# Department of Computing

**SE-210: Software Design and Architecture**

**Class:** BESE-9AB

# Lab 03: Object Oriented Design Principles (Class Level)

# Lab Engineer: Sundas Dawood

# 

**Ahmed Hassan Ismail – BESE-9B – 237897**

# Lab 03: Object Oriented Design Principles (Class level)

### Introduction:

Students have learned the theoretical concepts of *class level object-oriented design principles* in lectures. In this lab, students will learn how to identify and fix pieces of codes where these principles are violated.

### Lab Objectives:

After the completion of this lab, students will be able to identify which parts of software are violating *class level object-oriented principles* and how to fix them.

### Helping Material:

Please consult lectures slides and *Supporting Material – Design Principles* available on LMS.

### Lab Tasks

### Task 1

We have learned five principles of class level object-oriented design formally known as SOLID principles. The five principles are:

* The Single Responsibility Principle (SRP)
* The Open Closed Principle (OCP)
* The Liskov Substitution Principle (LSP)

Your task:

1. *For each of the principles*, you must *give one coding examples* where the *principles are violating*.
2. Given *a brief explanation of why the piece of code is violating* the principle.
3. A *refactored version of the code*, where the principles are respected.
4. Given *a brief explanation of why the refactored version of the code is respecting* the principle.

**Important Note:** You are not required to write a fully functional code. Only write enough code which can make your point.

**Example**

**Liskov Substitution Principle**

**Violating Code:**

class Rectangle{  
 protected int width;  
 protected int height;  
 public void setWidth(int w){width = w;}  
 public void setHeight(int h){height = h;}  
 public int getWidth(){return width;}  
 public int getHeight(){return height;}  
 public int getArea(){return width \* height;}  
}

class Square extends Rectangle {  
 public void setWidth(int w){  
 width = w;  
 height = w;  
 }

public void setHeight(int h){  
 width = h;  
 height = h;   
 }  
}

**Why the code is violating LSP**

The class *Square* is extending the *Rectangle* class. Mathematically, there exist a strong relationship between a rectangle and square (square being a special form of rectangle where width and height are equal). However, behaviorally a square object is completely different from a rectangle object. In the above code, the setWidth and setHeight overridden functions of Square class are changing the behavior of the same functions in the parent class. Thus the subclass is no more substitutable for the base class, which violates Liskov Substitution Principle.

**Refactored Code**

abstract class Shape{  
 public void setWidth(int w){};   
 public void setHeight(int h){};  
}

class Rectangle extends Shape{  
 protected int width;  
 protected int height;  
 public void setWidth(int w){width = w;}  
 public void setHeight(int h){height = h;}  
 public int getWidth(){return width;}  
 public int getHeight(){return height;}  
 public int getArea(){return width \* height;}  
}

class Square extends Shape {  
 protected int width;  
 protected int height;  
 public void setWidth(int w){  
 width = w;  
 height = w;  
 }

public void setHeight(int h){  
 width = h;  
 height = h;  
 }  
 public int getWidth(){return width;}  
 public int getHeight(){return height;}  
 public int getArea(){return width \* height;}  
}

**Why the refactored code is respecting LSP**

The common behavior of setting width and height of the square and rectangle classes is generalized in the shape class. Both square and rectangle class now extend the shape class. Each class now defines it’s own behavior of setting width and height and the behaviors are no more contradicting.

**Answer:**

|  |
| --- |
| Solution |
| Task 1  Example #1 SRP  Violation:  class car{  void steering(int diameter){};  void tyres(int diameter){};  void engine(int capacity){};  void body(char color, char body\_type){};  void doors(int no\_of\_doors){}; }  The above code is violating SRP which states that every class should be having single responsibility, and, in this code, a single class of car is having all the responsibilities that a car must perform. To fix this we should divide each responsibility to each class.  Refactored Code:  class car {  int no\_of\_doors;  char color;  char body\_type; }  class Steering extends car {  int diameter;  void type\_o\_steering(int diameter) {  } }  class Doors extends car {  int no\_of\_doors;  void type\_of\_doors(int no\_of\_doors) {  } }  class Body extends car {  char color;  char shape;  void type\_of\_body(char color, char shape) {  } }  class Engine extends car {  int capacity;  void Engine(int capacity) {   } }  Example #2 SRP  Violation:  class Bank\_account{  void deposit(int amount){};  void Withdraw(int amount){};  void owner\_details(String name,String details){}; }  This code is violating SRP as all functionalities are in same class.  Refactored code:  class Bank\_account{  String name;  char details;  int acc\_no;  void owner\_info(String name, char details, int acc\_no){} } class Deposit extends Bank\_account{  int amount;  void deposit\_cash(int acc\_no,int amount){}; } class Withdraw extends Bank\_account{  private int amount;  int acc\_no;  void withdraw(int amount,int acc\_no){}; }  Example #1 OCP:  Violation:  Class animal\_voices{ public static String  {  getDogNoise(Animal Dog)  {  if (animal instanceof Dog){  return "Woof";  }  getCatNoise(Animal cat)  if (animal instanceof Cat){  return "Miau";  }  }  If we must add another animal, we’ve to modify this code. Which means OCP is violated as its not close for modification.  Refactored Code:  Class Animal\_voices{  public static String voice;  void animals\_voice(voice);  }  Class Dog references Animal\_voice{  string getDogNoise(Animal Dog)  {  if (animal instanceof Dog){  return "Woof";  }  }  Class cat references Animal\_voice{  string getCatNoise(Animal cat)  if (animal instanceof Cat){  return "Miau";  }  }  Example #2 OCP  Violation:  Class animal\_voices{ public static String  {  getDogNoise(Animal Dog)  {  if (animal instanceof Dog){  return "Woof";  }  getCatNoise(Animal cat)  if (animal instanceof Cat){  return "Miau";  }  }  If we have to add another account related detail like Balance inquiry we’ve to modify this code. Which means OCP is violated as its not close for modification.  Refactored code:  Class Bank\_account{  Private string name;  Private details;  Int acc\_no  Void owner\_info(name, details,acc\_no){get;set}  }  Class Deposit refrences Bank\_account{  Private int amount;  Void\_deposit\_cash(acc\_no,amount);  }  Class Bank\_account{  Private string name;  Private details;  Int acc\_no  Void owner\_info(name, details,acc\_no){get;set}  }  Class Deposit refrences Bank\_account{  Private int amount;  Void\_deposit\_cash(acc\_no,amount);  }  Class Withdraw refrences Bank\_account  {  Private int amount;  Void withdraw(amount,acc\_no);  }  Class balance refrences Bank\_account  {  Void balance\_inquiry(acc\_no);  }  Example #1 LSP  Violation:  Class car {  Void owner\_details();  Void Engine\_details();  }  Class bike extends car{  Void body\_details(){get;set}  Void tyre\_details(){get;set}  }  Int main(){  Model\_Info(car);  Bike b;  Model\_Info(b);  }  Its violating LSP because car and bike are not subsituitable at each others place.  Refactored Code:  Class Vehicle {  Void owner\_details();  Void Engine\_details();  }  Class Car extends Vehicle{  Void body\_details(){get:set};  Void tyre\_details(){get:set}  }  Class bike extends Vehicle{  Void body\_details(){get;set}  Void tyre\_details(){get;set}  }  Int main(){  parking(car);  vehicle v;  parking(v);  }  Example #2 LSP  Violation:  class Car {      function startEngine() {          // Default engine start functionality      }      function accelerate() {          // Default acceleration functionality      }      function startEngine() {          $this->engageIgnition();          parent::startEngine();      }        private function engageIgnition() {          // Ignition procedure      }   }  }  class Bus extends car {      function accelerate() {          $this->increaseVoltage();          $this->connectIndividualEngines();      }      private function increaseVoltage() {          // Electric logic      }  private function connectIndividualEngines() {          // Connection logic      }  }  Both classes are non-substitutable as Car is not a Bus. And thus, we must make a vehicle which will extended by car and electric bus.  Refactored Code:  class Vehicle {      function startEngine() {          // Default engine start functionality      }      function accelerate() {          // Default acceleration functionality      }  }  class Car extends Vehicle {        function startEngine() {          $this->engageIgnition();          parent::startEngine();      }        private function engageIgnition() {          // Ignition procedure      }  }  class Bus extends Vehicle {      function accelerate() {          $this->increaseVoltage();          $this->connectIndividualEngines();      }     private function increaseVoltage() {          // Electric logic      }      private function connectIndividualEngines() {          // Connection logic      }  }  class Driver {      function go(Vehicle $v) {          $v->startEngine();          $v->accelerate();      }  } |

### Deliverables

Compile a single word document by filling in the solution part and submit this Word file on LMS. This lab grading policy is as follows: The lab is graded between 0 to 10 marks. The submitted solution can get a maximum of 5 marks. At the end of each lab or in the next lab, there will be a viva related to the tasks. The viva has a weightage of 5 marks. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your completed Word document to get your work graded. You must also submit this Word document on the LMS. In case of any problems with submissions on LMS, submit your Lab assignments by emailing it to **Sundas Dawood** <sundas.dawood@seecs.edu.pk>