Lab Manual

**CPCS203**

**Programming II**

**(Object-oriented)**

**1432/1433H**

**Lab - 4**

 **Learning Procedure**

1. Stage **J** (**Journey inside-out the concept**)
2. Stage **a1** (**apply the learned**)
3. Stage **v** (**verify the accuracy**)
4. Stage **a2** (**assess your work**)

**Laboratory 4:**

**Statement Purpose:**

This lab will give you how to **Defining Your Own Class** in java.

**Activity Outcomes:**

This lab teaches you the following topics:

* Be able to declare a new class
* Be able to write a constructor
* Be able to write instance methods that return a value
* Be able to write instance methods that take arguments
* Be able to instantiate an object
* Passing Objects
* Be able to use calls to instance methods to access and change the state of an object

**Instructor Note:**

As pre-lab activity, read Chapter 4 from the book (An Introduction to Object-Oriented Programming with Java, 4th Edition by C. THOMAS WU (Book’s website [www.mhhe.com/wu](http://www.mhhe.com/wu)), and also as given by your theory instructor.

**Names I.D.**

1. **.……………..………………………………. ………………………………**
2. **..…………………………………………….. ………………………………**
3. **.……………………………………………... ………………………………**
4. **.…………………………………………….. ..…………………………….**
5. **Stage J (Journey)**

**Classes and Objects**

So far we have been using only standard classes such as System, String, and others when we wrote programs. For a basic program, that is fine. However, we need to learn how to write programs using our own classes (in addition to using the standard classes) when the programs become large and complex. In this chapter, we learn the basics of how to define our own classes.

**Example 1:**

**Drive a Car Scenario**

Examine the code for the **Car, Driver and ServiceStation** classes. The **CarDemo.java** file creates instances of each of these classes and simulates a driver (named Ahmad) driving a car (a Rolls). The Rolls starts out clean but with an empty tank (see the Car constructor) so we must fill its tank before Ahmad can drive it.

Compile and run the Demo program and observe the results. Notice, in particular, that the Rolls becomes dirty after Ahmad drives it.

Find the code of the class **CarWash.java** below that contains most of the implementation of the CarWash class. Open this file and complete the implementation. In the **CarDemo.java** program, make a suitable instance of a CarWash object and finish the program by washing the Rolls and filling it with gas (at the appropriate place at the end of your CarDemo code indicated by comments).

Compile and run your CarDemo program and observe your results.

When you are ready, call lab Instructor and show him your completed code for CarDemo.java and the output of your program.

Add the following statement at the end of the main() method of CarDemo class

System.out.println ("Car tank is " + rolls.tank);

and recompile this class. **Did it compile? Why or why not?**

1. Create a java class and names it **Car** with the underneath definition of the **Car** class.

/\* This Class provides car objects that have either tanks "full" or "empty" and are either "clean" or "dirty” Car.java \*/

public class Car {

//data members

// the tank is either "full" or "empty"

private String tank;

// the body is either "clean" or "dirty"

private String body;

//default Constructors

public Car() {

tank="Empty:";

body="Clean:"; }

// public methods:

// Changes the condition of the car body

public void setBody(String state){

body=state; }

// Gas level in the car's tank (Empty / Full)

public void setTank(String level){

tank=level; }

// return Status of the Car

public String getStatus() {

return ("Tank is " + tank + " , body is "+ body); }

}

1. Now Create another java class and names it **Drive** with the underneath definition of the **Drive** class. Note This Class contains only one method **drive** (with **Car Object** as its parameter)

// Drive.java

// this class provides drive Objects

public class Drive

{

// Drive Car until it is out of gas and it gets dirty

public void drive (Car c)

{

c.setTank("Empty");

c.setBody("Dirty");

} }

1. Now Create another java class and names it **ServiceStation** with the underneath definition of the **ServiceStation** class. Note This Class contains only one method **fill** (with **Car Object** as its parameter)

// ServiceStation.java

// This Class provides a service station for cars

public class ServiceStation

{

// fills car tank with Gas

// Passing Object c of Car class

public void fill(Car c)

{

c.setTank("Full");

}

}

1. Now Create another java class and names it **CarWash** with the underneath definition of the **CarWash** class. Note This Class contains only one method **wash** (with **Car Object** as its parameter)

// CarWash.java

// This class provides a car wash for a Car

public class CarWash

{ // Wash My Car

// Passing Object c of Car class

public void wash(Car c)

{

**//complete the implementation here**

...

}

}

1. Now lastly **Create java main class** and names it **CarDemo** with the underneath definition of the **CarDemo** class.

// CarDemo.java // Main Class

public class CarDemo {

public static void main(String[] args) {

// Creating Objects of classes Car , Drive, ServiceStation, CarWash

Car rolls = new Car();

Drive Ahmad= new Drive();

ServiceStation gulf= new ServiceStation();

CarWash sasco= new CarWash();

// Initial Status of the Car before drive

System.out.println("Rolls: "+ rolls.getStatus());

System.out.println("Gas up the Rolls:");

gulf.fill(rolls);

System.out.println("Rolls: "+ rolls.getStatus());

System.out.println("I Start Driving The Rolls :");

Ahmad.drive(rolls);

System.out.println("Rolls: "+ rolls.getStatus());

System.out.println("Clean and Gas Up the Rolls:");

sasco.wash(rolls);

gulf.fill(rolls);

System.out.println("Rolls: "+ rolls.getStatus()); } }

1. **Stage a1 (apply)**

**Activity 1:**

**Television Class Scenario**

Everyone is familiar with a television. It is the object we are going to create in this lab.

First we need a blueprint. All manufacturers have the same basic elements in the televisions they produce as well as many options. We are going to work with a few basic elements that are common to all televisions. Think about a television in general. **It has a brand name (i.e. it is made by a specific manufacturer). The television screen has a specific size. It has some basic controls. There is a control to turn the power on and off. There is a control to change the channel. There is also a control for the volume.** At any point in time, the television’s state can be described by how these controls are set. We will write the television class. Each object that is created from the television class must be able to hold information about that instance of a television in fields. So a television object will have the following attributes:

• **manufacturer**. The manufacturer attribute will hold the brand name. This cannot change once the television is created, so will be a named constant.

• **screenSize.** The screenSize attribute will hold the size of the television screen. This cannot change once the television has been created so will be a named constant.

• **powerOn.** The powerOn attribute will hold the value true if the power is on, and false if the power is off.

• **channel.** The channel attribute will hold the value of the station that the television is showing.

• **volume.** The volume attribute will hold a number value representing the loudness (0 being no sound).

These attributes become **fields** in our class.

The television object will also be able to control the state of its attributes. These controls become **methods** in our class.

• **setChannel.** The setChannel method will store the desired station in the channel field.

• **power.** The power method will toggle the power between on and off, changing the value stored in the powerOn field from true to false or from false to true.

• **increaseVolume.** The increaseVolume method will increase the value

stored in the volume field by 1.

• **decreaseVolume.** The decreaseVolume method will decrease the value

stored in the volume field by 1.

• **getChannel.** The getChannel method will return the value stored in the

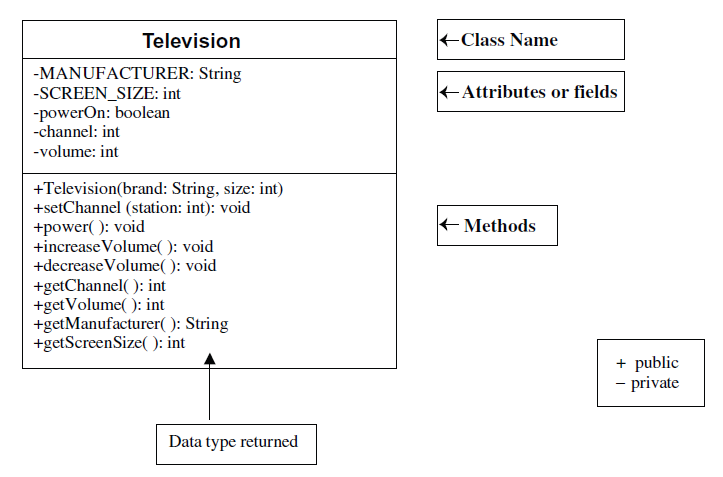
channel field.

• **getVolume.** The getVolume method will return the value stored in the volume field.

• **getManufacturer.** The getManufacturer method will return the constant value stored in the MANUFACTURER field.

• **getScreenSize.** The getScreenSize method will return the constant value stored in the SCREEN\_SIZE field.

We will also need a constructor method that will be used to create an instance of a Television. These ideas can be brought together to form a UML (Unified Modeling Language) diagram for this class as shown below.

****

**Task 1:**

**Creating a New Class**

1. In a new file, create a class definition called Television.
2. Put a program header (comments/documentation) at the top of the file

// The purpose of this class is to model a television

// Your name and today’s date

1. Declare the 2 constant fields listed in the UML diagram.
2. Declare the 3 remaining fields listed in the UML diagram.
3. Write a comment for each field indicating what it represents.
4. Save this file as *Television.java*.
5. Compile and debug. Do not run.

**Task 2:**

**Writing a Constructor**

1. Create a constructor definition that has two parameters, a manufacturer’s brand and a screen size. These parameters will bring in information
2. Inside the constructor, assign the values taken in from the parameters to the corresponding fields.
3. Initialize the powerOn field to false (power is off), the volume to 20, and the channel to 2.
4. Write comments describing the purpose of the constructor above the method header.
5. Compile and debug. Do not run.

**Task 3:**

**Methods**

1. Define methods called **getVolume, getChannel, getManufacturer, and getScreenSize** that return the value of the corresponding field.
2. Define method called **setChannel** accepts a value to be stored in the channel field.
3. Define method called **power** that changes the state from true to false or from false to true. This can be accomplished by using the NOT operator (!). If the boolean variable powerOn is true, then !powerOn is false and vice versa.

Use the assignment statement:

powerOn = !powerOn;

to change the state of powerOn and then store it back into powerOn (remember assignment statements evaluate the right hand side first, then assign the result to the left hand side variable).

1. Define two methods to change the volume. One method should be called **increaseVolume** and will increase the volume by 1. The other method should be called **decreaseVolume** and will decrease the volume by 1.
2. Write comments above each method header describing the purpose of the method.
3. Compile and debug. Do not run.

**Task 4**

**Running the Application**

1. You can only execute (run) a program that has a main method, so there is a driver program that is already written to test out your Television class. Copy the file TelevisionDemo.java from Whiteboard. Make sure it is in the same directory as Television.java.
2. Compile and run TelevisionDemo and follow the prompts.
3. If your output matches the output below, Television.java is complete and correct. You will not need to modify it further for this lab.

**You will not need to modify it further for this lab.**

**OUTPUT (boldface is user input)**

**A 55 inch Toshiba has been turned on.**

**What channel do you want? 56**

**Channel: 56 Volume: 21**

**Too loud!! I am lowering the volume.**

**Channel: 56 Volume: 15**

1. **Stage v (verify)**

**Home Activities:**

**Exercise 1:**

**Creating another instance of a Television**

1. Edit the TelevisionDemo.java file.
2. Add to the comment header as indicated at the top of the program.
3. Declare another Television object called portable.
4. Instantiate portable to be a Sharp 19 inch television.
5. Use a call to the power method to turn the power on.
6. Use calls to the methods to print what television was turned on.
7. Use calls to the methods to change the channel to the user’s preference and decrease the volume by two.
8. Use calls to the methods to print the changed state of the portable.
9. Compile and debug this class.
10. Run TelevisionDemo again.
11. The output for task #5 will appear after the output from above, since we added onto the bottom of the program. The output for task #5 is shown below.

**OUTPUT (boldface is user input)**

**A 19 inch Sharp has been turned on.**

**What channel do you want? 7**

**Channel: 7 Volumes: 18**

**Exercise 2:**

**Employee Class Scenario:**

**Part A:**

* Create a class called **Employee** that includes three pieces of information as instance variables about employee

**A first name (type String), A last name (type String) and A monthly salary (double).**

* Your class should have **a constructor that initializes the three instance variables.**
* Provide a **set** and a **get method** for each instance variable.
* If the monthly salary is not positive, set it to 0.0.

**Part B:**

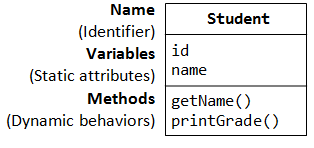
Write a test application named **EmployeeTest** that demonstrates class Employee's capabilities.

* Create two Employee objects and
* Display each object's yearly salary.
* Then give each Employee a 10% raise and display each Employee's yearly salary again.

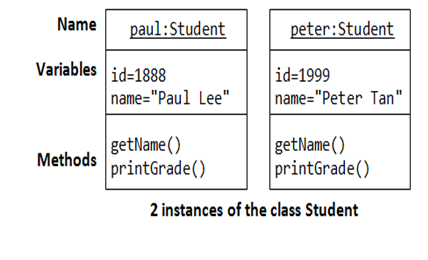
**(Optional) Exercise 3:**

As you all know by this time that Object-oriented programming (OOP) using Java is a programming language model organized around “**Classes**” and "**objects**”. The following figure shows **Student** is a **class** which contains

1. Variables (or attribute, state, field): contains the static attributes of the class.
2. Methods (or behavior, function, operation): contains the dynamic behaviors of the class.



And “paul” and “peter” are two instance (objects) of the Student class.



**Your Assignment Activities are as follows:**

1. Keeping into the consideration of above example you have to choose 3 different real-world scenarios, which explain the concept of class and object.

You have to draw and explain how the Object is related with the class and list the relevant attributes (data members) and behavior (methods) of the class.

1. Your other part of the assignment activities is to implement (Write Code) of above 3 different real life scenario in Java.

**Note:** Before submitting your report you have to makes sure that you will test your code with different data provided / entered by user using appropriate Standard classes available in java.

1. **Stage a2 (assess)**

**Homework 4:**

For this lab you are requested to solve Lab homework and to submit it before the deadline.

**Lab Homework:**

The laboratory homework of this lab (and also all the following labs in this manual) should include the following items/sections:

* + - * A cover page with your name, course information, lab number and title, and date of submission.
      * Programming: a brief description of the process you followed in conducting the coding of the homework solution.
      * Results obtained throughout the written code followed with brief analysis of these results.
      * A zip file including both the Java project folder and the work file contains all items/sections above.

**Lab Report:**

The laboratory report of this lab (and also all the following labs in this manual) should include the following items/sections:

* + - * A cover page with your name, course information, lab number and title, and date of submission.
      * A summary of the addressed topic and objectives of the lab.
      * Implementation: a brief description of the process you followed in conducting the implementation of the lab scenarios.
      * Results obtained throughout the lab implementation, the analysis of these results, and a comparison of these results with your expectations.
      * Answers to the given exercises at the end of the lab. If an answer incorporates new graphs, analysis of these graphs should be included here.
      * A conclusion that includes what you learned, difficulties you faced, and any suggested extensions/improvements to the lab.

**Note:** only a softcopy is required for both **homework** and **report**, please do not print or submit a hard copy.