|  |  |
| --- | --- |
| **Topic** | **Object Oriented Programming In Java** |
| **Learning Outcomes** | The term object-oriented programming means many different things. But at its heart, object-oriented programming is a type of computer programming based on the premise that all programs are essentially computer-based simulations of real-world objects or abstract concepts. At the end of the lesson, you will be able to:1. Understand What is class and object
2. Differentiate object from a classes
3. Create object and class
4. Performed instantiation.

**In This Chapter you will learn*** What object-oriented programming is
* Objects and classes
* To Investigating inheritance and interfaces
* To Designing programs with objects
 |
| **References** | Lemay, L., Perkins, C. L. (1996), Teach Yourself JAVA in 21 days,  Indianapolis, Indiana, Sams.net.Doug Lowe, Java® All-in-One For Dummies®, 4th Edition), New Jersey,  John Wiley & Sons, Inc.Barry Burd Ph.D., Java® For Dummies®, 7th Edition, New Jersey,  John Wiley & Sons, Inc.David J. Eck, Introduction to Programming Using Java Version 5.0, December 2006 Geneva, NY, Hobart and William Smith CollegesJester Lhee I. Pandio, Computer Programming 1, Global Port Taguig City, STI College |

**Discussion**

**Object-oriented programming languages**

Object-oriented programming is modelled on how, in the real world, objects are often made up of many kinds of smaller objects. This capability of combining objects, however, is only one very general aspect of object-oriented programming. Object-oriented programming provides several other concepts and features to make creating and using objects easier and more flexible, and the most important of these features is that of classes.

 **What is a Class?**

* Class is code that defines the behavior of a Java programming element called an object.
* A Class is a blue print of an object.
* A class is a template for multiple objects with similar features. Classes embody all the features of a particular set of objects.
* A class is a group of programming statements that specifies what the object is capable of doing.
* It also specify the state and behavior of the object

 **What is an Object?**

**An object**

* Is anything that is visible or tangible and is relatively stable in form, a thing, person, or matter to which thought or action is directed.
* Is an abstract data type that can include multiple properties and methods and may even contain other **objects.**
* An **object** is an entity that has both ***state*** and ***behavior***. The state of an object consists of any data that the object might be keeping track of, and the behavior consists of actions that the object can perform. The behaviors are represented in the class by one or more methods that can be called on to perform actions.

An ***instance*** of a class is another word for an actual object. If classes are an abstract representation of an object, an instance is its concrete representation.

**Characteristics of an Object**

1. **Object has an Identity**

Every object in an object-oriented program has an identity. In other words, every occurrence of a particular type of object — called an instance — can be distinguished from every other occurrence of the same type of object, as well as from objects of other types.

1. **Objects have type**

In Java, classes define types. Therefore, when you create an object from a type, you are saying that the object is of the type specified by the class.

The following example statement creates an object of type Invoice:

***Invoice*** i = new Invoice(); // our object i is of type Invoice.

In this case, the identity of this object (that is, its address in memory) is assigned to the variable i, which the compiler knows can hold references to objects of type Invoice.

1. **Objects have state**

The combination of the values for all the attributes of an object is called the object’s state.

Example

import java.awt.\*;

public class Ball {

 Color color;

 float size;

 String texture;

 }

import java.awt.\*;

public class ObjectState {

 static Ball ball = new Ball();

 public static void main(String[] args){

 ball.color = Color.BLUE;

 ball.size = 6.5f;

 ball.texture = "Striped";

 System.out.println("Color of the Ball : " + ball.color);

 System.out.println("Size of the Ball : " + ball.size);

 System.out.println("Texture of the Ball : " + ball.texture);

 }

}

In here the values assigned to the object ball (Color.BLUE, 6.5f, |Striped”) is its state.

**Objects have behavior**

Means that they can do things. Like state, the specific behavior of an object depends on its type, but unlike state, behavior is not different for each instance of a type.

Modifying our previous example:

 import java.awt.\*;

public class Ball {

 Color color;

 float size;

 String texture;

 public void setColor(Color col){

 color = col;

 }

public Color getColor(){

 return color;

 }

 }

 The above example shows the behavior of the object ball namely setColor() and getColor()

 For a more specific example a ball can bounce. We can also create a behavior for our ball object such as bounce similar to this:

public void Bounce(){

 String bounceSpeed = “Fast”;

 int bounceHeight = 5;

}

The difference between a class and an object is similar to the difference between a blueprint and a house. A blueprint is a plan for a house. A house is an implementation of a blueprint. One set of blueprints can be used to build many houses. Likewise, a class is a plan for an object, and an object is — in Java terms — an instance of a class. You can use a single class to create more than one object.

 **Creating object from a class**

In Java, you can create an object from a class in several ways. The most straightforward way is to create a variable that provides a name you can use to refer to the object, use the new keyword to create an instance of the class, and then assign the resulting object to the variable. The general form of a statement that does that bit of magic looks like this:

ClassName variableName = new ClassName();

Why do you have to list the class name twice? The first time, you’re providing a type for the variable. In other words, you’re saying that the variable you’re creating here can be used to hold objects created from the ClassName class. The second time you list the class name, you’re creating an object from the class. The new keyword tells Java to create an object, and the class name provides the name of the class to use to create the object. The equal sign (=) is an assignment operator. It simply says to take the object created by the new keyword and assign it to the variable. Thus, this statement actually does three things:

* It creates a variable named varaibleName that can be used to hold objects created from the ClassName class. At this point, no object has been created — just a variable that can be used to store objects
* It creates a new object in memory from the ClassName class.
* It assigns this newly created object to the variableName variable. That way, you can use the variableName variable to refer to the object that was created.

Example

This is the blue print of a greeter class. Here we only specify the behavior of the greeter class which is sayHello.

 public class Greeter {

 public Greeter() {

 }

 public void sayHello(){

 System.out.println("Hello! Welcome to Java Programming!");

 }

}

// in order to implement the greeter class you need to call it inside another class and create an object. The process of creating an object is known as ***instantiation.***

public class Hello {

 public static void main(String args[] ){

Greeter greeter = new Greeter(); //*Instantiation*

 greeter.sayHello():

 }

}

**Note:**

 In including ***new Greeter()*** in the instantiation you are actually calling the statements inside the public Greeter() method known as the ***constructor***.

**Challenge:**

Write any statement in the constructor and see what happen.

**Another version of the greeter class**

import javax.swing.JOptionPane;

public class Greeter{

public void sayHello() {

JOptionPane.showMessageDialog(null,"Hello, World!", "Greeter", JOptionPane.INFORMATION\_MESSAGE);

}

}

Note:

The important point to realize here is that the Hello class doesn’t have to be changed to use this new version of the Greeter class. Instead, all you have to do is replace the old Greeter class with the new one, recompile the Greeter class, and the Hello class won’t know the difference. That’s one of the main benefits of object-oriented programming.

Importing Java API Classes

 You may have noticed that the Greeter class in our revised verwion includes this statement:

***import javax.swing.JOptionPane;***

The purpose of the import statement is to let the compiler know that the program is using a class that’s defined by the Java API called JOptionPane.

The import statement is used to call the attention of the compiler, that our class will use an API (short for Application Programming Interface) which contains several built-in classes, In this case the JOptionPane class.

Because the Java API contains literally thousands of classes, some form of organization is needed to make the classes easier to access. Java does this by grouping classes into manageable groups called packages. In the previous example, the package that contains the JOptionPane class is named javax.swing.

Import statements are never required. But if you don’t use import statements to import the API classes your program uses, you must fully qualify the names of the classes when you use them by listing the package name in front of the class name.

Example

 javax.swing.JOptionPane.showMessageDialog(null, "Hello, World!", "Greeter", javax.swing.JOptionPane.INFORMATION\_MESSAGE);

Rules for working with import statements:

* Import statements must appear at the beginning of the class file, before any class declarations
* You can include as many import statements as are necessary to import all the classes used by your program.
* You can import all the classes in a particular package by listing the package name followed by an asterisk wildcard, like this:import javax.swing.\*;
* Because many programs use the classes that are contained in the java.lang package, you don’t have to import that package. Instead, those classes are automatically available to all programs. The System class is defined in the java.lang package. As a result, you don’t have to provide an import statement to use this class