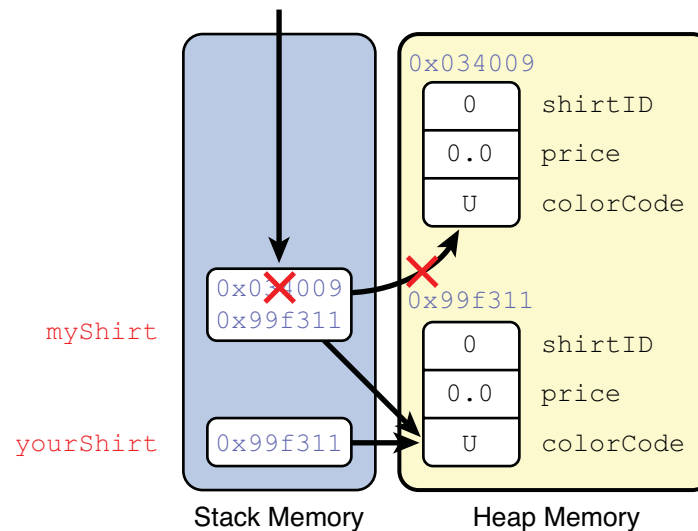
```
public static void main (String args[]) {  
  
    int counter;  
    counter = 10;  
    Shirt myShirt = new Shirt ( );  
}
```





# Assigning an Object Reference From One Variable to Another

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





## Using the String Class

- Creating a String object with the new keyword:

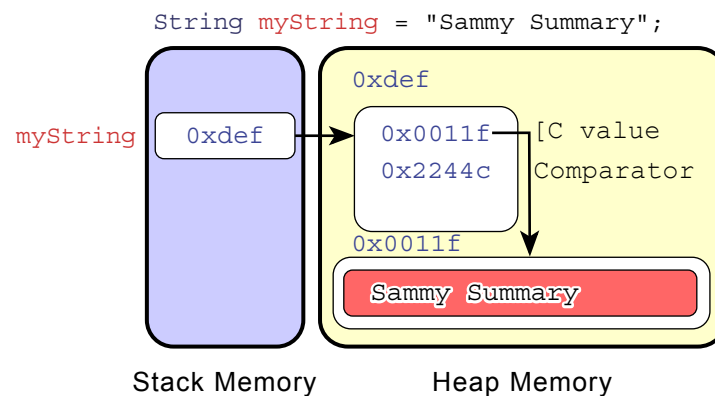
```
String myName = new String("Fred Smith");
```

- 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
```



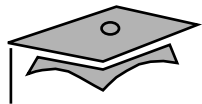
# Investigating the Java Class Libraries

- **Universal Resource Locator (URL) to view the Java SE specification:**

`http://java.sun.com/reference/api/`

- **Example:**

`http://java.sun.com/javase/6/docs/api/`



# 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 packages and classes. The `String` class is selected and highlighted in red. The main content area displays the class details for `java.lang.String`.

**Navigation Pane (Left):**

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

**Main Content Area (Right):**

**Overview Package Class Use Tree Deprecated Index**

**PREV CLASS NEXT CLASS**

**SUMMARY: NESTED | FIELD | CONSTR | METHOD    DETAIL: FIELD | CONSTR | METHOD**

---

**java.lang**

## **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
```





# **Module 6**

## **Using Operators and Decision Constructs**



## Objectives

- Identify relational and conditional operators
- Create `if` and `if/else` constructs
- Use the `switch` constructs

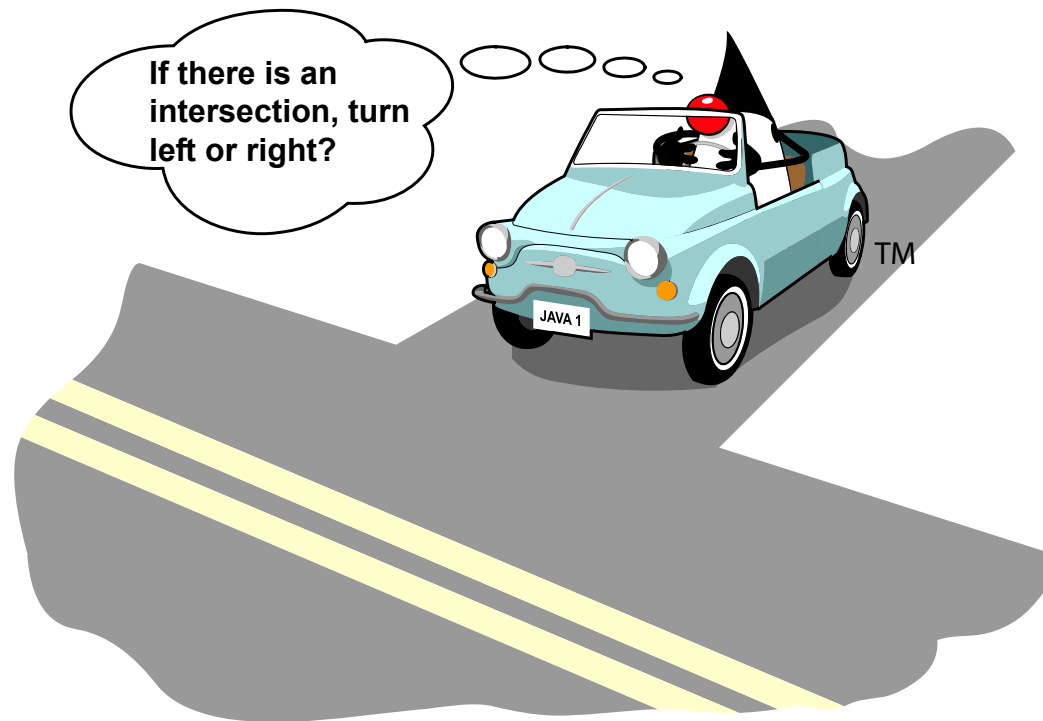


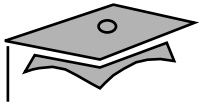
## Relevance

- When you must make a decision that has several different paths, how do you ultimately choose one path over all the other paths?
- For example, what are all of the things that go through your mind when you are going to purchase an item?



# 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 TOP_FLOOR = 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     }
```



## Elevator Example

```
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
33 }
```



## The ElevatorTest.java File

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



# Relational Operators

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





# 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 }  
16
```



# Common Conditional Operators

Operation	Operator	Example
If one condition AND another condition	&&	<pre>int i = 2; int j = 8; ((i &lt; 1) &amp;&amp; (j &gt; 6))</pre>
If either one condition OR another condition		<pre>int i = 2; int j = 8; ((i &lt; 1)    (j &gt; 10))</pre>
NOT	!	<pre>int i = 2; (!(i &lt; 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
2  public class IfElevator {
3
4      public boolean doorOpen=false; // Doors are closed by default
5      public int currentFloor = 1; // All elevators start on first floor
6      public final int TOP_FLOOR = 10;
7      public final int MIN_FLOORS = 1;
8
9      public void openDoor() {
10         System.out.println("Opening door.");
11         doorOpen = true;
12         System.out.println("Door is open.");
13     }
14     public void closeDoor() {
15         System.out.println("Closing door.");
16         doorOpen = false;
```



## The `if` Construct

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



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



## Nested `if` Statements

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





## Nested `if` Statements

```
37         System.out.println("Going down one floor.");
38         currentFloor--;
39         System.out.println("Floor: " + currentFloor);
40     }
41 }
42 }
43
44
45 }
46
```



# 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 TOP_FLOOR = 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     public void closeDoor() {
14         System.out.println("Closing door.");
15         doorOpen = false;
16         System.out.println("Door is closed.");
17     }
```



## The if/else Construct

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



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



## Chaining if/else Constructs

```
18  else if (month == 4 || month == 6 || month == 9 || month == 11) {
19      System.out.println("There are 30 days in that month.");
20  }
21
22  else {
23      System.out.println("Invalid month.");
24  }
25  }
26  }
27
```





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



## Using the switch Construct

```
18  case 2:
19      System.out.println("There are 28 days in that month.");
20      break;
21  case 4:
22  case 6:
23  case 9:
24  case 11:
25      System.out.println("There are 30 days in that month.");
26      break;
27  default:
28      System.out.println("Invalid month.");
29      break;
30  }
31 }
32 }
33
```



## When to Use `switch` Constructs

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



# **Module 7**

## Using Loop Constructs



## Objectives

- Create `while` loops
- Develop `for` loops
- Create `do/while` loops



## Relevance

What are some situations when you would want to continue performing a certain action, as long as a certain condition existed?



# Creating `while` Loops

## Syntax:

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





# Creating while Loops

## Example:

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



## Creating while Loops

```
18  doorOpen = false;
19      System.out.println("Door is closed.");
20  }
21
22  public void goUp() {
23      System.out.println("Going up one floor.");
24      currentFloor++;
25      System.out.println("Floor: " + currentFloor);
26  }
27
28  public void goDown() {
29      System.out.println("Going down one floor.");
30      currentFloor--;
31      System.out.println("Floor: " + currentFloor);
32  }
33
34  public void setFloor() {
35
```



## Creating while Loops

```
36  // Normally you would pass the desiredFloor as an argument to the
37      // setFloor method. However, because you have not learned how to
38      // do this yet, desiredFloor is set to a specific number (5)
39      // below.
40
41      int desiredFloor = 5;
42
43      while (currentFloor != desiredFloor) {
44          if (currentFloor < desiredFloor) {
45              goUp();
46          }
47          else {
48              goDown();
49          }
50      }
51
52  }
53
```



## Nested while Loops

### Example potential solution:

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



# Developing a for Loop

## Syntax:

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



## Developing a for Loop

### Example:

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



## Developing a for Loop

```
18  doorOpen = false;
19      System.out.println("Door is closed.");
20  }
21
22  public void goUp() {
23      System.out.println("Going up one floor.");
24      currentFloor++;
25      System.out.println("Floor: " + currentFloor);
26  }
27
28  public void goDown() {
29      System.out.println("Going down one floor.");
30      currentFloor--;
31      System.out.println("Floor: " + currentFloor);
32  }
33
34  public void setFloor() {
35
```



## Developing a for Loop

```
36 // Normally you would pass the desiredFloor as an argument to the
37 // setFloor method. However, because you have not learned how to
38 // do this yet, desiredFloor is set to a specific number (5)
39 // below.
40 int desiredFloor = 5;
41
42 if (currentFloor > desiredFloor) {
43     for (int down = currentFloor; down != desiredFloor; --down) {
44         goDown();
45     }
46 }
47 else {
48     for (int up = currentFloor; up != desiredFloor; ++up) {
49         goUp();
50     }
51 }
52 }
53 }
54
```





# Nested for Loops

## Example:

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



# Coding a do/while Loop

## Syntax:

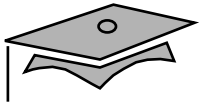
```
do {  
    code_block;  
}  
while (boolean_expression); // Semicolon is mandatory.
```



# Coding a do/while Loop

## Example:

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



## Coding a do/while Loop

```
18  doorOpen = false;
19      System.out.println("Door is closed.");
20  }
21
22  public void goUp() {
23      System.out.println("Going up one floor.");
24      currentFloor++;
25      System.out.println("Floor: " + currentFloor);
26  }
27
28  public void goDown() {
29      System.out.println("Going down one floor.");
30      currentFloor--;
31      System.out.println("Floor: " + currentFloor);
32  }
33
34  public void setFloor() {
35
```



## Coding a do/while Loop

```
36 // Normally you would pass the desiredFloor as an argument to the
37 // setFloor method. However, because you have not learned how to
38 // do this yet, desiredFloor is set to a specific number (5)
39 // below.
40
41 int desiredFloor = 5;
42
43 do {
44     if (currentFloor < desiredFloor) {
45         goUp();
46     }
47     else if (currentFloor > desiredFloor) {
48         goDown();
49     }
50 }
51 while (currentFloor != desiredFloor);
52 }
53
54 }
```



# Nested do/while Loops

## Example:

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



## Nested do/while Loops

```
18  }  
19      while (colCount < width);  
20  
21      System.out.println();  
22      rowCount++;  
23  }  
24  while (rowCount < height);  
25  }  
26  }  
27
```



## 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.





# **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

How do you structure or implement the operations performed on an object?



# Creating and Invoking Methods

## Syntax:

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



## Basic Form of a Method

### Example:

```
public void displayInformation() {  
    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
```



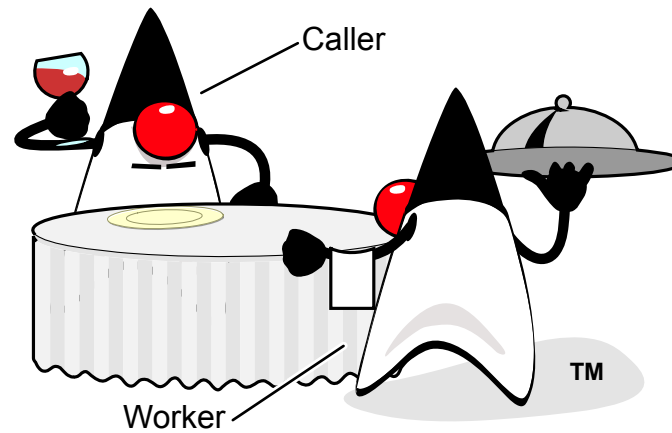
# Invoking a Method From a Different Class

## Example:

```
1
2  public class ShirtTest {
3
4      public static void main (String args[]) {
5
6          Shirt myShirt;
7          myShirt = new Shirt();
8
9          myShirt.displayInformation();
10
11
12      }
13  }
14
```



# Calling and Worker Methods





# Invoking a Method in the Same Class

## Example:

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





## Invoking a Method in the Same Class

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



## Invoking a Method in the Same Class

```
37         goUp();
38     }
39     else {
40         goDown();
41     }
42 }
43 }
44
45 public int getFloor() {
46     return currentFloor;
47 }
48
49 public boolean checkDoorStatus() {
50     return doorOpen;
51 }
52 }
53
```

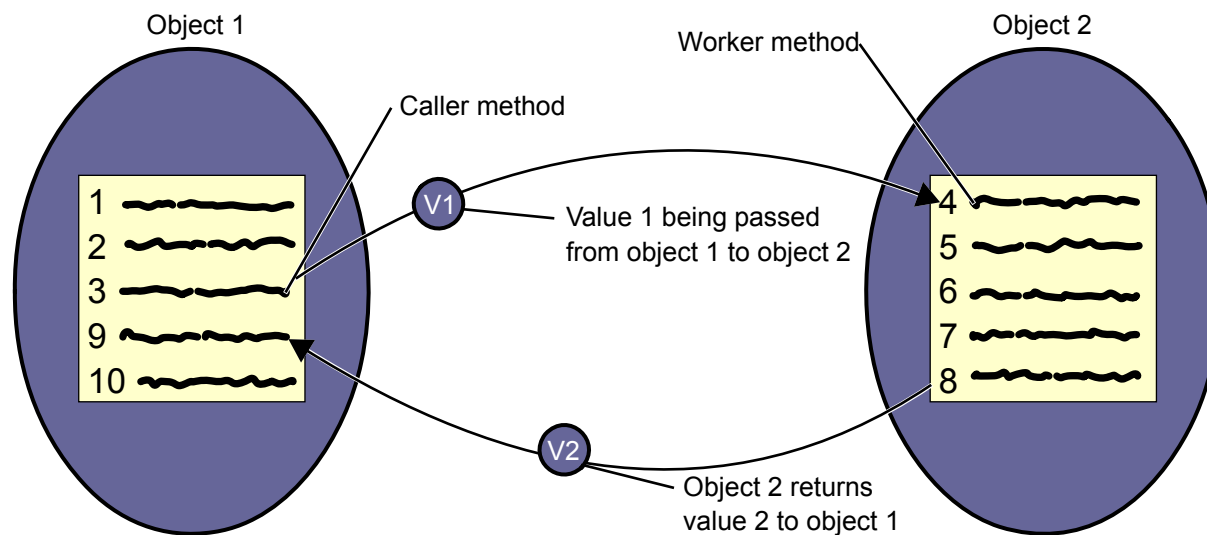


## Guidelines for Invoking Methods

- 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();
```



## Invoking Methods With Arguments

```
17     myElevator.goDown();  
18  
19     myElevator.setFloor(myElevator.TOP_FLOOR);  
20  
21     myElevator.openDoor();  
22 }  
23 }  
24
```





# 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 result= numberOne + numberTwo;  
  
    return result;  
}
```

- **Example:**

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



# Receiving Return Values

## Example:

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



## Receiving Return Values

```
18  myElevator.goDown();
19
20      int curFloor = myElevator.getFloor();
21      System.out.println("Current Floor: " + curFloor);
22
23      myElevator.setFloor(curFloor+1);
24
25      myElevator.openDoor();
26  }
27  }
28
```



## 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.



## 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

- **Example:**

```
char size = Shirt.convertShirtSize(16);
```





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



## Using Method Overloading

```
17
18     public float sum(int numberOne, float numberTwo) {
19
20         System.out.println("Method Three");
21
22         return numberOne + numberTwo;
23     }
24 }
25
```



# 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 }
```



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



## Uses for Method Overloading

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



## Uses for Method Overloading

```
36     System.out.println("Item ID: " + shirtID);
37     System.out.println("Item description:" + description);
38     System.out.println("Color Code: " + colorCode);
39     System.out.println("Item price: " + price);
40     System.out.println("Quantity in stock: " + quantityInStock);
41
42     } // end of display method
43 } // end of class
44
```



# 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 }
17
```



# **Module 9**

## **Implementing Encapsulation and Constructors**



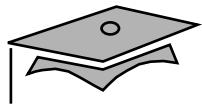
## Objectives

- Use encapsulation to protect data
- Create constructors to initialize objects



## Relevance

- The earliest elevators, or lifts, required a user to manipulate one or more pulleys, ropes, and gears to operate the elevator. Modern elevators hide the detail and only require a user to push a few buttons to operate an elevator. What are the advantages of modern elevators over the older elevators?
- Many elevators, such as a service elevator in a factory, require keys before they can be operated. Other elevators require a key to travel to a particular floor, such as the top floor in a hotel. Why are these keys important?
- What do you think of when you hear the words *private* and *public*?



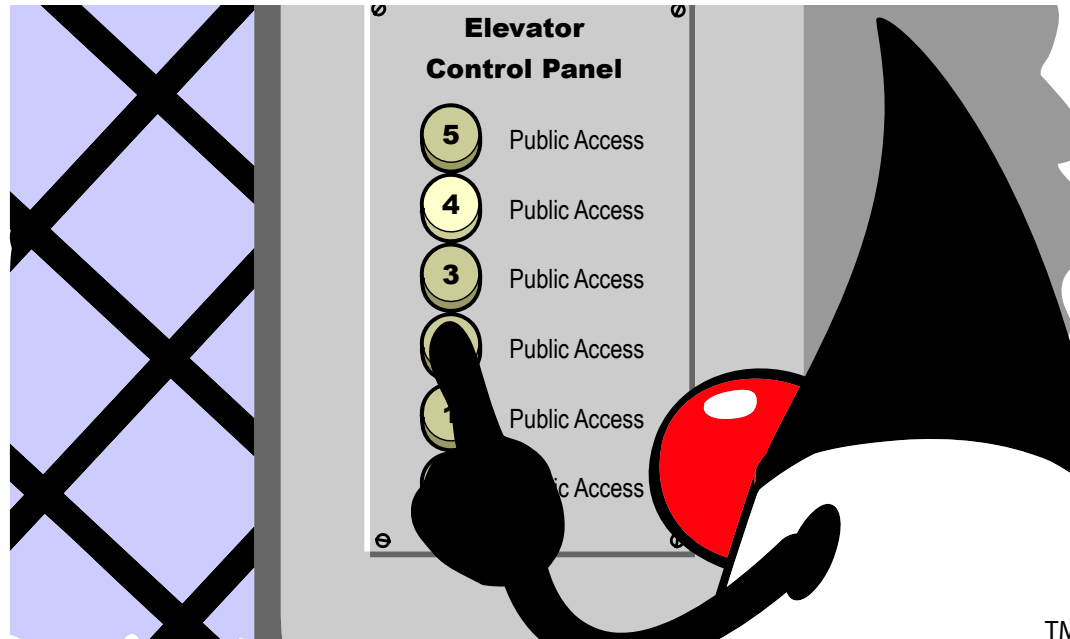
# Using Encapsulation







# The public Modifier



TM

```
public int currentFloor=1;  
  
public void setFloor(int desiredFloor) {  
    ...  
}
```



# The `public` Modifier

## Example:

```
1
2  public class PublicElevator {
3
4      public boolean doorOpen=false;
5      public int currentFloor = 1;
6      public int weight =0;
7
8      public final int CAPACITY=1000;
9      public final int TOP_FLOOR = 5;
10     public final int BOTTOM_FLOOR = 1;
11 }
12
```



# The public Modifier

## Example:

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

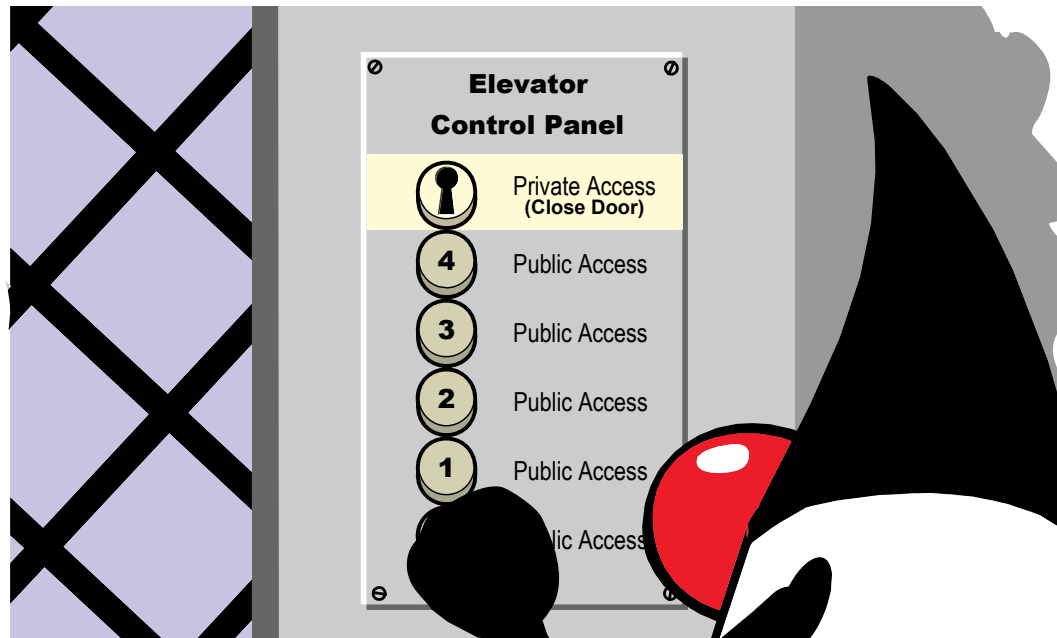


## The `public` Modifier

```
18  pubElevator.currentFloor = 1; //go to the first floor
19      pubElevator.doorOpen = true; //passengers get on/off
20      pubElevator.currentFloor++; //elevator moves with door open
21      pubElevator.doorOpen = false;
22      pubElevator.currentFloor--;
23      pubElevator.currentFloor--;
24  }
25  }
26
```



# The private Modifier



```
private int currentFloor=1;  
private void calculateCapacity() {  
    ...  
}
```



# The private Modifier

## Example:

```
1
2  public class PrivateElevator1 {
3
4      private boolean doorOpen=false;
5      private int currentFloor = 1;
6      private int weight =0;
7
8      private final int CAPACITY=1000;
9      private final int TOP_FLOOR = 5;
10     private final int BOTTOM_FLOOR = 1;
11 }
12
```



# The private Modifier

## Example:

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



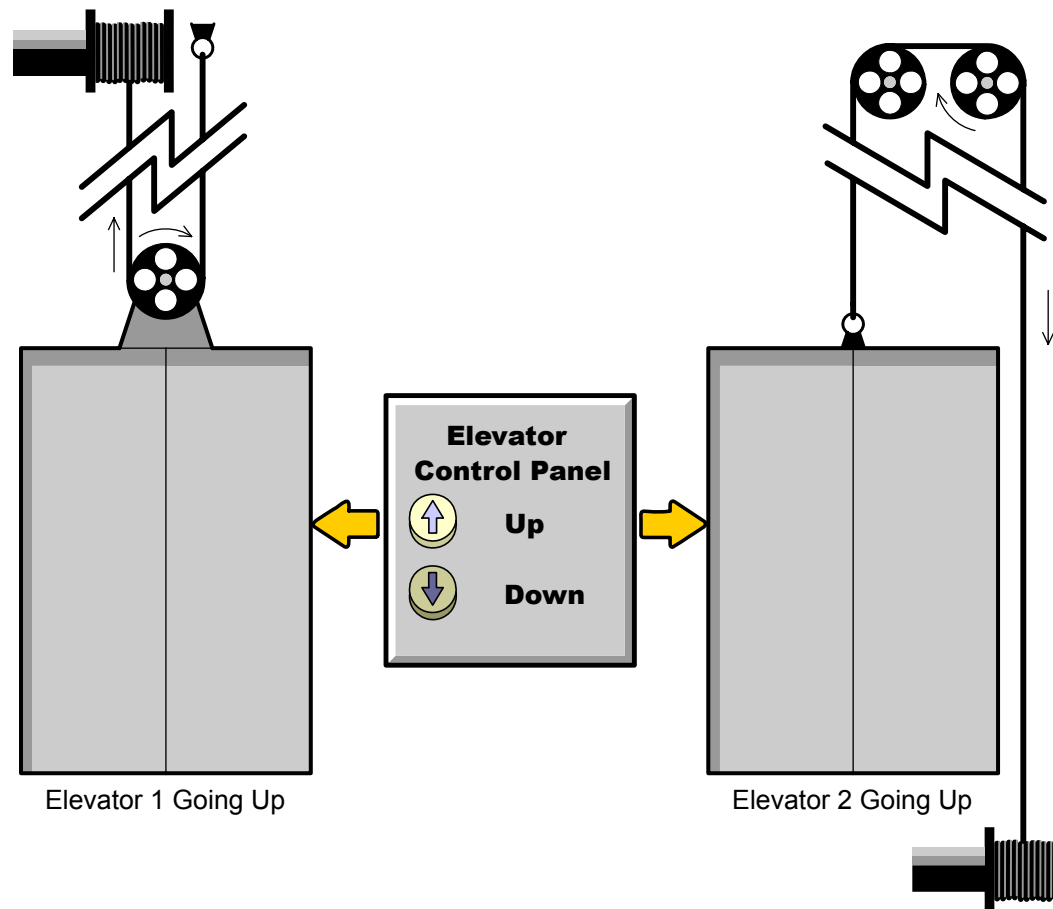
## The private Modifier

```
18     privElevator.currentFloor++;
19
20     //jump to currentFloor 7 (only 5 floors in building)
21     privElevator.currentFloor = 7;
22     privElevator.doorOpen = true; //passengers get on/off
23     privElevator.doorOpen = false;
24     privElevator.currentFloor = 1; //go to the first floor
25     privElevator.doorOpen = true; //passengers get on/off
26     privElevator.currentFloor++; //elevator moves with door open
27     privElevator.doorOpen = false;
28     privElevator.currentFloor--;
29     privElevator.currentFloor--;
30 }
31 }
32
```





# Interface and Implementation





# Interface and Implementation

## Example:

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



## Interface and Implementation

```
18  }  
19  
20  // Additional get and set methods for shirtID, description,  
21  // price, and quantityInStock would follow  
22  
23  } // end of class  
24
```



# Interface and Implementation

## Example:

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



## Interface and Implementation

```
18  colorCode = privShirt.getColorCode();
19
20  // The PrivateShirtTest1 class can set an invalid colorCode
21  System.out.println("Color Code: " + colorCode);
22  }
23  }
24
```



# Interface and Implementation

## Example:

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



## Interface and Implementation

```
18  switch (newCode) {
19      case 'R':
20      case 'G':
21      case 'B':
22          colorCode = newCode;
23          break;
24      default:
25          System.out.println("Invalid colorCode. Use R, G, or B");
26      }
27  }
28
29  // Additional get and set methods for shirtID, description,
30  // price, and quantityInStock would follow
31
32  } // end of class
33
```



# Interface and Implementation

## Example:

```
1
2  public class PrivateShirt2Test {
3
4      public static void main (String args[]) {
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();
```





## Interface and Implementation

```
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 }
24
```



# Encapsulated Elevator

## Example:

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



## Encapsulated Elevator

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



## Encapsulated Elevator

```
37
38     public void goUp() {
39         if (!doorOpen) {
40             if (currentFloor < TOP_FLOOR) {
41                 currentFloor++;
42                 System.out.println(currentFloor);
43             }
44             else {
45                 System.out.println("Already on top floor.");
46             }
47         }
48         else {
49             System.out.println("Doors still open!");
50         }
51     }
52
53     public void goDown() {
54         if (!doorOpen) {
55             if (currentFloor > BOTTOM_FLOOR) {
```



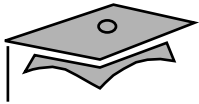
## Encapsulated Elevator

```
56         currentFloor--;
57         System.out.println(currentFloor);
58     }
59     else {
60         System.out.println("Already on bottom floor.");
61     }
62 }
63 else {
64     System.out.println("Doors still open!");
65 }
66 }
67
68 public void setFloor(int desiredFloor) {
69     if ((desiredFloor >= BOTTOM_FLOOR) && (desiredFloor <= TOP_FLOOR)) {
70
71         while (currentFloor != desiredFloor) {
72             if (currentFloor < desiredFloor) {
73                 goUp();
74             }
```



## Encapsulated Elevator

```
75
76     else {
77         goDown();
78     }
79     }
80     }
81     else {
82         System.out.println("Invalid Floor");
83     }
84 }
85
86 public int getFloor() {
87     return currentFloor;
88 }
89
90 public boolean getDoorStatus() {
91     return doorOpen;
92 }
93 }
```



# Encapsulated Elevator

## Example:

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



## Encapsulated Elevator

```
18  privElevator.closeDoor();
19      privElevator.goDown();
20      privElevator.goDown();
21
22      int curFloor = privElevator.getFloor();
23
24      if (curFloor != 5 && ! privElevator.getDoorStatus()) {
25          privElevator.setFloor(5);
26      }
27
28      privElevator.setFloor(10);
29      privElevator.openDoor();
30  }
31  }
32
```





## 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
```



# Describing Variable Scope

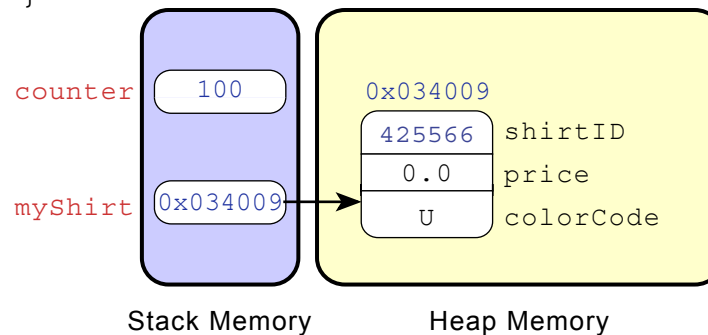
## Example:

```
1  public class Person2 {
2
3      // begin scope of int age
4      private int age = 34;
5
6      public void displayName() {
7          // begin scope of String name
8          String name = "Peter Simmons";
9          System.out.println("My name is " + name + " and I am " + age );
10
11     }    // end scope of String name
12
13     public String getName () {
14
15         return name; // this causes an error
16     }
17 }    // 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
2  public class ConstructorShirt1 {
3
4      private int shirtID = 0; // Default ID for the shirt
5      private String description = "-description required-"; // default
6
7      // The color codes are R=Red, B=Blue, G=Green, U=Unset
8      private char colorCode = 'U';
9      private double price = 0.0; // Default price for all items
10     private int quantityInStock = 0; // Default quantity for all items
11
12     public ConstructorShirt1(char startingCode) {
13
14         switch (startingCode) {
15             case 'R':
16             case 'G':
17             case 'B':
```



## Creating Constructors

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



# Creating Constructors

## Example:

```
1
2  public class ConstructorShirt1Test {
3
4      public static void main (String args[]) {
5
6          ConstructorShirt1 constShirt = new ConstructorShirt1('R');
7          char colorCode;
8
9          colorCode = constShirt.getColorCode();
10
11         System.out.println("Color Code: " + colorCode);
12     }
13 }
14
```



## Default Constructor

- **Example:**

```
ConstructorShirt1 constShirt = new ConstructorShirt1();
```

- **Example:**

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





## Default Constructor

```
15
16     public char getColorCode() {
17         return colorCode;
18     }
19 } // end of class
20
```



# Overloading Constructors

## Example:

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



# Overloading Constructors

```
18  switch (startingCode) {
19      case 'R':
20      case 'G':
21      case 'B':
22          colorCode = startingCode;
23          break;
24      default:
25          System.out.println("Invalid colorCode. Use R, G, or B");
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;
```



# Overloading Constructors

```
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  }
```



# Overloading Constructors

```
56 public int getQuantityInStock() {  
57     return quantityInStock;  
58 }  
59  
60 } // end of class  
61
```



# Overloading Constructors

## Example:

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



## Overloading Constructors

```
17 System.out.println("Object 2 Color Code: " + colorCode);
18
19     colorCode = constShirtThird.getColorCode();
20     quantity = constShirtThird.getQuantityInStock();
21     System.out.println("Object 3 Color Code: " + colorCode);
22     System.out.println("Object 3 Quantity on Hand: " + quantity);
23 }
24 }
25
```



# **Module 10**

## Creating and Using Arrays





## 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

- An array is an orderly arrangement of something, such as an ordered list. What are some things that people use arrays for in their daily lives?
- If a one-dimensional array is a list of items, what is a two-dimensional array?
- How do you access items in an array?



# 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('G'), new Shirt('g',1000)};
```

- **Error:**

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





# Accessing a Value Within an Array

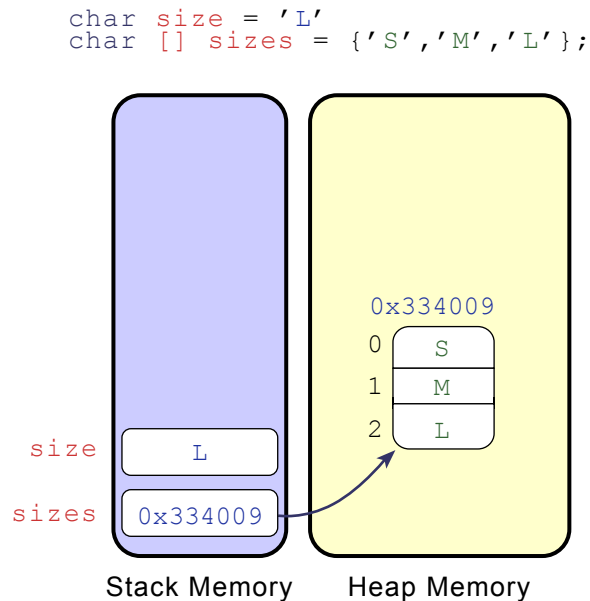
## Examples:

```
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];
```



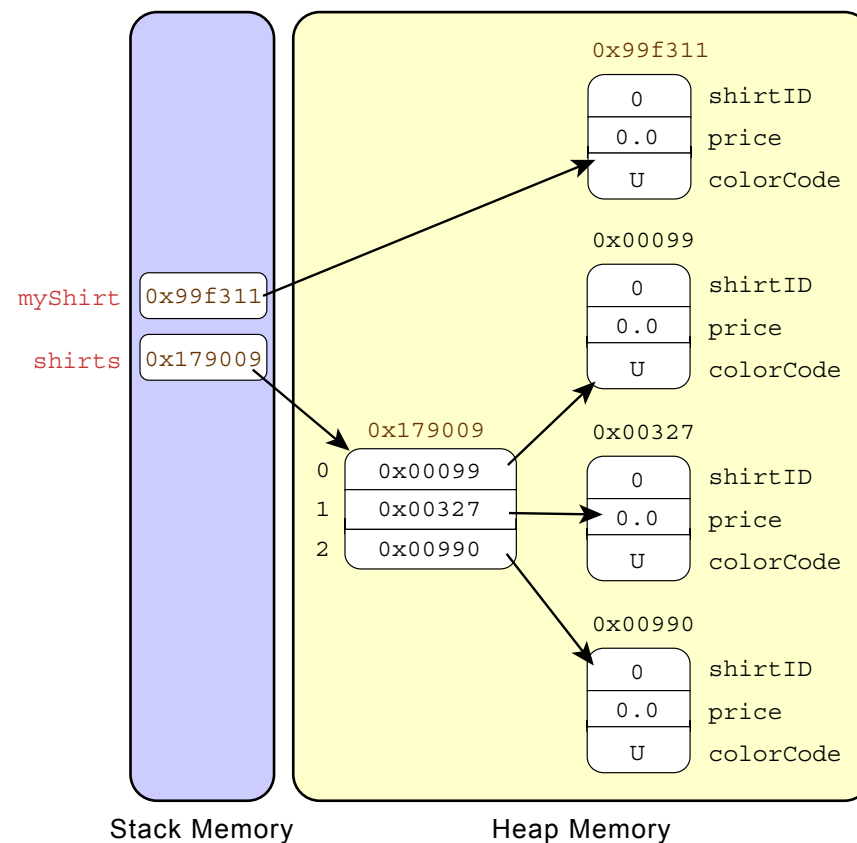
# Storing Primitive Variables and Arrays of Primitives in Memory





# Storing Arrays of References in Memory

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





# 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;  
}
```



## Enhanced For Loop

- The enhanced for loop can be used to make your loops more compact and easy to read
- This form of the for loop is designed for iteration through arrays
- Example:

```
class ExampleFor {  
    public static void main(String[] args) {  
        int[] numbers = {1,3,5,7,9,11,13,15,17,19};  
        int sum=0;  
        for (int item : numbers) {  
            sum = sum + item;  
        }  
        System.out.println("Sum is: " + sum);  
    }  
}
```



## 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  }  
9
```



# Converting `String` Arguments to Other Types

## Example:

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

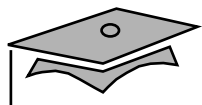


## The Varargs Feature

- You can create a method that can accept variable number of arguments.
- A method can have at most one parameter that is a vararg
- Vararg must be the last parameter taken by the method. It is denoted by an object type, a set of ellipses ( ... ), and the name of the variable. For example:

```
class VarMessage{  
    public static void showMessage(String... names) {  
        for (String list: names)  
            System.out.println(list);  
    }  
    public static void main (String args[]){  
        showMessage (args)  
    }  
}
```





# 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				



# **Module 11**

## **Implementing Inheritance**



## Objectives

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



## Relevance

- Inheritance refers to the passing down of something from one organism to some another organism. What are some physical characteristics that you have inherited?
- From whom did you inherit your characteristics?
- What class hierarchy are you from?
- Did you inherit characteristics from multiple classes?
- What does it mean if something is “abstract?”
- What do you think an abstract class is?



# Inheritance

Hat	Sock
ID price description colorCode R=Red, B=Blue, G=Green quantityInStock	ID price description colorCode R=Red, B=Blue, G=Green quantityInStock
calculateID() displayInformation()	calculateID() displayInformation()

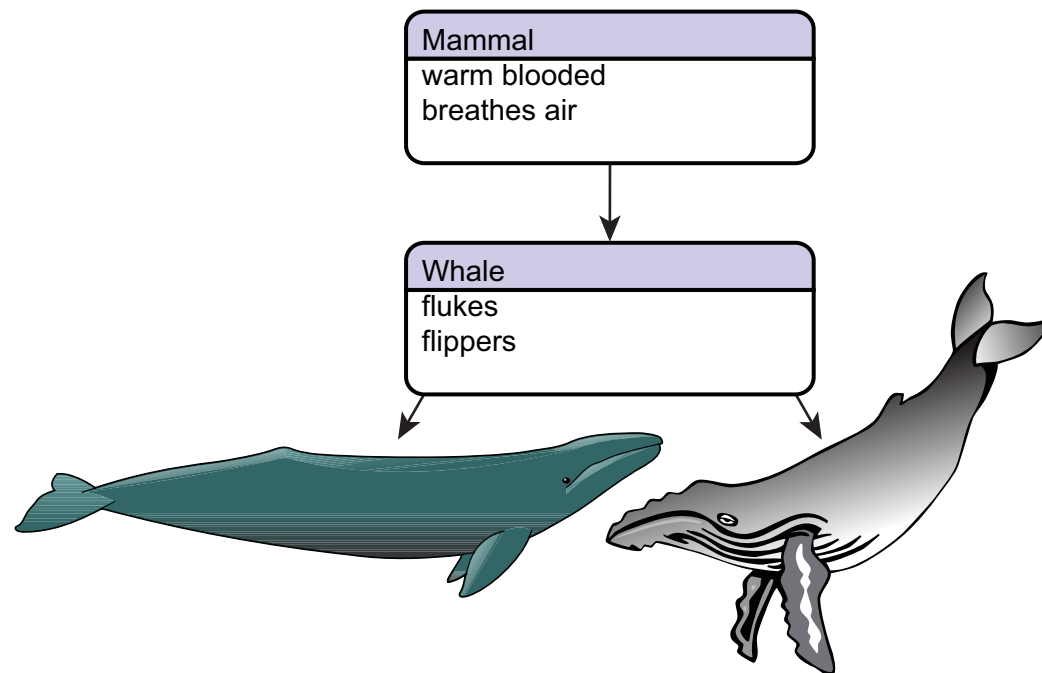




<b>Pant</b>	<b>Shirt</b>
ID price size gender M=Male, F=Female description colorCode B=Blue, T=Tan quantityInStock	ID price description colorCode R=Red, B=Blue, G=Green quantityInStock
calculateID() displayInformation()	calculateID() displayInformation()

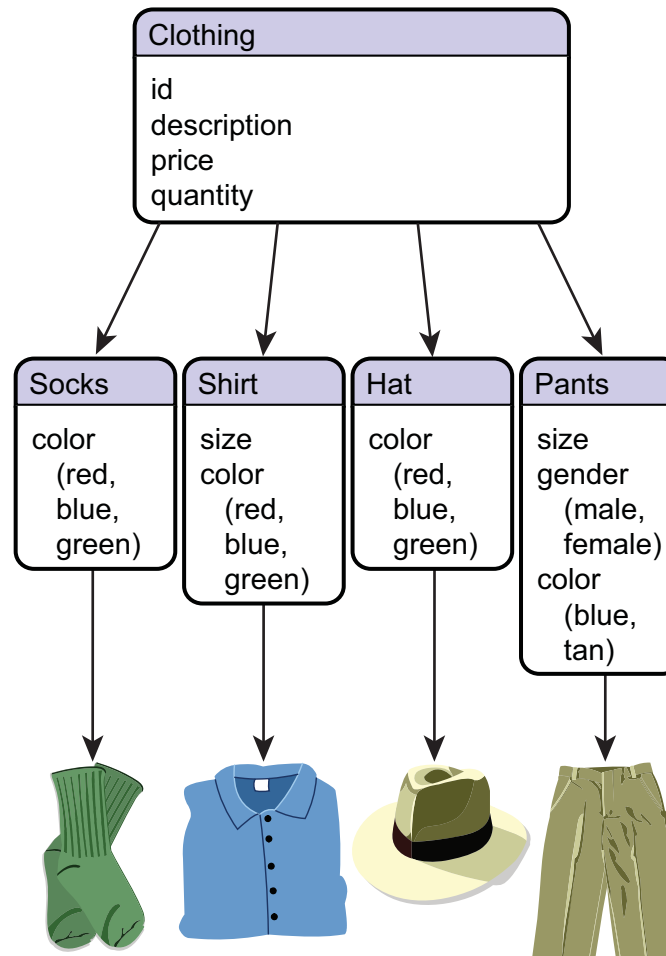


# Superclasses and Subclasses





# Testing Superclass and Subclass Relationships





# Modeling Superclasses and Subclasses

Hat:Clothing	Socks:Clothing
colorCode R=Red, B=Blue, G=Green	colorCode R=Red, B=Blue, G=Green
displayInformation()	displayInformation()

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



# Modeling Superclasses and Subclasses

Clothing
ID price description quantityInStock
calculateID()



## Declaring a Superclass

### Example:

```
1
2  public class Clothing {
3
4      private int ID = 0; // Default ID for all clothing
5      private String description = "-description required-"; // default
6
7      private double price = 0.0; // Default price for all clothing
8      private int quantityInStock = 0; // Default quantity for all clothing
9
10     private static int UNIQUE_ID=0; //Static member incremented in
    constructor to generate uniqueId
11
12     public Clothing() {
13         ID = UNIQUE_ID++;
14     }
15
16     public int getID() {
```



## Declaring a Superclass

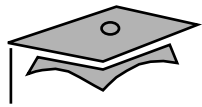
```
17  return ID;
18  }
19
20  public void setDescription(String d) {
21      description = d;
22  }
23
24  public String getDescription() {
25      return description;
26  }
27
28  public void setPrice(double p) {
29      price = p;
30  }
31
32  public double getPrice() {
33      return price;
34 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



## Declaring a Superclass

```
36 public void setQuantityInStock(int q) {  
37     quantityInStock = q;  
38 }  
39  
40 public int getQuantityInStock() {  
41     return quantityInStock;  
42 }  
43  
44 } // end of class  
45
```





# 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 displayInformation() {
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
17
```



# 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  
}
```