# Object-Oriented Development (CIS1056-N) Worksheet 07(B): Object-Oriented Programming

## Before You Start

Remember: You are not expected to complete the entire brief within the allotted two hours, but to make a start and continue outside of the class.

Before proceeding with object-oriented programming, you must be comfortable with program flow control *and* methods. You should be able to read, understand, and write conditional (if) and iterative statements (while and for). You should be able to read and write simple Java methods (see last worksheet 5).

## Introduction

Now we understand the fundamentals of programming, it is time to take a deeper dive into the dominant programming paradigm of our time, object-oriented programming.

Based on the teaching material so far, it should now be apparent that Java is an “object-first” language. You have worked with several Java objects already, notably Scanner and String. Note Java is not a *purely* object-based language (as Java supports primitive types), but as we need some class structure to write a meaningful program, objects are inescapable.

**Hint:** In all that follows, each class is to be in its own file with the same name as the class name.

## 1. Lecture recap: Lamp

Create a new NetBeans project called **Lamp**.

Looking back at the example of the **Lamp** class studied during the *Introduction to OOP* lecture, create a file Lamp.java and a file TestLamp.java.

Add the following code to Lamp.java:

*Activity continues next page.*

**public class Lamp {**

**// Properties**

**private boolean state;**

**// Constructor**

**public Lamp() {**

**this.state = false;**

**}**

**// Methods**

**public void switchOn() {**

**this.state = true;**

**}**

**public void switchOff() {**

**this.state = false;**

**}**

**public boolean isState() {**

**return state;**

**}**

**}**

TestLamp.java will be your driver class to test your Lamp class. Add the following code to TestLamp.java:

**public class TestLamp {**

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

**Lamp lamp1 = new Lamp();**

**System.out.println("State is " + lamp1.isState());**

**lamp1.switchOn();**

**System.out.println("State is " + lamp1.isState());**

**lamp1.switchOff();**

**System.out.println("State is " + lamp1.isState());**

**}**

**}**

Run TestLamp.java. Your output should be:

**State is false**

**State is true**

**State is false**

This simple example demonstrates **encapsulation,** the bundling of data with the methods that operate on that data, whilst restricting direct access outside of the object. The Lamp object has a state, which can only be accessed through **switchOn()**, **switchOff()**, and **isState()**.

## 2. Lift

Create a new NetBeans project called **Lift**.

In the project **Lift**, create a new file Lift.java and add the following code:

**public class Lift {**

**// Properties**

**private int currentFloor;**

**private boolean doorsOpen = false;**

**// Constructors**

**public Lift() {**

**this(0);**

**}**

**public Lift(int currentFloor) {**

**this.currentFloor = currentFloor;**

**}**

**// Methods**

**public int getCurrentFloor() {**

**return currentFloor;**

**}**

**public boolean isDoorsOpen() {**

**return doorsOpen;**

**}**

**public void toogleDoors() {**

**doorsOpen = !doorsOpen;**

**}**

**public void setCurrentFloor(int cf) {**

**currentFloor = cf;**

**}**

**public void goUp() {**

**if(currentFloor < 10)**

**if(!doorsOpen)**

**currentFloor++;**

**else**

**System.out.println("Error lift doors not closed");**

**else**

**System.out.println("Error already at top floor");**

**}**

**}**

In the project Lift, create a new file TestLift.java and add the following code:

**public class TestLift {**

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

**Lift elevator = new Lift();**

**// open the doors**

**elevator.toogleDoors();**

**elevator.goUp(); // will fail - doors are open**

**// close the doors**

**elevator.toogleDoors();**

**elevator.goUp();**

**elevator.goUp();**

**System.out.println("The current floor is: " +**

**elevator.getCurrentFloor());**

**}**

**}**

Compile then execute **TestLift**. Your output should be:

Error lift doors not closed

The current floor is: 2

## 3. Add a second Lift object

Modify **TestLift** by adding a second **Lift** object. After the **elevator** declaration add the following line:

**Lift glassElevator = new Lift(4);**

What **Lift** property is set by the value 4?

Printing the current floor should give you a clue:

**System.out.println("The current floor is: " +**

**glassElevator.getCurrentFloor());**

## 4. Setting the top floor

The top floor for every **Lift** object is currently hardcoded to 10, as seen in the method **goUp()**:

**public void goUp() {**

**if(currentFloor < 10)**

**if(!doorsOpen)**

**currentFloor++;**

**else**

**System.out.println("Error lift doors not closed");**

**else**

**System.out.println("Error already at top floor");**

**}**

Let’s update the constructors to add a new private property **topFloor**:

**private int topFloor;**

Update the constructors to the following:

**// Constructors**

**public Lift() {**

**this(0);**

**}**

**public Lift(int currentFloor) {**

**this(currentFloor, 10);**

**}**

**public Lift(int currentFloor, int topFloor) {**

**this.currentFloor = currentFloor;**

**this.topFloor = topFloor;**

**}**

Finally update the conditional statement in goUp() to:

**public void goUp() {**

**if(currentFloor < topFloor)**

**if(!doorsOpen)**

**currentFloor++;**

**else**

**System.out.println("Error lift doors not closed");**

**else**

**System.out.println("Error already at top floor");**

**}**

Update the main method in TestLift to try out your new property. Ensure you create different instances of TestLift using all three constructors. Verify that it is not possible to go above the topFloor.

## 5. Car

Create a new NetBeans project called **Car**.

In the project **Car**, create a new file Car.java. The Car class should have three instance variables as follows:

* **topSpeed**, an int.
* **taxBand**, a char.
* **turbo**, a Boolean.

This class should not have a main method.

Create a new java file called TestCar.java. We will use this class to test the Car class. This file should have a main method.

In TestCar.java, create three Car objects (called c1, c2 and c3 respectively) and change the state of these objects as follows:

* c1 = top speed of 121mph, tax band C, no turbo.
* c2 = top speed of 178mph, tax band E, has a turbo.
* c3 = top speed of 62mph, tax band A, no turbo.

Print all of the values for each car in the following format:

Car name: c1

Top speed: 121 mph

Tax band: C

Turbo: false

## 6. Rectangle

Write a class Rectangle with the following characteristics:

* Instance variables: length and width.
* Methods: computeSurface and computePerimeter.

The constructor should initialise the instance variables to some default values.

Now write a driver class TestRectangle to exercise your Rectangle class. It should print the length, width, surface and perimeter of the rectangle instantiated.

Remember an instance of a class exists when an object of that class is created, i.e. it now exists in memory.

## 7. Password Checker

Write a program that validates a password by ensuring it contains at least one digit, one lowercase letter and one uppercase letter. The password must also be at least 8characters long.

Your program must NOT use regular expressions and make good use of methods to check if a character is a lowercase letter, one to check if a character is an uppercase letter and finally a third one to check if a character is a digit.

Your program must display suitable messages depending on the outcome.

Try to make sure you use methods for each identifiable task.

**Note**: Don’t create a class for this exercise, use static methods instead.

## 8. Password Class

Now create a class Password with the following instance variables that define its properties:

* String word
* int length
* boolean upper
* boolean lower
* boolean digit

Now give your class some behaviour by adding a few methods to obtain the length and check that the password contains some uppercase characters, some lowercase ones and some digits.

Test your Password class in a suitable driver class.

## 9. Pig Latin Class

Write an application that asks the user to enter a word.

The word is then used to create an instance of a class called **PigLatinText**. The word provided by the user stored within the **PigLatinText** instance as well as its Pig Latin equivalent.

You will need to create a method to translate the word to Pig Latin.

Pig Latin form = second letter to the end of the word + first letter of the word + “ay”.

Create a suitable driver class (i.e. the class with a main() method) to create instances and test the **PigLatinText** class.

Improvements:

1. Update your application by adding the option of offering the user the opportunity of entering and translating another word until “No” is entered.
2. Ask the user to enter a sentence in standard English, and then take each word and create an instance of **PigLatinText**. Store each **PigLatinText** in an array (the size of which is determined by the number of words the user entered).
3. Improve the **PigLatinText** translation to follow the rules found here: <https://www.dictionary.com/e/pig-latin/>

## Document History

Revision 0 (07-Nov-21): This is the initial version of the 2022/23 exercise.