Course Booklet for Basic Java

Know how of Basic Java
By Emertxe

version1.0 (Jan 16, 2013)

Table of Contents

Chapter 1: Explaining Java Technology

1.1 Objective

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

- 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

1.2 History of Java-language

Since 1995, Java has changed our world . . . and our expectations..

Today, with technology such a part of our daily lives, we take it for granted that we can be connected and access applications and content anywhere, anytime. Because of Java, we expect digital devices to be smarter, more functional, and way more entertaining.

In the early 90s, extending the power of network computing to the activities of everyday life was a radical vision. In 1991, a small group of Sun engineers called the "Green Team" believed that the next wave in computing was the union of digital consumer devices and computers. Led by James Gosling, the team worked around the clock and created the programming language that would revolutionize our world – Java.

The Green Team demonstrated their new language with an interactive, handheld home-entertainment controller that was originally targeted at the digital cable television industry. Unfortunately, the concept was much too advanced for the them at the time. But it was just right for the Internet, which was just starting to take off. In 1995, the team announced that the Netscape Navigator Internet browser would incorporate Java technology.

Today, Java not only permeates the Internet, but also is the invisible force behind many of the applications and devices that power our day-to-day lives. From mobile phones to handheld devices, games and navigation systems to e-business solutions, Java is everywhere!

1.3 Key Concepts of the Java Programming Language

The Java programming language was designed to be:

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

1.3.1 Object-Oriented

1.3.2 Distributed

1.3.3 Simple

1.3.4 Multithreaded

1.3.5 Secure

1.3.6 Platform-independent

1.4 Java Technology Product Groups

Java technologies, such as the Java Virtual Machine, are included (in different forms) in three different groups of products, each designed to fulfill the needs of a particular target market:

1.4.1 J2SE - Java 2 Standard Edition

This edition is used to build desktop solutions, such as stand-alone applications, like your word processor or spread-sheet, and distributed applications. For ex: Applet.

1.4.2 J2EE - Java 2 Enterprise Edition

This edition is used to build powerful multi-tier enterprise applications such as e-commerce sites. Many popular sites are built using J2EE.

1.4.3 JavaTM Platform, Micro Edition (Java ME)

Creates applications for resource-constrained consumer devices. For example, you can use the Java ME SDK to create a game that runs on a cellular phone.

1.5 Product Life Cycle (PLC) Stages

The Product Life Cycle (PLC) represents an industry-accepted1 set of stages you should follow when developing any new product. There are seven stages in the PLC. These stages are:

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

1.5.1 Analysis

1.5.2 Design

1.5.3 Development

1.5.4 Testing

1.5.5 Implementation

1.5.6 Maintenance

1.5.7 End-of-life (EOL)

Chapter 2: Analyzing a Problem and Designing a Solution

2.1 Objectives

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

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

2.2 Analyzing a Problem Using Object-Oriented Analysis

2.2.1 Identifying a Problem Domain

2.2.2 Identifying Objects

2.2.3 Identifying Object Attributes and Operations

2.3 Designing Classes

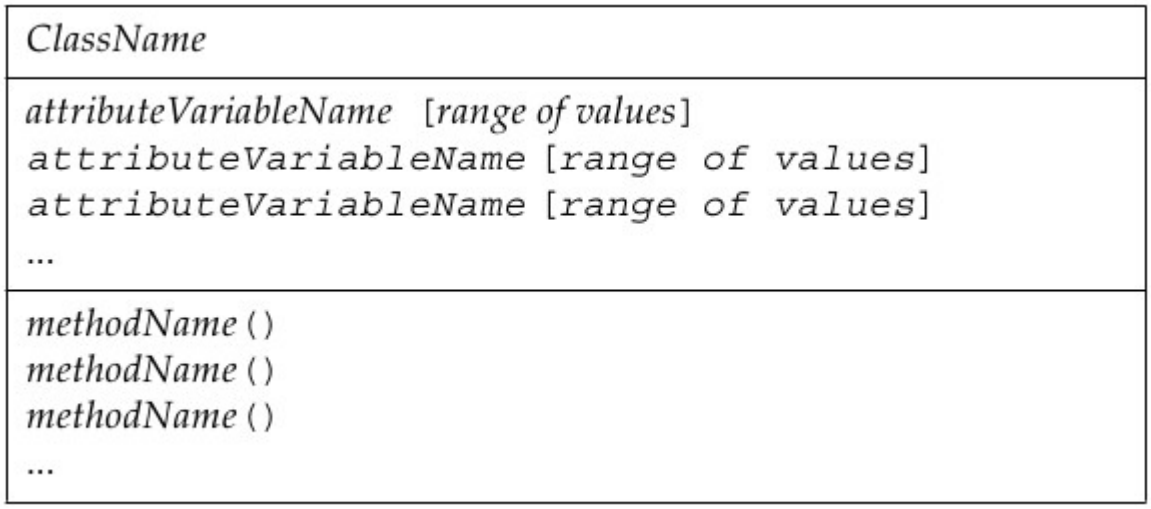
Identifying objects helps you design the class or blueprint for each of the objects in a system.

"a group whose members have certain attributes in common."

2.3.1 Modeling Classes

Classes are the blueprints for Objects. In Object-Oriented design terms, each Object (created shoe) is created using the class (blueprint) and it is called an instance of a class.

Classes are modeled in the Design stage where each class is represented as a Class diagram in the UML (Unified Modeling Language) notation.



Chapter 3: Developing and Testing a Java Technology Program

3.1 Objectives

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

- 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

3.2 Identifying the Components of a Class

3.2.1 Structuring Classes

Classes are composed of the Java technology code necessary to instantiate objects, such as Shirt objects. This course divides the code in a Java class file into four separate sections:

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

3.2.1.1 Class Declaration

You must declare a class for each class you designed for the problem domain. For each class, you must write a class declaration. The syntax for declaring a class is:

```
[modifiers] class class_identifier
```

3.2.1.2 Variable Declarations and Assignments

3.2.1.3 Comments

3.2.1.4 Methods

Methods follow the attribute variable declarations in a class. The syntax for methods is:

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

3.3 Creating and Using a Test Class

3.3.1 The main Method

3.4 Compiling and Executing (Testing) a Program

3.4.1 Compiling a Program

Compiling converts the class files you write into bytecode that can be executed by a Java Virtual Machine. Remember the rules for naming your Java source files. If a source file contains a public class, the source file must use the same name as the public class, with a .java extension. For example, the class Shirt must be saved in a file called Shirt.java.

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

   javac filename

For example:

javac Shirt.java

3.4.2 Executing (Testing) a Program

When you have successfully compiled your source code files, you can execute and test them using the Java Virtual Machine. To execute and test your program:

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

   java classname

For example:

java ShirtTest

Chapter 4: Declaring, Initializing, and Using Variables

4.1 Objectives

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

- 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
- Modify variable values using operators
- Use promotion and type casting

4.2 Identifying Variable Use and Syntax

You use variables for storing and retrieving data for your program.

4.2.1 Uses for Variables

4.2.2 Variable Declaration and Initialization

Attribute variable declarations and initialization follow the same general syntax. The syntax for attribute variables declaration and initialization is:

Local variables can be declared and initialized separately (on separate lines of code) or in a single line of code. The syntax for declaring a variable inside of a method is:

type identifier;

The syntax for initializing a variable inside of a method is:

identifier = value;

The syntax for declaring and initializing a variable inside of a method is:

type identifier [= value];

4.3 Describing Primitive Data Types

Many of the values in Java technology programs are stored as primitive data types.

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

4.3.1 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
`long`	64 bits	-2^{63} to $2^{63} -1$ (-9,223,372,036854,775,808 to 9,223,372,036,854,775,807, or 18,446,744,073,709,551,616 possible values)	2 -2036854775808L 1L

4.3.2 Floating Point Primitive Types

Type	Float Length	Examples of Allowed Literal Values
`float`	32 bits	99F -32745699.01F 4.2E6F (engineering notation for $4.2 * 10^6$)
`double`	64 bits	-1111 2.1E12 99970132745699.999

4.3.3 Textual Primitive Type

4.3.4 Logical Primitive Type

Computer programs must often make decisions. The result of a decision,whether the statement in the program is true or false, can be saved in boolean variables. Variables of type boolean can store only:

4.4 Declaring Variables and Assigning Values to Variables

4.4.1 Naming a Variable

4.4.2 Assigning a Value to a Variable

You can assign a value to a variable at the time the variable is declared or you can assign the variable later. To assign a value to a variable during declaration, add an equal sign after the declaration followed by the value to be assigned.

For example, the price attribute variable in the Shirt class could be assigned the value 12.99 as the price for a Shirt object.

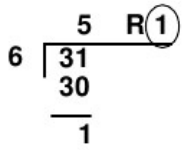
```
double price = 12.99;
```

4.4.3 Constants

4.5 Using Arithmetic Operators to Modify Values

Java has many operators. Here we will look at the arithmetic, increment / decrement operators. In subsequent modules we will look into some of the other operators that are available to us in Java.

4.5.1 Standard Mathematical Operators

Purpose	Operator	Example	Comments
Addition	+	`sum = num1 + num2;` If `num1` is 10 and `num2` is 2, `sum` is 12.	
Subtraction	–	`diff = num1 - num2;` If `num1` is 10 and `num2` is 2, `diff` is 8.	
Multiplication	*	`prod = num1 * num2;` If `num1` is 10 and `num2` is 2, `prod` is 20.	
Division	/	`quot = num1 / num2;` If `num1` is 31 and `num2` is 6, `quot` is 5	Division returns an integer value (with no remainder).
Remainder	%	`mod = num1 % num2;` If `num1` is 31 and `num2` is 6, `mod` is 1.	Remainder finds the remainder of the first number divided by the second number.  Remainder always gives an answer with the same sign as the first operand.

4.5.2 Increment and Decrement Operators (++ and --)

Operator	Purpose	Example	Notes

++	Pre-increment (*++variable*)	`int i = 6;`
		`int j = ++i;`
		`i is 7, j is 7`	

	Post-increment (*variable++*)	`int i = 6;`
		`int j = i++;`
		`i is 7, j is 6`
			The value of `i` is assigned to `j` before `i` is incremented. Therefore, `j` is assigned `6`.

4.5.3 Operator Precedence

To make mathematical operations consistent, the Java programming language follows the standard mathematical rules for operator precedence. Operators are processed in the following order:

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

If standard mathematical operators of the same precedence appear successively in a statement, the operators are evaluated from left to right.

Chapter 5: Creating and Using Objects

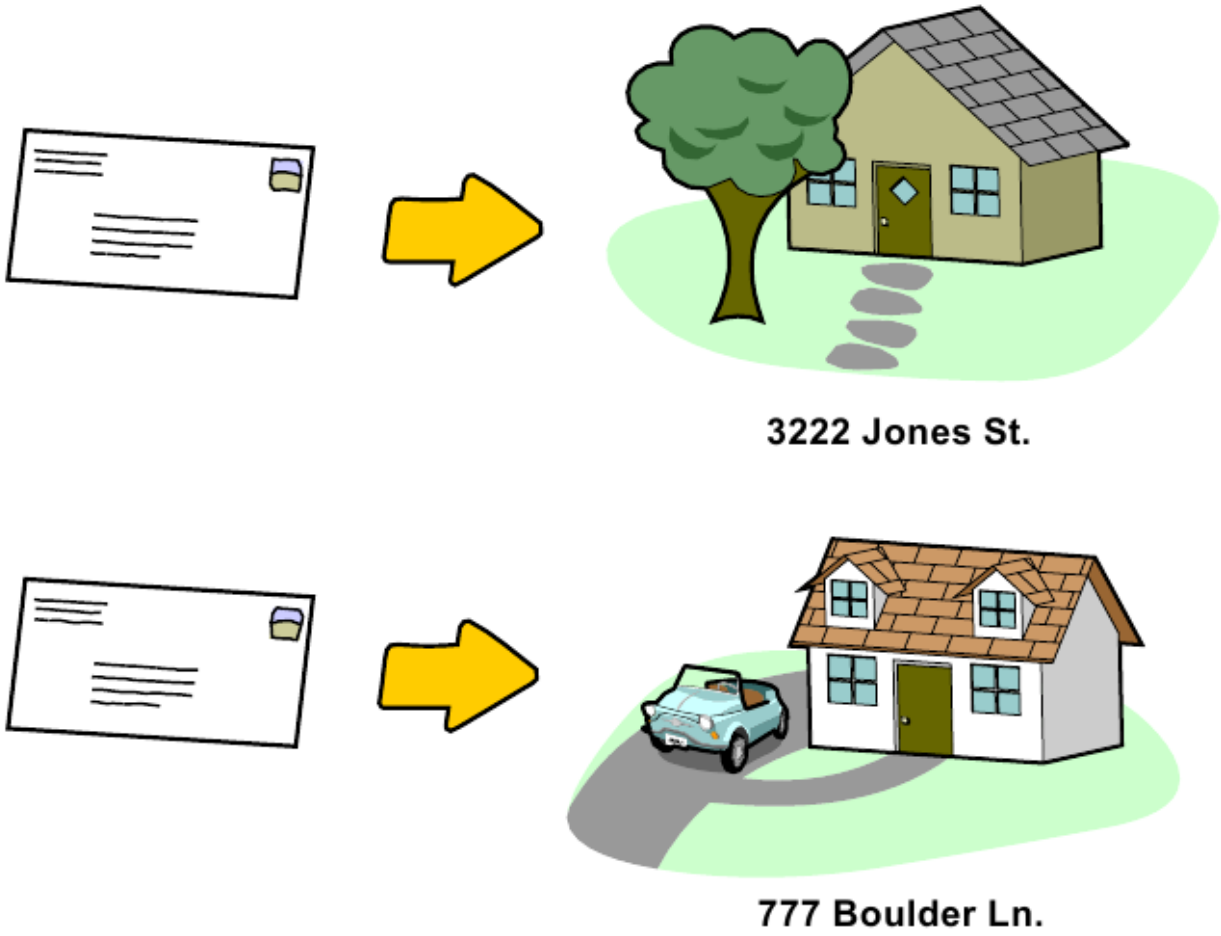
5.1 Objectives

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

- 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 SE class library specification to learn about other classes in this API

5.2 Declaring Object References, Instantiating Objects, and Initializing Object References

Object reference variables are variables containing an address to an object in memory. A letter is like a reference variable because it has an address pointing to a particular building object.



5.2.1 Declaring Object Reference Variables

To declare an object reference variable, state the class that you want to create an object from, then select the name you want to use to refer to the object.

The syntax for declaring object reference variables is:

```
Classname identifier;
```

5.2.2 Instantiating an Object

After you declare the object reference, you can create the object that you refer to.

The syntax for instantiating an object is:

```
new Classname();
```

5.2.3 Initializing Object Reference Variables

The final step in creating an object reference variable is to initialize the object reference variable by assigning the newly created object to the object reference variable. Just as with variable assignment or initialization, you do this with the equal sign (=).

The syntax for initializing an object to an object reference variable is:

```
identifier = new Classname();
```

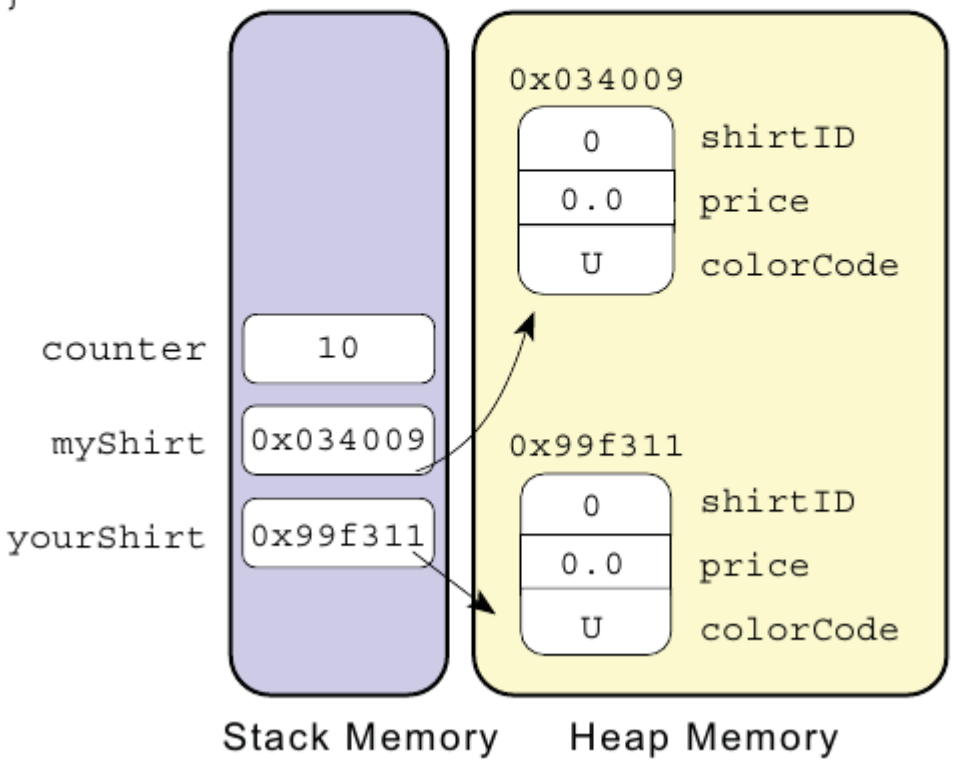
5.2.4 Using an Object Reference Variable to Manipulate Data

You use the dot (.) operator with an object reference to manipulate the values or to invoke the methods of a specific object.

5.2.5 Storing Object Reference Variables in Memory

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

  int counter;
  counter = 10;
  Shirt myShirt = new Shirt ( );
}
```



5.3 Using the String Class

The String class provides you with the ability to store a sequence of characters. You will use the String class frequently throughout your programs. Therefore, it is important to understand some of the special characteristics of strings in the Java programming language.

5.3.1 Creating a String Object With the new Keyword

There are a number of ways to create and initialize a String object. One way is to use the new keyword to create a String object and, at the same time, define the string with which to initialize that object:

```
String myName = new String("karthik");
```

5.3.2 Creating a String Object Without the new Keyword

String types are unique because they are the only class that allows you to build objects without using the new keyword. For example:

```
String myName = "karthik";
```

5.3.3 Storing String Objects in Memory

5.4 Investigating the Java Class Libraries

All of the Java technology SDKs contain a series of pre-written classes for you to use in your programs. These Java technology class libraries are documented in the class library specification for the version of the SDK that you are using.

5.4.1 Java Class Library Specification

Chapter 6: Using Operators and Decision Constructs

6.1 Objectives

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

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

6.2 Using Relational and Conditional Operators

6.2.1 Relational Operators

Relational operators compare two values to determine their relationship.

Condition	Operator	Example
Is equal to (or "is the same as")	==	int i=1; (i == 1)
Is not equal to (or "is not the same as")	!=	int i=2; (i != 1)
Is less than	<	int i=0; (i < 1)
Is less than or equal to	<=	int i=1; (i <= 1)
Is greater than	>	int i=2; (i > 1)
Is greater than or equal to	>=	int i=1; (i >= 1)

6.2.2 Conditional Operators

You will also need the ability to make a single decision based on more than one condition. Under such circumstances, you can use conditional operators to evaluate complex conditions as a whole.

Operation	Operator	Example

If one condition AND another condition	&&	`int i = 2;`
		`int j = 8;`
		`((i < 1) && (j > 6))`

If either one condition OR another condition	||	`int i = 2;`
		`int j = 8;`
		`((i < 1) || (j > 10))`

NOT	!	`int i = 2;` `(!(i < 3))`

6.3 Creating if and if/else Constructs

An if statement, or an if construct, executes a block of code if an expression is true.

6.3.1 The if Construct

There are a few variations on the basic if construct. However, the simplest is:

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

6.3.2 Nested if Statements

6.3.3 The if/else Construct

```
 if (boolean_expression)
 {
   code_block1;
  } // end of if construct
   else
   {
    code_block2;
   } // end of else construct
    // program continues here
```

6.3.4 Chaining if/else Constructs

```
if (boolean_expression)
{
   code_block1;
} // end of if construct
else if (boolean_expression){
    code_block2;
```

```
} // end of else if construct
else {
   code_block3;
}
// program continues here
```

6.4 Using the switch Construct

The switch construct helps you avoid confusing code because it simplifies the organization of the various branches of code that can be executed.

Syntax:

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

Chapter 7: Using Loop Constructs

7.1 Objectives

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

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

7.2 Creating while Loops

A while loop iterates through a code block while an expression yields a value of true.

The syntax for a while loop is:

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

7.2.1 Nested while Loops

7.3 Developing a for Loop

The for loop allows your program to loop through a sequence of statements for a predetermined number of times.

Syntax:

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

7.3.1 Nested for Loops

7.4 Coding a do/while Loop

The do/while loop is a one-to-many iterative loop: the condition is at the bottom of the loop and is processed after the body. The body of the loop is therefore processed at least once.

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

7.4.1 Nested do/while Loops

Chapter 8: Developing and Using Methods

8.1 Objectives

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

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

8.2 Creating and Invoking Methods

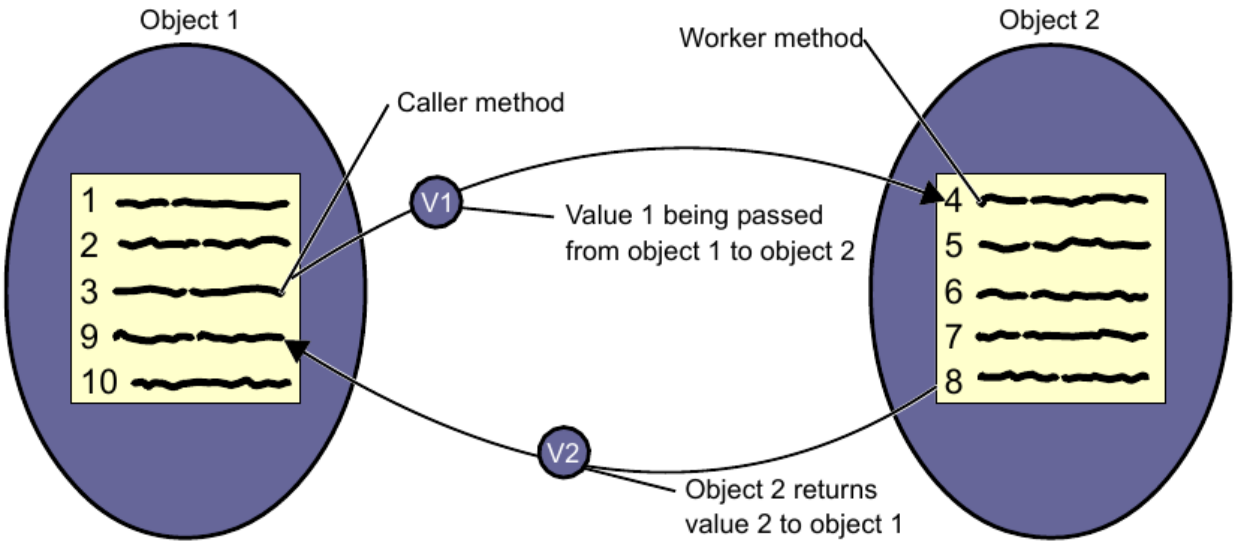
Most of the code you write for a class is contained within one or more methods. Methods let you divide the work that your program does into separate logical tasks or behaviors.

The syntax of all method declarations is as follows:

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

8.3 Passing Arguments and Returning Values

Methods can be invoked by a calling method with a list of arguments (variables or values to be used by the worker method). Additionally, methods can return a value to the calling method that can be used in the calling method.



8.4 Creating static Methods and Variables

Methods and variables that are unique to an instance are called instance methods and instance variables.

You have also been using methods that do not require object instantiation, such as the main method. These are called class methods or static methods; you can invoke them without creating an object first.

8.4.1 Declaring static Methods

Static methods are declared using the static keyword.

For ex:

```
static Properties getProperties()
```

8.4.2 Invoking static Methods

Because static or class methods are not part of any object instance (just the class), you should not use an object reference variable to invoke them. Instead, use the class name.

The syntax for invoking a static method is:

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

8.4.3 Declaring static Variables

8.4.4 Accessing static Variables

You should use the class name to access a static variable.

The syntax for invoking a static variable is:

Classname.variable;

8.5 Using Method Overloading

In the Java programming language, there can be several methods in a class that have the same name but different arguments (different method signatures). This concept is called method overloading.

Chapter 9: Implementing Encapsulation and Constructors

9.1 Objectives

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

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

9.2 Using Encapsulation

In object-oriented programming, the term encapsulation refers to the hiding of data within a class (a safe “capsule”) and making it available only through certain methods. Encapsulation is important because it makes it easier for other programmers to use your classes and protects certain data within a class from being modified inappropriately.

9.2.1 Visibility Modifiers

9.2.2 The public Modifier

9.2.3 The private Modifier

9.2.4 Get and Set Methods

9.3 Describing Variable Scope

Variables declared within a method, constructor, or other block of code, cannot be used throughout a class. The scope of a variable refers to the extent that a variable can be used within a program.

9.3.1 How Instance Variables and Local Variables Appear in Memory

Instance or attribute variables are stored in a different part of memory from where local variables are stored.

9.4 Creating Constructors

Constructors are method-like structures that are invoked automatically when you instantiate an object. Constructors are usually used to initialize values in an object.

The syntax for a constructor is similar to the syntax for a method declaration:

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

9.4.1 Default Constructor

9.4.2 Overloading Constructors

Chapter 10: Creating and Using Arrays

10.1 Objectives

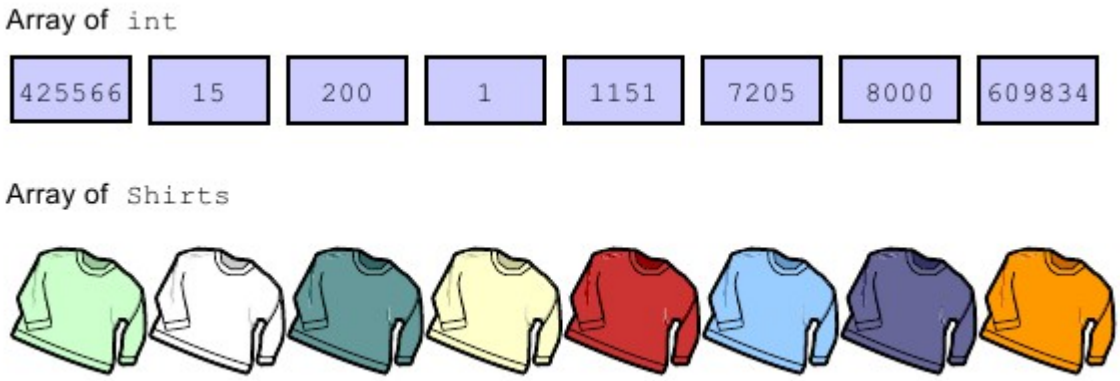
Upon completion of this module, you should be able to:

- 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

10.2 Creating One-Dimensional Arrays

The Java programming language allows you to group multiple values of the same type (lists) using one-dimensional arrays. Arrays are useful when you have related pieces of data (such as the ages for several people), but you do not want to create separate variables to hold each piece of data.

The following figure illustrates several one-dimensional arrays.



10.2.1 Declaring a One-Dimensional Array

Arrays are handled by an implicit Array object (which is not available in the Java API, but is available in your code). Just as with any object, you must declare an object reference to the array, instantiate an Array object, and then initialize the Array object before you can use it.

The syntax used to declare a one-dimensional array is:

type [] array_identifier;

10.2.2 Instantiating a One-Dimensional Array

Before you can initialize an array, you must instantiate an Array object large enough to hold all of the values in the array. Instantiate an array by defining the number of elements in the array. The syntax used to instantiate an Array object is:

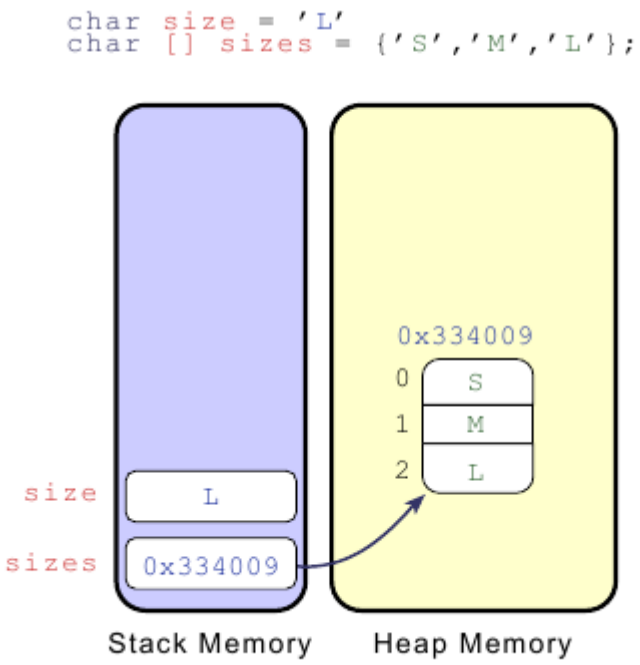
array_identifier = new type [length];

10.2.3 Initializing a One-Dimensional Array

You can fill the contents of an array after you have created the array.
The syntax for setting the values in an array is:
array_identifier[index] = value;

10.2.4 Storing One-Dimensional Arrays in Memory

Arrays are objects referred to by an object reference variable.



10.3 Setting Array Values Using the length Attribute and a Loop

10.3.1 The length Attribute

All Array objects have a length attribute variable that contains the length of the array.

10.3.2 Setting Array Values Using a Loop

10.3.3 Using the args Array in the main Method

public static void main (String args[]);

When you pass strings to your program on the command line, the strings are put in the args array. To use these strings, you must extract them from the args array and, optionally, convert them to their proper type.

10.3.4 The varargs Feature

The varargs or the variable arguments feature allows you to create a method that can accept a variable number of arguments

10.4 Describing Two-Dimensional Arrays

A two-dimensional array (an array of arrays) is similar to a spreadsheet with multiple columns (each column represents one array or list of items) and multiple rows.

10.4.1 Declaring a Two-Dimensional Array

Two dimensional arrays require an additional set of square brackets. The process of creating and using two-dimensional arrays is otherwise the same as with one-dimensional arrays.

The syntax for declaring a two dimensional array is:

type [][] array_identifier;

10.4.2 Instantiating a Two-Dimensional Array

The syntax for instantiating a two-dimensional array is:

array_identifier = new type [number_of_arrays] [length];

Chapter 11: Implementing Inheritance

11.1 Objectives

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

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

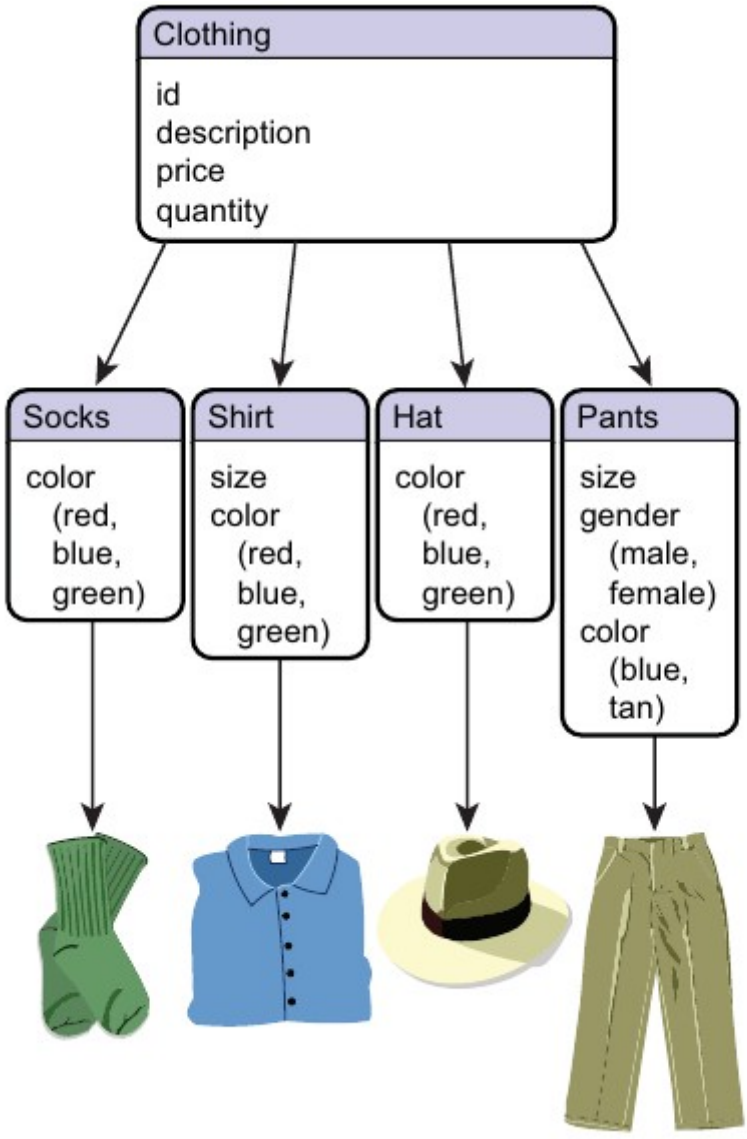
11.2 Inheritance

Inheritance enables programmers to put common members (variables and methods) in one class and have other classes inherit these common members from this new class.

11.2.1 Superclasses and Subclasses

The class containing members common to several other classes is called the superclass or the parent class. The classes that inherit from, or extend, the superclass, are called subclasses or child classes.

The following figure illustrates this new class hierarchy.



11.2.2 Declaring a Superclass

11.2.3 *Declaring a Subclass*

Use the extends keyword to indicate that a class inherits from another class. To declare that a class is a subclass of another class, use the following syntax in your class declaration:

[class_modifier] class class_identifier extends superclass_identifier

11.3 Abstraction

Abstraction refers to creating classes that are very general and do not contain methods with a particular implementation or method body code.

11.4 Classes in the Java API

Classes in the Java programming language are grouped into packages depending on their functionality.

For example, all classes related to the core Java programming language are in the java.lang package, which contains classes that are fundamental to the Java programming language, such as String, Math, and Integer.

11.4.1 Implicitly Available Classes

Classes in the java.lang package can be implicitly referred to in all programs.

11.4.2 Importing and Qualifying Classes

- You can import the class using an import statement.
- You can refer to the class using a fully qualified class name.

11.4.2.1 The import Statement

You can use import statements to make your code clearer to the reader because import statements shorten the code that you must enter to explicitly reference a Java API class.

There are two forms of the import statement:

import package_name.class_name;

import package_name.*;

11.4.2.2 Specifying the Fully Qualified Name

Instead of specifying the java.awt package, you could refer to the Button class as java.awt.Button throughout the program. For example, the following is a class declaration that uses the fully qualified name when specifying its superclass.

The syntax for the fully qualified name is as follows.

package_name.class_name