# Department of Computing

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

**Class:** BESE-9AB

# Lab 06: Creational Design Patterns

# Instructor: Dr. Hasan Ali Khattak

# 

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

# Lab 06: Creational Design Patterns

### Introduction:

Students will have hands-on experience of implementing a creational design pattern to a given problem.

### Lab Objectives:

This objective of this lab is to get a practical understanding and knowledge of Builder design pattern.

### Helping Material:

Please consult lecture slides on LMS.

### Lab Tasks

### Task 1

Given the interface and the Builder test class, you need to provide missing functionality using separate builders for “Car” and “Motorcycle” classes. Here, director refers to base builder class.

//The common interface

interface Builder

{

void startUpOperations();

void buildBody();

void insertWheels();

void addHeadlights();

void endOperations();

/\*The following method is used to retrieve the object that is constructed.\*/

Product getVehicle();

}

public class BuilderPatternExample {

public static void main(String[] args) {

System.out.println("\*\*\*Builder Pattern Demo\*\*\*");

Director director = new Director();

Builder fordCar = new Car("Ford");

Builder hondaMotorycle = new MotorCycle("Honda");

// Making Car

director.construct(fordCar);

Product p1 = fordCar.getVehicle();

p1.showProduct();

//Making MotorCycle

director.construct(hondaMotorycle );

Product p2 = hondaMotorycle.getVehicle();

p2.showProduct();

}

}

Based on the scenario given above, propose a solution using builder design pattern. Your tasks are as follows:

1. Draw UML Class diagram of the solution – after applying builder design pattern.
2. Fully functional code of the solution. Your main function should check the underlying operating system and automatically create widgets specific for the underlying operating system. To check the underlying operating system use following code:

**Answer:**

|  |
| --- |
| Solution |
| UML Class Diagram:    Source Code:  package builder.pattern.demo;  import java.util.LinkedList;  // Builders common interface  interface IBuilder  {  void BuildBody();  void InsertWheels();  void AddHeadlights();  Product GetVehicle();  }  // Car is ConcreteBuilder  class Car implements IBuilder  {  private Product product = new Product();  @Override  public void BuildBody()  {  product.Add("This is a body of a Car");  }  @Override  public void InsertWheels()  {  product.Add("4 wheels are added");  }  @Override  public void AddHeadlights()  {  product.Add("2 Headlights are added");  }  @Override  public Product GetVehicle()  {  return product;  }  }  // Motorcycle is a ConcreteBuilder  class MotorCycle implements IBuilder  {  private Product product = new Product();  @Override  public void BuildBody()  {  product.Add("This is a body of a Motorcycle");  }  @Override  public void InsertWheels()  {  product.Add("2 wheels are added");  }  @Override  public void AddHeadlights()  {  product.Add("1 Headlights are added");  }  @Override  public Product GetVehicle()  {  return product;  }  }  // "Product"  class Product  {  // We can use any data structure that you prefer. We have used LinkedList here.  private LinkedList<String> parts;  public Product()  {  parts = new LinkedList<String>();  }  public void Add(String part)  {  //Adding parts  parts.addLast(part);  }  public void Show()  {  System.out.println("\n Product completed as below :");  for(int i=0;i<parts.size();i++)  {  System.out.println(parts.get(i));  }  }  }  // "Director"  class Director  {  IBuilder myBuilder;  // A series of steps—for the production  public void Construct(IBuilder builder)  {  myBuilder=builder;  myBuilder.BuildBody();  myBuilder.InsertWheels();  myBuilder.AddHeadlights();  }  }  class BuilderPatternEx  {  public static void main(String[] args)  {  System.out.println("\*\*\*Builder Pattern Demo\*\*\*\n");  Director director = new Director();  IBuilder carBuilder = new Car();  IBuilder motorBuilder = new MotorCycle();  // Making Car  director.Construct(carBuilder);  Product p1 = carBuilder.GetVehicle();  p1.Show();  //Making MotorCycle  director.Construct(motorBuilder);  Product p2 = motorBuilder.GetVehicle();  p2.Show();  }  }  Output: |

### 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.