Design Patterns:

SingleTon Pattern:

```
package com.singleton;


public class Logger {
   private static Logger instance;


   private Logger() {
      System.out.println("Logger instance created.");
   }


   public static Logger getInstance() {
      if (instance == null) {
         instance = new Logger();
      }
      return instance;
   }


   public void log(String message) {
      System.out.println("Log: " + message);
   }
}
```

Main.java:

```
package com.singleton;


public class Main {
   public static void main(String[] args) {


      Logger logger1 = Logger.getInstance();
      logger1.log("Starting the application...");
```

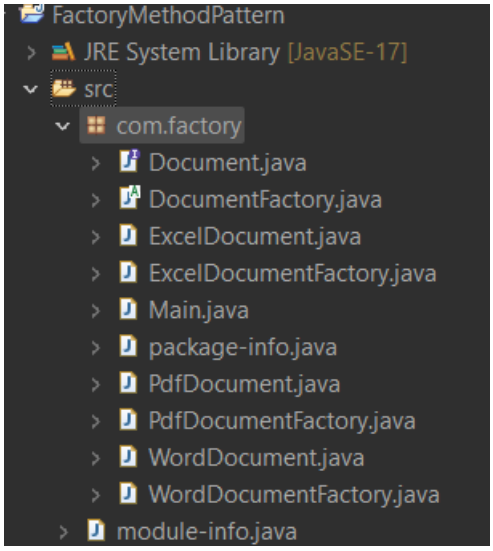
```
        Logger logger2 = Logger.getInstance();
        logger2.log("Continuing the application...");

        if (logger1 == logger2) {
            System.out.println("Only one Logger instance exists. Singleton confirmed!");
        } else {
            System.out.println("Multiple instances found. Singleton failed.");
        }
    }
}
```

Output:

```
Logger instance created.
Log: Starting the application...
Log: Continuing the application...
Only one Logger instance exists. Singleton confirmed!
```

Exercise 2: Implementing Factory Method:

Flow:



Main:

```java
package com.factory;

public class Main {
    public static void main(String[] args) {

        DocumentFactory wordFactory = new WordDocumentFactory();
        Document wordDoc = wordFactory.createDocument();
        wordDoc.open();

        DocumentFactory pdfFactory = new PdfDocumentFactory();
        Document pdfDoc = pdfFactory.createDocument();
        pdfDoc.open();

        DocumentFactory excelFactory = new ExcelDocumentFactory();
        Document excelDoc = excelFactory.createDocument();
        excelDoc.open();
    }
}
```

Document Open:

```java
package com.factory;

public abstract class DocumentFactory {
    public abstract Document createDocument();

}

```

```java
package com.factory;

public interface Document {
    void open();

}

```

Word:

```java
package com.factory;

public class WordDocumentFactory extends DocumentFactory{
    @Override
    public Document createDocument() {
        return new WordDocument();
    }
}
```

```java
package com.factory;

public class WordDocument implements Document{
    @Override
    public void open() {
        System.out.println("Opening Word Document.");
    }
}
```

PDF:

```java
package com.factory;

public class PdfDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new PdfDocument();
    }
}
```

```java
package com.factory;

public class PdfDocument implements Document {
    @Override
    public void open() {
        System.out.println("Opening PDF Document.");
    }
}
```

Excel:

```java
package com.factory;

public class ExcelDocumentFactory extends DocumentFactory{
    @Override
    public Document createDocument() {
        return new ExcelDocument();
    }
}
```

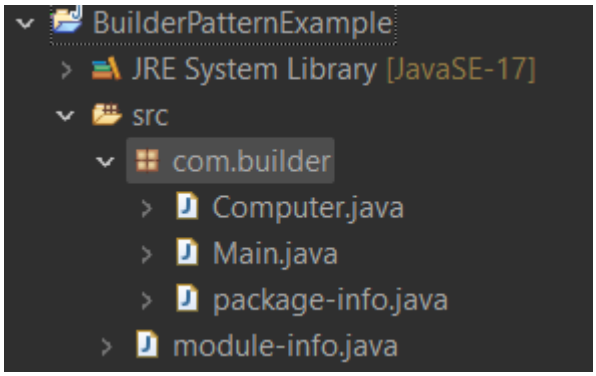
```java
package com.factory;

public class ExcelDocument implements Document{
    @Override
    public void open() {
        System.out.println("Opening Excel Document.");
    }
}
```

Output:

```
Problems  Javadoc  Declaration  Search  Console  Console ×
<terminated> Main (1) [Java Application] C:\Users\tunug\.p2\pool\plugins\org.eclipse
Opening Word Document.
Opening PDF Document.
Opening Excel Document.
```

Exercise 3: Builder Pattern:

Flow:



Computer:

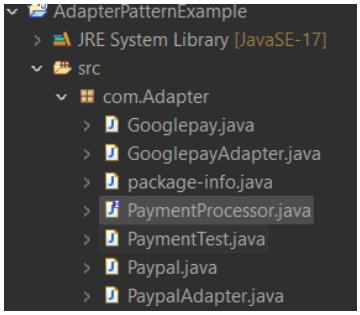
```java
package com.builder;

public class Main {
    public static void main(String[] args) {

        Computer gamingPC = new Computer.Builder()
                .setCPU("Intel Core i9")
                .setRAM("32GB")
                .setStorage("1TB SSD")
                .setGraphicsCard("NVIDIA RTX 4080")
                .build();

        System.out.println("Gaming PC Specs:");
        gamingPC.showSpecs();

        System.out.println();

        Computer officePC = new Computer.Builder()
                .setCPU("Intel Core i5")
                .setRAM("8GB")
                .setStorage("512GB SSD")
                .build();

        System.out.println("Office PC Specs:");
        officePC.showSpecs();
    }
}
```

Output:

```
Gaming PC Specs:
CPU: Intel Core i9
RAM: 32GB
Storage: 1TB SSD
Graphics Card: NVIDIA RTX 4080

Office PC Specs:
CPU: Intel Core i5
RAM: 8GB
Storage: 512GB SSD
Graphics Card: null
```

Exercise 4: Adapter Pattern:

Flow:



Googlepay:

```java
package com.Adapter;

public class Googlepay {
    public void makeGooglepayPayment(double amount) {
        System.out.println("Processing payment of $" + amount + " via Googlepay.");
    }
}
```

GooglePayAdapter:

```java
package com.Adapter;

public class GooglepayAdapter implements PaymentProcessor {
    private Googlepay stripeGateway;

    public GooglepayAdapter(Googlepay stripeGateway) {
        this.stripeGateway = stripeGateway;
    }

    @Override
    public void processPayment(double amount) {
        stripeGateway.makeGooglepayPayment(amount);
    }
}
```

Paypal.java:

```java
package com.Adapter;

public class Paypal {
    public void sendPayPalPayment(double amount) {
        System.out.println("Processing payment of $" + amount + " via PayPal.");
    }
}
```

PaypalAdapter:

```java
package com.Adapter;

public class PaypalAdapter implements PaymentProcessor {
    private Paypal paypal;

    public PaypalAdapter(Paypal paypal) {
        this.paypal = paypal;
    }

    @Override
    public void processPayment(double amount) {
        paypal.sendPayPalPayment(amount);
    }
}
```

PaymentProcessor:

```java
package com.Adapter;

public interface PaymentProcessor {
    void processPayment(double amount);
}
```

PaymentTest:

```java
package com.Adapter;

public class PaymentTest {
    public static void main(String[] args) {
        Googlepay stripe = new Googlepay();
        PaymentProcessor googlepayPayment = new GooglepayAdapter(stripe);
        googlepayPayment.processPayment(150.00);
        Paypal paypal = new Paypal();
        PaymentProcessor paypalPayment = new PaypalAdapter(paypal);
        paypalPayment.processPayment(200.50);
    }
}
```

Output:

```
Problems  Javadoc  Declaration  Search  Console  Console ×
<terminated> PaymentTest [Java Application] C:\Users\tunug\.p2\pool\plugins\org
Processing payment of $150.0 via Googlepay.
Processing payment of $200.5 via PayPal.
```

Exercise – 5: Decorator Pattern:

Notifier interface:

```java
package com.decorator;

public interface Notifier {
    void send(String message);
}
```

EmailNotifier:

```java
package com.decorator;

public class EmailNotifier implements Notifier {
    @Override
    public void send(String message) {
        System.out.println("Sending Email: " + message);
    }
}
```

SMSNotifier:

```java
package com.decorator;

public class SMSNotifierDecorator extends NotifierDecorator {

    public SMSNotifierDecorator(Notifier notifier) {
        super(notifier);
    }
    @Override
    public void send(String message) {
        super.send(message);
        sendSMS(message);
    }
    private void sendSMS(String message) {
        System.out.println("Sending SMS: " + message);
    }
}
```

NotifierDecorator:

```java
package com.decorator;

public abstract class NotifierDecorator implements Notifier {
    protected Notifier wrappedNotifier;

    public NotifierDecorator(Notifier notifier) {
        this.wrappedNotifier = notifier;
    }

    @Override
    public void send(String message) {
        wrappedNotifier.send(message);
    }
}
```

NotificationTest:

```java
package com.decorator;

public class NotificationTest {
    public static void main(String[] args) {
        Notifier notifier = new EmailNotifier();
        notifier = new SMSNotifierDecorator(notifier);
        notifier.send("Server is down!");
    }
}
```

Output:

```
Sending Email: Server is down!
Sending SMS: Server is down!
```

Exercise6: Proxy Pattern:

Image Interface:

```java
package com.proxy;

public interface Image {
    void display();
}
```

ImageViewer:

```java
package com.proxy;

public class ImageViewer {
    public static void main(String[] args) {
        Image image1 = new ProxyImage("nature.jpg");
        Image image2 = new ProxyImage("space.png");
        image1.display();
        System.out.println();
        image1.display();
        System.out.println();
        image2.display();
    }
}
```

ProxyImage:

```java
package com.proxy;

public class ProxyImage implements Image {
    private String filename;
    private RealImage realImage;

    public ProxyImage(String filename) {
        this.filename = filename;
    }
    @Override
    public void display() {
        if (realImage == null) {
            realImage = new RealImage(filename);
        } else {
            System.out.println("Image already loaded from cache: " + filename);
        }
        realImage.display();
    }
}
```

RealImage:

```java
package com.proxy;

public class RealImage implements Image {
    private String filename;

    public RealImage(String filename) {
        this.filename = filename;
        loadFromRemoteServer();
    }

    private void loadFromRemoteServer() {
        System.out.println("Loading image from remote server: " + filename);
    }
    @Override
    public void display() {
        System.out.println("Displaying image: " + filename);
    }
}
```

Output:

```
Problems  @ Javadoc  Declaration  Search  Console  Console ×
<terminated> ImageViewer [Java Application] C:\Users\tunug\.p2\pool\plugins\or
Loading image from remote server: nature.jpg
Displaying image: nature.jpg

Image already loaded from cache: nature.jpg
Displaying image: nature.jpg

Loading image from remote server: space.png
Displaying image: space.png
```

Exercise:7-Observer Pattern:

Observer.java

```java
package com.observer;

public interface Observer {
    void update(double price);
}
```

MobileApp-

```java
package com.observer;

public class MobileApp implements Observer
{
    private String name;
    public MobileApp(String name) {
        this.name = name;
    }
    @Override
    public void update(double price) {
        System.out.println("MobileApp " + name + ": Stock price changed to " + price);
    }
}
```

StockMarket.java-

```java
package com.observer;

import java.util.List;

import java.util.ArrayList;


public class StockMarket implements Stock {

  private List<Observer> observers = new ArrayList<>();

  private double stockPrice;

  @Override

  public void registerObserver(Observer o) {

    observers.add(o);

  }

  @Override

  public void removeObserver(Observer o) {

    observers.remove(o);

  }

  @Override

  public void notifyObservers() {

    for (Observer observer : observers) {

      observer.update(stockPrice);
```

```
    }
  }
  public void setStockPrice(double price) {
    this.stockPrice = price;
    System.out.println("\nStock price updated to: " + price);
    notifyObservers();
  }
}
```

Main.java-

```java
package com.observer;

public class Main
{
        public static void main(String[] args) {
                StockMarket market = new StockMarket();
                Observer mobileUser = new MobileApp("Alice");
                Observer webUser = new WebApp("Bob");
                market.registerObserver(mobileUser);
                market.registerObserver(webUser);
                market.setStockPrice(120.50);
                market.setStockPrice(135.75);
                market.removeObserver(mobileUser);
                market.setStockPrice(145.25);
            }
}
```

Output-

```
Stock price updated to: 120.5
MobileApp Alice: Stock price changed to 120.5
WebApp Bob: Stock price changed to 120.5

Stock price updated to: 135.75
MobileApp Alice: Stock price changed to 135.75
WebApp Bob: Stock price changed to 135.75

Stock price updated to: 145.25
WebApp Bob: Stock price changed to 145.25
```

Exercise 8 – Strategy Pattern

PaymentStrategy-

```java
package com.strategy;

public interface PaymentStrategy {
    void pay(double amount);
}
```

CreditCardPayment-

```java
package com.strategy;
public class CreditCardPayment implements PaymentStrategy{
    private String cardNumber;
        private String cardHolder;
        public CreditCardPayment(String cardNumber, String cardHolder) {
            this.cardNumber = cardNumber;
            this.cardHolder = cardHolder;
        }
        @Override
        public void pay(double amount) {
            System.out.println("Paid " + amount + " using Credit Card [" + cardNumber + "] for " + cardHolder);
        }
}
```

PaymentContext-

```java
package com.strategy;

public class PaymentContext {
    private PaymentStrategy paymentStrategy;
    public void setPaymentStrategy(PaymentStrategy paymentStrategy) {
        this.paymentStrategy = paymentStrategy;
    }
    public void payAmount(double amount) {
        if (paymentStrategy == null) {
            System.out.println("No payment method selected.");
        } else {
            paymentStrategy.pay(amount);
        }
    }
}
```

PaypalPayment-

```java
package com.strategy;
public class PayPalPayment implements PaymentStrategy {
    private String email;
        public PayPalPayment(String email) {
            this.email = email;
        }
        @Override
        public void pay(double amount) {
            System.out.println("Paid " + amount + " using PayPal with account: " + email);
        }
}
```

Main.java-

```java
package com.strategy;
public class Main {
    public static void main(String[] args) {
        PaymentContext context = new PaymentContext();
        context.setPaymentStrategy(new CreditCardPayment("1234-5678-9876-5432", "John Doe"));
        context.payAmount(250.75);
        context.setPaymentStrategy(new PayPalPayment("john.doe@example.com"));
        context.payAmount(120.50);
    }
}
```

Output –

```
Problems  Javadoc  Declaration  Search  Console  Console ×
<terminated> Main (3) [Java Application] C:\Users\tunug\.p2\pool\plugins\org.eclipse.just
Paid 250.75 using Credit Card [1234-5678-9876-5432] for John Doe
Paid 120.5 using PayPal with account: john.doe@example.com
```

Exercise 9- CommandPattern

Command.java-

```java
package com.command;

public interface Command {
      void execute();

}
```

Light.java-

```java
package com.command;

public class Light {
    public void turnOn() {
        System.out.println("Light is ON");
    }

    public void turnOff() {
        System.out.println("Light is OFF");
    }
}
```

LightOnCommand-

```java
package com.command;

public class LightOnCommand implements Command{
     private Light light;

        public LightOnCommand(Light light) {
            this.light = light;
        }

        @Override
        public void execute() {
            light.turnOn();
        }
}
```

LightOffCommand-

```java
package com.command;

public class LightOffCommand implements Command{
     private Light light;

        public LightOffCommand(Light light) {
            this.light = light;
        }

        @Override
        public void execute() {
            light.turnOff();
        }
}
```

Output-

```
Light is ON
Light is OFF
```

Eercise10 - MVC Command

Student.java:

```
package com.mvc;


public class Student {

   private String id;

   private String name;

   private String grade;


   // Getters and setters

   public String getId() {

      return id;

   }


   public void setId(String id) {

      this.id = id;

   }


   public String getName() {

      return name;

   }


   public void setName(String name) {

      this.name = name;

   }


   public String getGrade() {

      return grade;

   }


   public void setGrade(String grade) {
```

```
        this.grade = grade;
    }
}
```

StudentController:

```
package com.mvc;


public class StudentController
{
            private Student model;
    private View view;

    public StudentController(Student model, View view) {
        this.model = model;
        this.view = view;
    }

    // Controller methods to update model
    public void setStudentName(String name) {
        model.setName(name);
    }

    public void setStudentId(String id) {
        model.setId(id);
    }

    public void setStudentGrade(String grade) {
        model.setGrade(grade);
    }

    // Controller methods to retrieve model data
    public String getStudentName() {
```

```
        return model.getName();
    }

    public String getStudentId() {
        return model.getId();
    }

    public String getStudentGrade() {
        return model.getGrade();
    }

    public void updateView() {
        view.displayStudentDetails(model.getId(), model.getName(), model.getGrade());
    }

}
```

View.java:

```
package com.mvc;

public class View {
            public void displayStudentDetails(String id, String name, String grade) {
        System.out.println("Student Details:");
        System.out.println("ID    : " + id);
        System.out.println("Name  : " + name);
        System.out.println("Grade : " + grade);
    }
}
```

Main.java-

```
package com.mvc;

public class Main {
 public static void main(String[] args) {
```

```
    Student student = new Student();
    student.setId("101");
    student.setName("Harshitha");
    student.setGrade("A++");

    View view = new View();

    StudentController controller = new StudentController(student, view);

    controller.updateView();

    controller.setStudentName("Kavya");
    controller.setStudentGrade("A+");

    controller.updateView();
 }
}
```

Output-

```
Student Details:
ID    : 101
Name  : Harshitha
Grade : A++
Student Details:
ID    : 101
Name  : Kavya
Grade : A+
```

Exercise 11 – DependencyInjectionExample:

CustomerRepositary:

```java
package com.dependency;

public interface CustomerRepositary {
    String findCustomerById(String customerId);
}
```

CustomerRepositaryImpl:

```java
package com.dependency;

public class CustomerRepositaryImpl implements CustomerRepositary {
    @Override
        public String findCustomerById(String customerId) {
            return "Customer[ID: " + customerId + ", Name: Harshitha]";
        }
}
```

CustomerService:

```java
package com.dependency;

public class CustomerService {
    private CustomerRepositary customerRepositary;
        public CustomerService(CustomerRepositary customerRepositary) {
            this.customerRepositary = customerRepositary;
        }
        public void displayCustomer(String customerId) {
            String customer = customerRepositary.findCustomerById(customerId);
            System.out.println("Customer Info: " + customer);
        }
}
```

Main.java:

```java
package com.dependency;

public class Main {
        public static void main(String[] args) {
            CustomerRepositary repositary = new CustomerRepositaryImpl();
            CustomerService service = new CustomerService(repositary);
            service.displayCustomer("C101");
        }
}
```

Output:

```
Customer Info: Customer[ID: C101, Name: Harshitha]
```