**Assignment – 1**

|  |  |
| --- | --- |
| **Student Name/ID Number:** | Francis Roel L. Abarca | bdse-0922-113 |
| **Academic Year:** | 2022-2023 |
| **Unit Assessor:** | Archana Sakpal |
| **Project Title:** | Assignment1– Inversion of Control & DI |
| **Issue Date:** | 4/11/2023 |
| **Submission Date:** | 4/11/2023 |
| **Internal Verifier Name** | Archana Sakpal |
| **Date:** | 4/11/2023 |

|  |
| --- |
| **Learner declaration** |
| I certify that the work submitted for this assignment is my own and research sources are fully acknowledged.  Student signature: Date: 4/11/2023 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Module No:** | 7 | **IU No:** | 1 | **Exercise No.** | 1 |

|  |  |
| --- | --- |
| **Lab Assessment Statement** | **Assignment 1 - Inversion of Control**  **You are handling the Billing module of an e-commerce application. You have developed below class.**  package lithan.training.javawebapp  public class BillingService {  private PaymentProcessor paymentProcessor;    public bool completePayment() {  return paymentProcessor.process();  }  }  **Note:**   * **You are using a 3rd party library (external library) to process the payments** * **This external library provides an interface PaymentProcessor for clients to process the payments** * **The external library supports 2 modes of payment. “gpay” and “credit\_card”.** * **It also has a Factory class which will instantiate appropriate processor based on mode of payment.** * **Below is the brief outline of the classes provided by external library**   public interface PaymentProcessor {  public bool process();  }  public class GooglePayProcessor implements PaymentProcessor {  public bool process() {  // process payment  return true;  }  }  public class CreditCardProcessor implements PaymentProcessor {  public bool process() {  // process payment  return true;  }  }  public PaymentProcessorFactory {  private final GooglePayProcessor gpayProcessor = new GooglePayProcessor();  private final CreditCardProcessor cardProcessor = new CreditCardProcessor ();    public PaymentProcessor getPaymentProcessor(String mode) {  if ("gpay".equalsIgnoreCase(mode)) {  return gpayProcessor;  } else if ("credit\_card".equalsIgnoreCase(mode)) {  return cardProcessor ;  }  return null;  }  }     1. **Enhance the BillingService class so it gets instantiated and the dependencies shall get injected.**    * Hint: What method would you use to instantiate BillingService class? Remember, this is the bean developed by you. 2. **Write Java Configuration class to instantiate required classes from external library.** |
| **Technical Environment** | - |
| **Guidelines** | - |
| **Duration** | 120 mins |

Solution:

Create a Maven Project

Quickstart 1.1

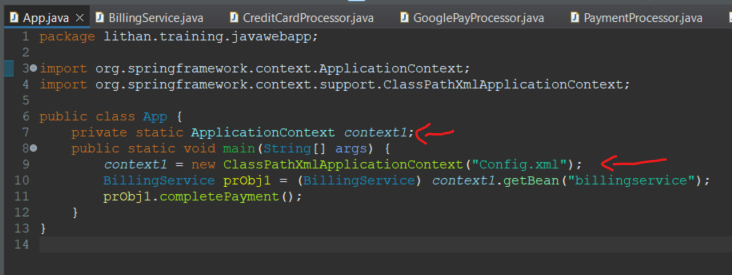
1. Source Code of BillingService Class

Text

Description automatically generated

The BillingService class utilizes the getter and setter methods together with the Setter Dependency Injection (SDI) with the purpose of removing its reliance of the BillingService to paymentProcessor which makes BillingService.java loosely coupled and allows the developer to gain the ability to reuse the classes.

2. Source Code of App.java



The IoC container inside this project is used to manage all the components inside the application for it to be used which are then enclosed into Spring Beans.

Inside the IoC container, we used the Spring ApplicationContext container since it already holds most of the functionality of the BeanFactoryContainer.

The context1 method reads the Config.xml metadata to have an idea as to what pointers and objects to instantiate from which class available, set them up then assemble its dependencies between the objects given.

3. Source Code for Config.xml

Text

Description automatically generated

Inside the Config.xml file, both the constructor and setter-based Dependency Injections were used. The main rule for constructors is to use Constructor arguments for required dependencies and setters for optional dependencies.

For BillingService.java class, we used a Setter-based Dependency Injection because the class utilizes getter and setter methods with no constructor argument.

For PaymentProcessorFactory.java class, this class utilizes the Factory method and Constructor-based Dependency Injection method since compared to BillingService.java class, this class contains constructor arguments. Text

Description automatically generated

In this class, the Factory Method is used to inject the bean which is required to initialize getPaymentProcessor since the Factory bean in the project is considered to be the reference of the bean which is inside the PaymentProcessorFactory class.

4. Screen capture of the Output file.

Text

Description automatically generated

Text

Description automatically generated

When the value for the constructor-arg inside the customerpayment bean is changed, the output result is also changed to match what’s specified inside the value.