Week 1 - Design Patterns - Hands-On Exercises

Exercise 1: Implementing the Singleton Pattern

Java Files: Logger.java , Main.java

CODE:

Logger.java

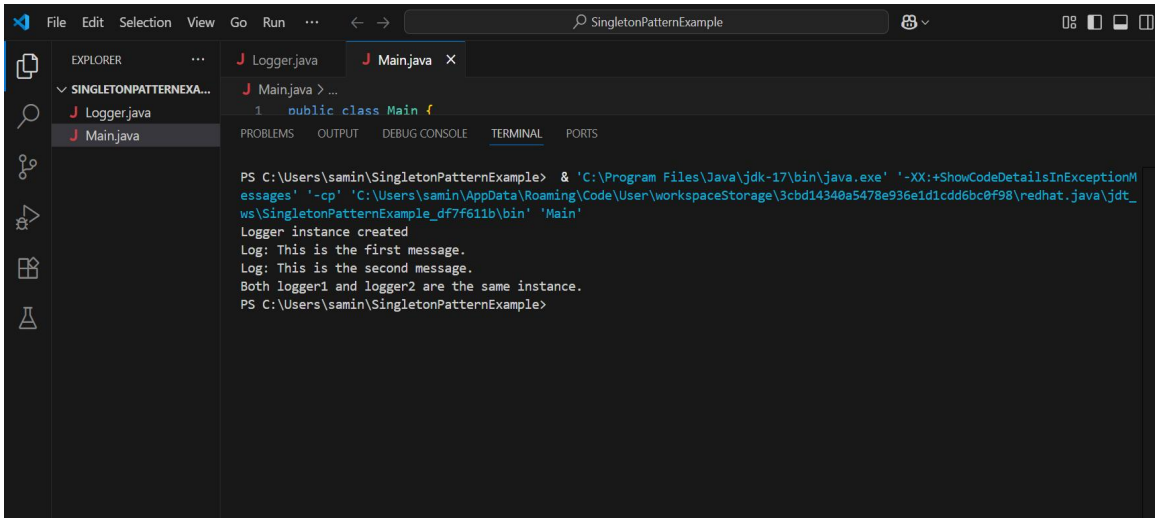
```
public class Logger {
   private static Logger instance;
   private Logger() {
      System.out.println("Logger instance created");
   }
   public static Logger getInstance() {
      if (instance == null) {
         instance = new Logger();
      }
      return instance;
   }
   public void log(String message) {
      System.out.println("Log: " + message);
   }
}
```

Main.java

```
public class Main {
      public static void main(String[] args) {
      Logger logger1 = Logger.getInstance();
```

```
        logger1.log("This is the first message.");
        Logger logger2 = Logger.getInstance();
        logger2.log("This is the second message.");
        if (logger1 == logger2) {
            System.out.println("Both logger1 and logger2 are the same instance.");
        } else {
            System.out.println("Different instances exist! Singleton pattern failed.");
        }
    }
}
```

OUTPUT:



Exercise 2: Implementing the Factory Method Pattern

Java Files:Document.java ,WordDocument.java ,PdfDocument.java ,

ExcelDocument.java ,DocumentFactory.java ,WordDocumentFactory.java ,etc.

CODE:

Document.java

```
public interface Document {
   void open();
}
```

WordDocument.java

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

PdfDocument.java

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

ExcelDocument.java

```
public class ExcelDocument implements Document {
   @Override
   public void open() {
```

```
        System.out.println("This is an Excel document.");
    }
}
```

DocumentFactory.java

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

WordDocumentFactory.java

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

PdfDocumentFactory.java

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

ExcelDocumentFactory.java

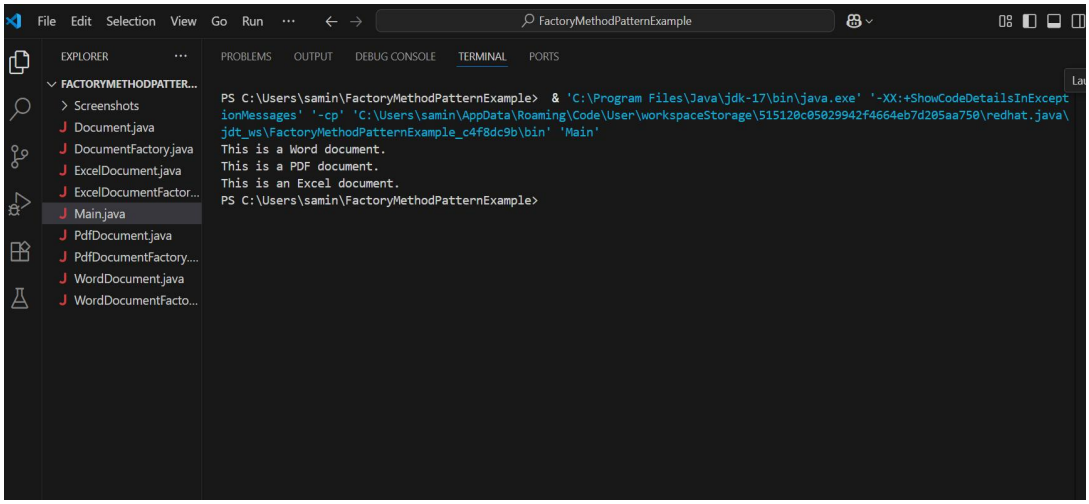
```
public class ExcelDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new ExcelDocument();
```

```
    }
}
```

Main.java

```
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();
    }
    }
```

OUTPUT:



Exercise 3: Implementing the Builder Pattern

Java Files: Computer.java (with nested Builder) , Main.java

CODE:

Computer.java

```
public class Computer {

   private String Name;

   private String CPU;

   private String RAM;

   private String storage;

   private String graphicsCard;

   private Computer(Builder builder) {

       this.Name = builder.Name;

      this.CPU = builder.CPU;

      this.RAM = builder.RAM;

      this.storage = builder.storage;

      this.graphicsCard = builder.graphicsCard;

   }

   public static class Builder {

      private String Name;

      private String CPU;

      private String RAM;

      private String storage;

      private String graphicsCard;

       public Builder setName(String Name) {

         this.Name = Name;

         return this;
```

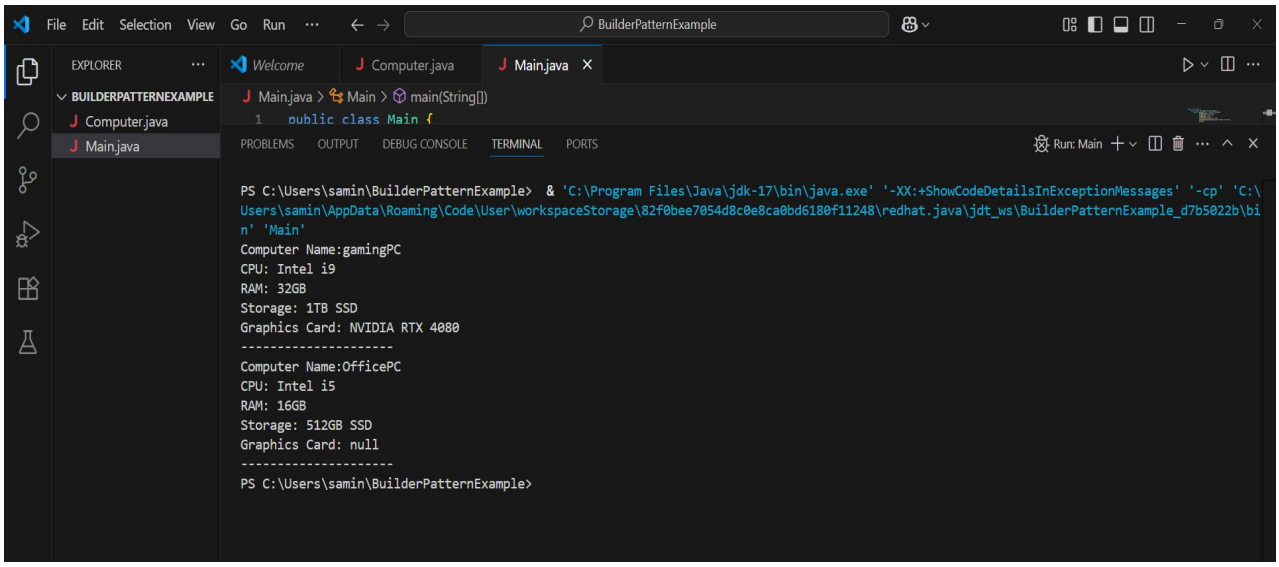
```
    }
    public Builder setCPU(String CPU) {
        this.CPU = CPU;
        return this;
    }
    public Builder setRAM(String RAM) {
        this.RAM = RAM;
        return this;
    }
    public Builder setStorage(String storage) {
        this.storage = storage;
        return this;
    }
    public Builder setGraphicsCard(String graphicsCard) {
        this.graphicsCard = graphicsCard;
        return this;
    }
    public Computer build() {
        return new Computer(this);
    }
}
public void showSpecs() {
    System.out.println("Computer Name:"+ Name);
    System.out.println("CPU: " + CPU);
    System.out.println("RAM: " + RAM);
    System.out.println("Storage: " + storage);
```

```
        System.out.println("Graphics Card: " + graphicsCard);
        System.out.println("---------------------");
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Computer gamingPC = new Computer.Builder()
                .setName("gamingPC")
                .setCPU("Intel i9")
                .setRAM("32GB")
                .setStorage("1TB SSD")
                .setGraphicsCard("NVIDIA RTX 4080")
                .build();
        Computer officePC = new Computer.Builder()
                .setName("OfficePC")
                .setCPU("Intel i5")
                .setRAM("16GB")
                .setStorage("512GB SSD")
                .build();
        gamingPC.showSpecs();
        officePC.showSpecs();
    }
}
```

OUTPUT:


File Edit Selection View Go Run
BuilderPatternExample
EXPLORER
BUILDERPATTERNEXAMPLE
Computer.java
Main.java
Welcome
Computer.java
Main.java
Main.java > Main > main(String[])
1 public class Main {
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Run: Main
PS C:\Users\samin\BuilderPatternExample> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\samin\AppData\Roaming\Code\User\workspaceStorage\82f0bee7054d8c0e8ca0bd6180f11248\redhat.java\jdt_ws\BuilderPatternExample_d7b5022b\bin' 'Main'
Computer Name:gamingPC
CPU: Intel i9
RAM: 32GB
Storage: 1TB SSD
Graphics Card: NVIDIA RTX 4080

Computer Name:OfficePC
CPU: Intel i5
RAM: 16GB
Storage: 512GB SSD
Graphics Card: null

PS C:\Users\samin\BuilderPatternExample>

Exercise 4: Implementing the Adapter Pattern

Java Files: PaymentProcessor.java, StripeAdapter.java, PayPalGateway.java, PayPalAdapter.java, etc.

CODE:

PaymentProcessor.java

```
public interface PaymentProcessor {

   void processPayment(double amount);

}
```

PayPalAdapter.java

```
public class PayPalAdapter implements PaymentProcessor {

   private PayPalGateway payPal;

   public PayPalAdapter(PayPalGateway payPal) {

      this.payPal = payPal;

   }

   @Override

   public void processPayment(double amount) {

      payPal.sendMoney(amount);

   }

}
```

StripeAdapter.java

```
public class StripeAdapter implements PaymentProcessor {

   private StripeGateway stripe;

   public StripeAdapter(StripeGateway stripe) {

      this.stripe = stripe;

   }

   @Override
```

```
    public void processPayment(double amount) {
        stripe.makePayment(amount);
    }
}
```

PayPalGateway.java

```
public class PayPalGateway {
    public void sendMoney(double amount) {
        System.out.println("Payment of $" + amount + " processed through PayPal.");
    }
}
```

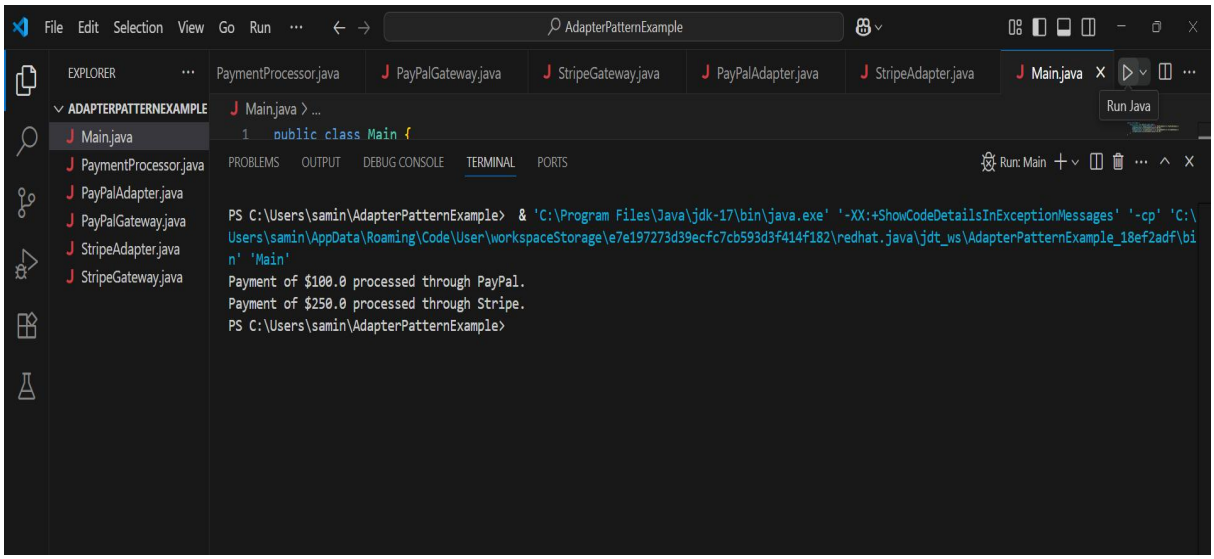
StripeGateway.java

```
public class StripeGateway {
    public void makePayment(double amount) {
        System.out.println("Payment of $" + amount + " processed through Stripe.");
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        PaymentProcessor paypalProcessor = new PayPalAdapter(new PayPalGateway());
        paypalProcessor.processPayment(100.00);
        PaymentProcessor stripeProcessor = new StripeAdapter(new StripeGateway());
        stripeProcessor.processPayment(250.00);
    }
}
```

OUTPUT:


File Edit Selection View Go Run
AdapterPatternExample
EXPLORER
ADAPTERPATTERNEXAMPLE
Main.java
PaymentProcessor.java
PayPalAdapter.java
PayPalGateway.java
StripeAdapter.java
StripeGateway.java
Main.java > ...
1 public class Main {
Run Java
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Run: Main
PS C:\Users\samin\AdapterPatternExample> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\samin\AppData\Roaming\Code\User\workspaceStorage\e7e197273d39ecfc7cb593d3f414f182\redhat.java\jdt_ws\AdapterPatternExample_18ef2adf\bin' 'Main'
Payment of $100.0 processed through PayPal.
Payment of $250.0 processed through Stripe.
PS C:\Users\samin\AdapterPatternExample>

Exercise 5: Implementing the Decorator Pattern

Java Files: Notifier.java, EmailNotifier.java, SMSNotifierDecorator.java, etc.

CODE:

Notifier.java

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

EmailNotifier.java

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

NotifierDecorator.java

```
public abstract class NotifierDecorator implements Notifier {
   protected Notifier wrappee;
   public NotifierDecorator(Notifier notifier) {
      this.wrappee = notifier;
   }
   @Override
   public void send(String message) {
      wrappee.send(message);
   }
}
```

SMSNotifierDecorator.java

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

SlackNotifierDecorator.java

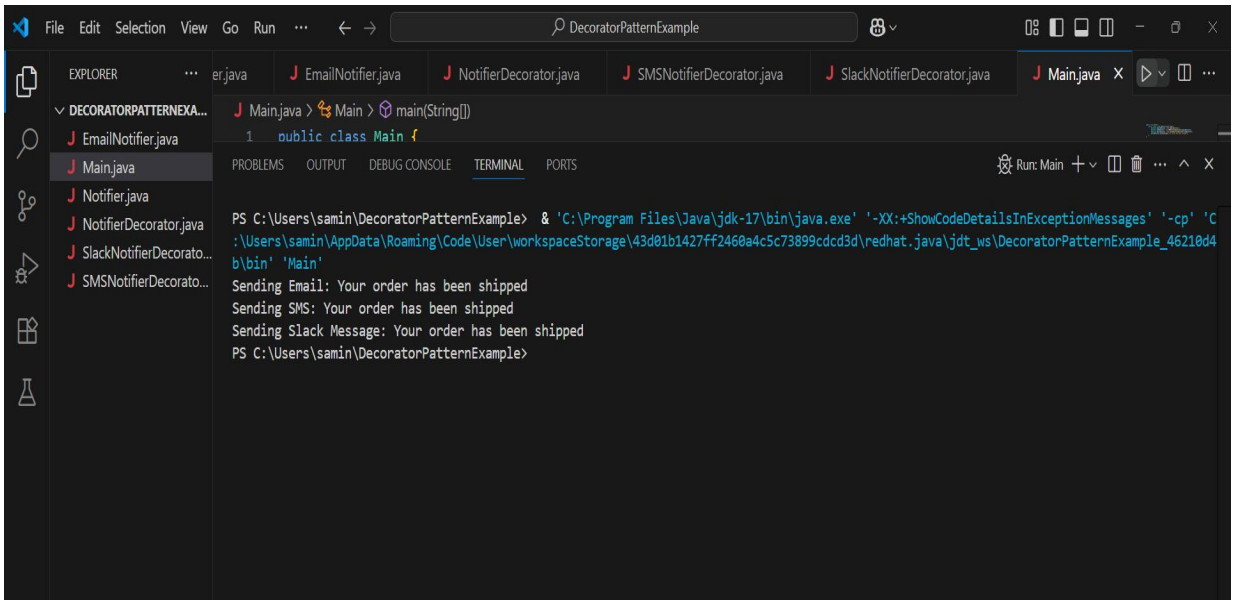
```
public class SlackNotifierDecorator extends NotifierDecorator {
    public SlackNotifierDecorator(Notifier notifier) {
        super(notifier);
    }
   @Override
    public void send(String message) {
        super.send(message);
        System.out.println("Sending Slack Message: " + message);
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Notifier emailNotifier = new EmailNotifier();
```

```
        Notifier smsNotifier = new SMSNotifierDecorator(emailNotifier);

        Notifier multiNotifier = new SlackNotifierDecorator(smsNotifier);

        multiNotifier.send("Your order has been shipped");

    }

}
```

OUTPUT:



Exercise 6: Implementing the Proxy Pattern

Java Files: Image.java, RealImage.java, ProxyImage.java, Main.java

CODE:

Image.java

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

RealImage.java

```
public class RealImage implements Image {
   private String fileName;
   public RealImage(String fileName) {
      this.fileName = fileName;
      loadFromServer();
   }
   private void loadFromServer() {
      System.out.println("Loading image from server: " + fileName);
   }
   @Override
   public void display() {
      System.out.println("Displaying image: " + fileName);
   }
}
```

ProxyImage.java

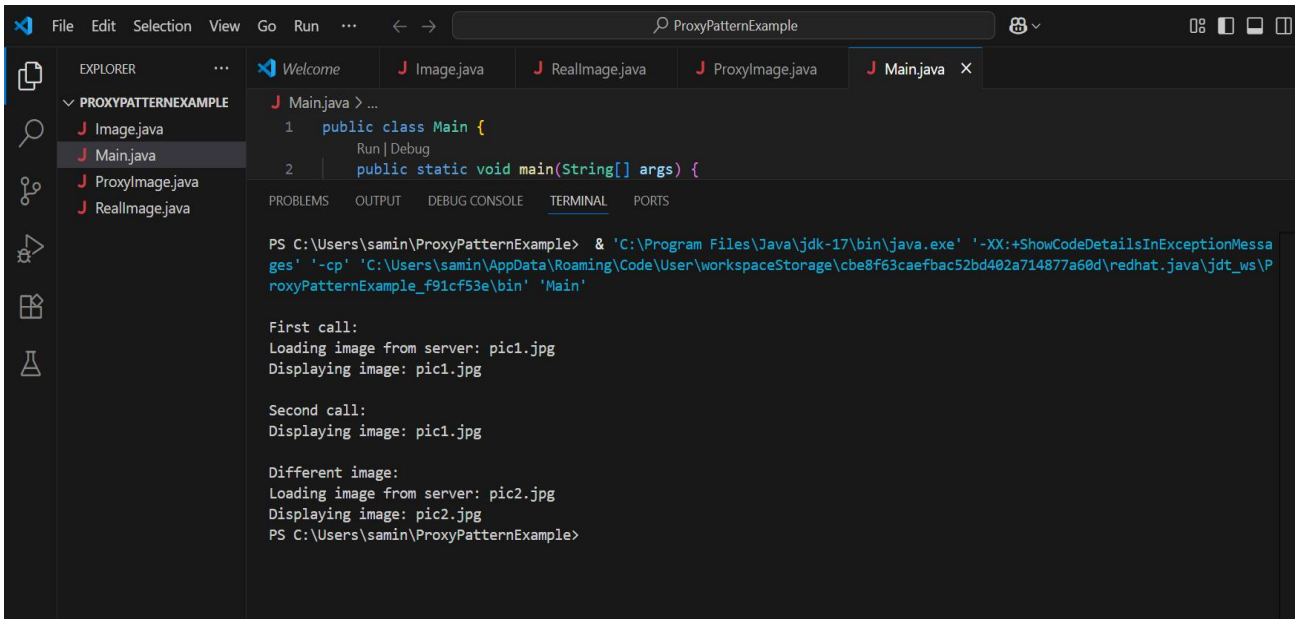
```
public class ProxyImage implements Image {
   private RealImage realImage;
   private String fileName;
```

```
    public ProxyImage(String fileName) {
        this.fileName = fileName;
    }
    @Override
    public void display() {
        if (realImage == null) {
            realImage = new RealImage(fileName);
        }
        realImage.display();
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Image image1 = new ProxyImage("pic1.jpg");
        Image image2 = new ProxyImage("pic2.jpg");
        System.out.println("\nFirst call:");
        image1.display();
        System.out.println("\nSecond call:");
        image1.display();
        System.out.println("\nDifferent image:");
        image2.display();
    }
}
```

OUTPUT:


File Edit Selection View Go Run
ProxyPatternExample
EXPLORER
PROXYPATTERNEXAMPLE
Image.java
Main.java
ProxyImage.java
RealImage.java
Welcome
Image.java
RealImage.java
ProxyImage.java
Main.java
Main.java > ...
public class Main {
Run | Debug
public static void main(String[] args) {
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\samin\ProxyPatternExample> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\samin\AppData\Roaming\Code\User\workspaceStorage\cbe8f63caefbac52bd402a714877a60d\redhat.java\jdt_ws\ProxyPatternExample_f91cf53e\bin' 'Main'
First call:
Loading image from server: pic1.jpg
Displaying image: pic1.jpg
Second call:
Displaying image: pic1.jpg
Different image:
Loading image from server: pic2.jpg
Displaying image: pic2.jpg
PS C:\Users\samin\ProxyPatternExample>

Exercise 7: Implementing the Observer Pattern

Java Files: Stock.java, StockMarket.java, Observer.java, MobileApp.java, WebApp.java,Main.java.

CODE:

Stock.java

```
public interface Stock {
   void registerObserver(Observer o);
   void removeObserver(Observer o);
   void notifyObservers();
}
```

StockMarket.java

```
import java.util.ArrayList;
import java.util.List;
public class StockMarket implements Stock {
   private List<Observer> observers = new ArrayList<>();
   private double price;
   public void setPrice(double newPrice) {
      this.price = newPrice;
      notifyObservers(); // Notify all when price changes
   }
   @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(price);
        }
    }
}
```

Observer.java

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

MobileApp.java

```
public class MobileApp implements Observer {
    private String name;
    public MobileApp(String name) {
        this.name = name;
    }
    @Override
    public void update(double price) {
        System.out.println(name + " Mobile App received update of stock price: $" +
price);
    }
}
```

WebApp.java

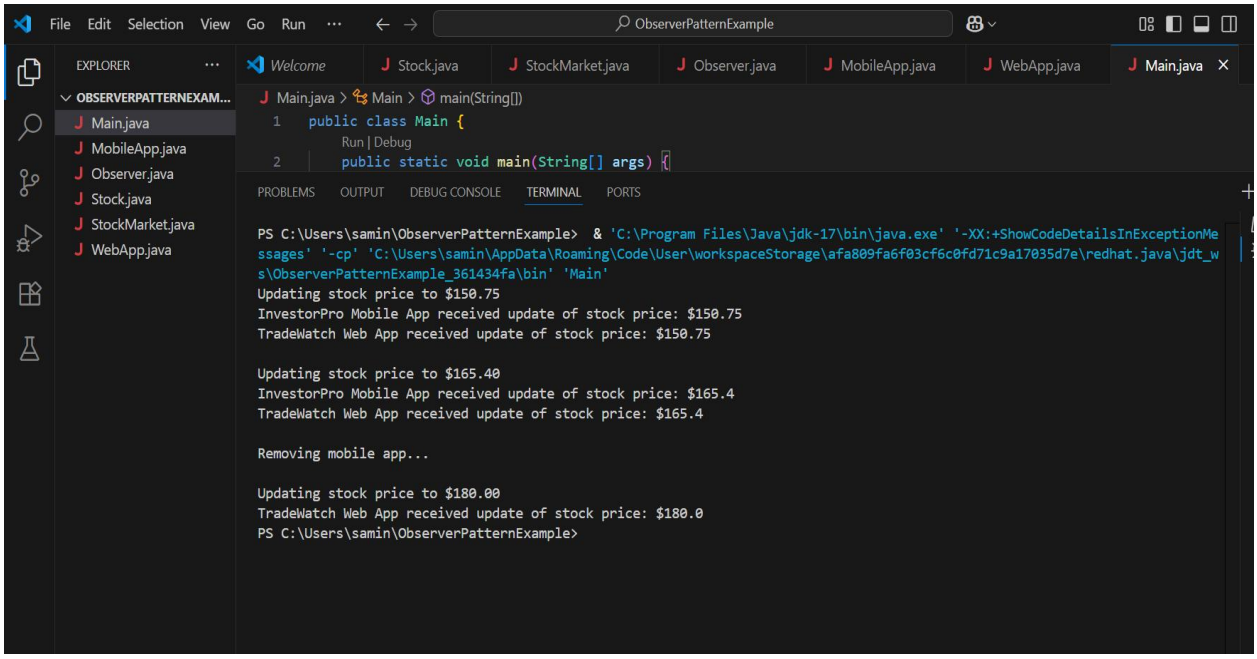
```
public class WebApp implements Observer {
    private String name;
    public WebApp(String name) {
        this.name = name;
    }
    @Override
    public void update(double price) {
        System.out.println(name + " Web App received update of stock price: $" + price);
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        StockMarket stockMarket = new StockMarket();

        Observer mobileApp = new MobileApp("InvestorPro");
        Observer webApp = new WebApp("TradeWatch");

        stockMarket.registerObserver(mobileApp);
        stockMarket.registerObserver(webApp);

        System.out.println("Updating stock price to $150.75");
        stockMarket.setPrice(150.75);

        System.out.println("\nUpdating stock price to $165.40");
```

```
        stockMarket.setPrice(165.40);


        System.out.println("\nRemoving mobile app...");
        stockMarket.removeObserver(mobileApp);


        System.out.println("\nUpdating stock price to $180.00");
        stockMarket.setPrice(180.00);
    }
}
```

OUTPUT:



Exercise 8: Implementing the Strategy Pattern

Java Files: PaymentStrategy.java, CreditCardPayment.java, PayPalPayment.java, PaymentContext.java,Main.java.

CODE:

PaymentStrategy.java

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

CreditCardPayment.java

```
public class CreditCardPayment implements PaymentStrategy {
   private String cardNumber;
   public CreditCardPayment(String cardNumber) {
      this.cardNumber = cardNumber;
   }
   @Override
   public void pay(double amount) {
      System.out.println("Paid $" + amount + " using Credit Card: " + cardNumber);
   }
}
```

PayPalPayment.java

```
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 account: " + email);
    }
}
```

PaymentContext.java

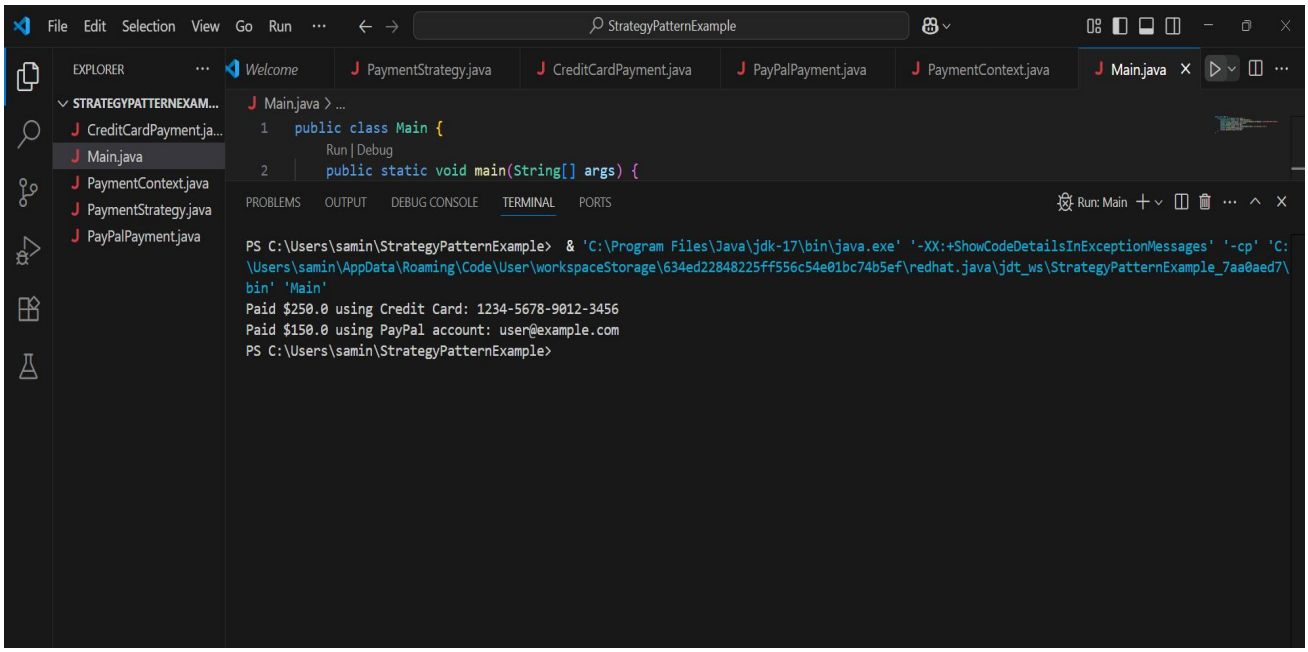
```
public class PaymentContext {
    private PaymentStrategy strategy;
    public void setPaymentStrategy(PaymentStrategy strategy) {
        this.strategy = strategy;
    }
    public void payAmount(double amount) {
        if (strategy == null) {
            System.out.println("No payment strategy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

Main.java

```
public class Main {
    public static void main(String[] args) {
        PaymentContext context = new PaymentContext();
        PaymentStrategy creditCard = new CreditCardPayment("1234-5678-9012-3456");
        context.setPaymentStrategy(creditCard);
        context.payAmount(250.00);
        PaymentStrategy paypal = new PayPalPayment("user@example.com");
```

```
        context.setPaymentStrategy(paypal);

        context.payAmount(150.00);

    }

}
```

OUTPUT:



Exercise 9: Implementing the Command Pattern

Java Files: Command.java, Light.java, LightOnCommand.java, LightOffCommand.java, RemoteControl.java

CODE:

Command.java

```
public interface Command {
   void execute();
}
```

LightOnCommand.java

```
public class LightOnCommand implements Command {
   private Light light;
   public LightOnCommand(Light light) {
      this.light = light;
   }
   @Override
   public void execute() {
      light.turnOn();
   }
}
```

LightOffCommand.java

```
public class LightOffCommand implements Command {
   private Light light;
   public LightOffCommand(Light light) {
      this.light = light;
   }
   @Override
```

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

RemoteControl.java

```
public class RemoteControl {
    private Command command;
    public void setCommand(Command command) {
        this.command = command;
    }
    public void pressButton() {
        if (command != null) {
            command.execute();
        } else {
            System.out.println("No command assigned to button.");
        }
    }
}
```

Light.java

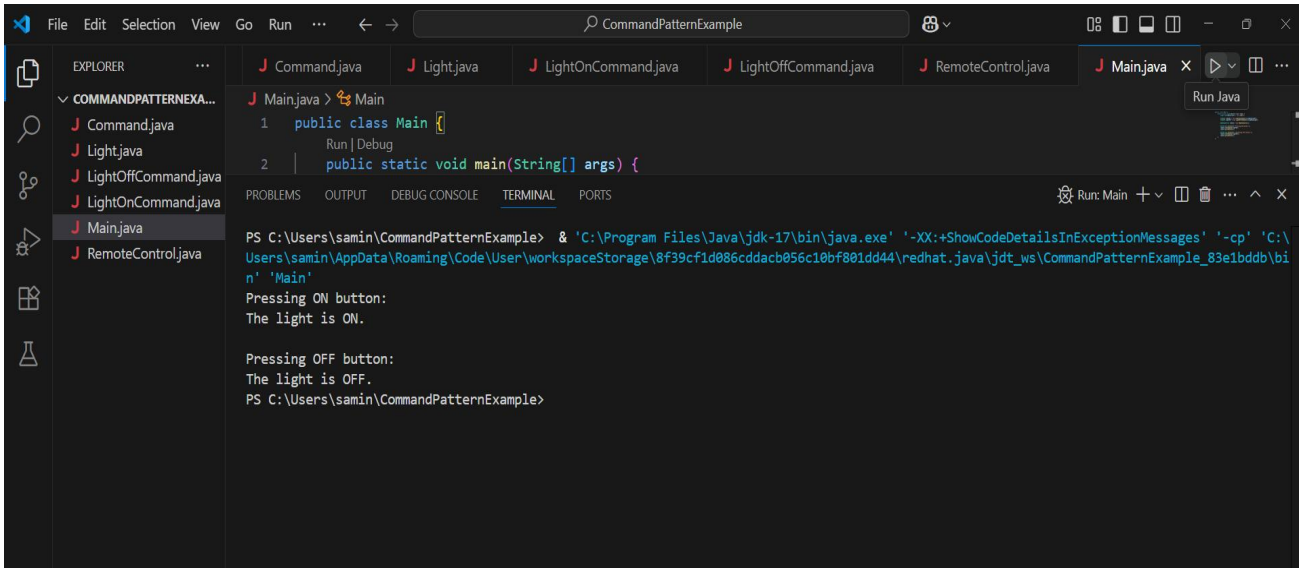
```
public class Light {
    public void turnOn() {
        System.out.println("The light is ON.");
    }
    public void turnOff() {
        System.out.println("The light is OFF.");
    }
```

```
    }
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Light livingRoomLight = new Light();
        Command lightOn = new LightOnCommand(livingRoomLight);
        Command lightOff = new LightOffCommand(livingRoomLight);
        RemoteControl remote = new RemoteControl();
        System.out.println("Pressing ON button:");
        remote.setCommand(lightOn);
        remote.pressButton();
        System.out.println("\nPressing OFF button:");
        remote.setCommand(lightOff);
        remote.pressButton();
    }
}
```

OUTPUT:



Exercise 10: Implementing the MVC Pattern

Java Files: Student.java, StudentView.java, StudentController.java, Main.java.

CODE:

Student.java

```
public class Student {
   private String name;
   private String id;
   private String grade;
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public String getId() {
      return id;
   }
   public void setId(String id) {
      this.id = id;
   }
   public String getGrade() {
      return grade;
   }
   public void setGrade(String grade) {
      this.grade = grade;
   }
```

```
}
```

StudentView.java

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

StudentController.java

```
public class StudentController {
    private Student model;
    private StudentView view;
    public StudentController(Student model, StudentView view) {
        this.model = model;
        this.view = view;
    }
    public void setStudentName(String name) {
        model.setName(name);
    }
    public void setStudentId(String id) {
        model.setId(id);
    }
```

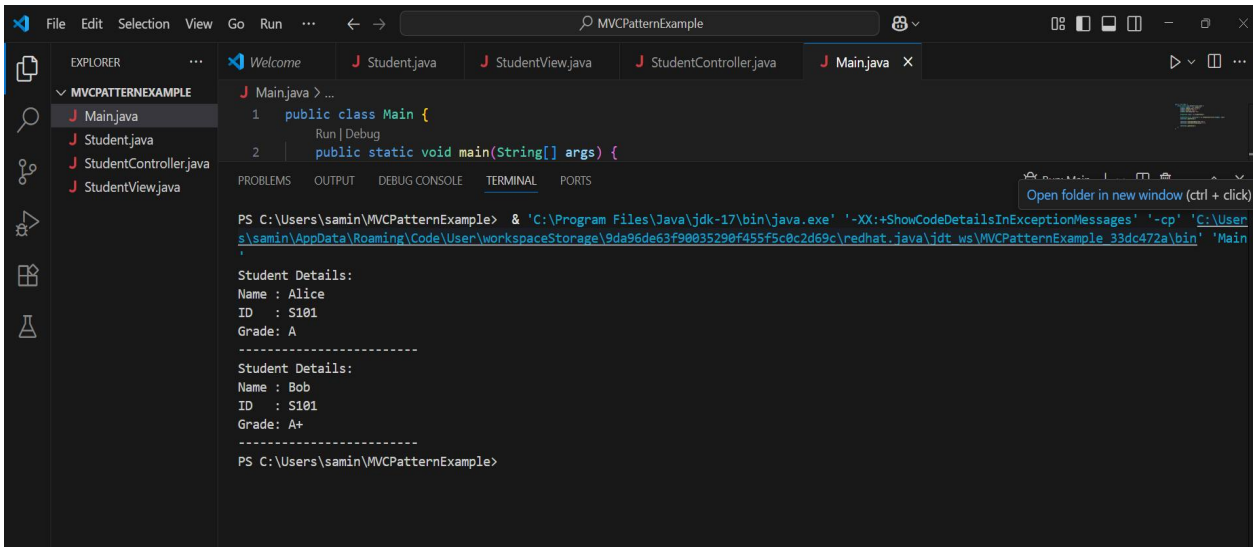
```
    public void setStudentGrade(String grade) {
        model.setGrade(grade);
    }
    public String getStudentName() {
        return model.getName();
    }
    public String getStudentId() {
        return model.getId();
    }
    public String getStudentGrade() {
        return model.getGrade();
    }
    public void updateView() {
        view.displayStudentDetails(model.getName(), model.getId(), model.getGrade());
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Student student = new Student();
        student.setName("Alice");
        student.setId("S101");
        student.setGrade("A");
        StudentView view = new StudentView();
        StudentController controller = new StudentController(student, view);
        controller.updateView();
```

```
        controller.setStudentName("Bob");
        controller.setStudentGrade("A+");
        controller.updateView();
    }
}
```

OUTPUT:



Exercise 11: Implementing Dependency Injection

Java Files: Customer.java, CustomerRepository.java, CustomerRepositoryImpl.java, CustomerService.java, Main.java

CODE:

Customer.java

```
public class Customer {
   private String id;
   private String name;
   public Customer(String id, String name) {
      this.id = id;
      this.name = name;
   }
   public String getId() {
      return id;
   }
   public String getName() {
      return name;
   }
}
```

CustomerRepository.java

```
public interface CustomerRepository {
   Customer findCustomerById(String id);
}
```

CustomerRepositoryImpl.java

```
public class CustomerRepositoryImpl implements CustomerRepository {
   @Override
```

```
    public Customer findCustomerById(String id) {
        return new Customer(id, "John Doe");
    }
}
```

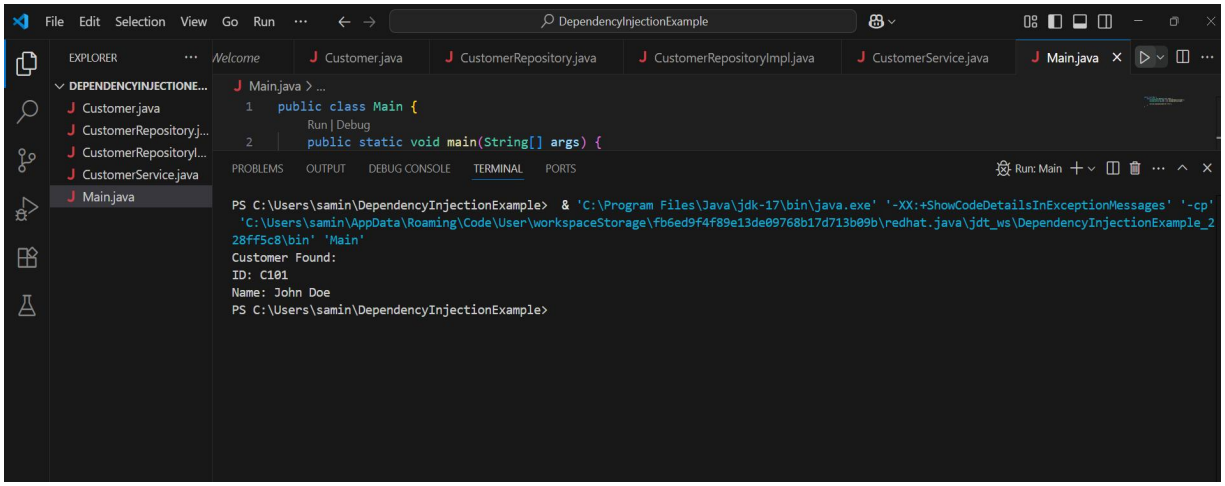
CustomerService.java

```
public class CustomerService {
    private CustomerRepository repository;
    public CustomerService(CustomerRepository repository) {
        this.repository = repository;
    }
    public void displayCustomer(String id) {
        Customer customer = repository.findCustomerById(id);
        System.out.println("Customer Found:");
        System.out.println("ID: " + customer.getId());
        System.out.println("Name: " + customer.getName());
    }
}
```

Main.java

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

OUTPUT:


File Edit Selection View Go Run
DependencyInjectionExample
EXPLORER
DEPENDENCYINJECTIONE...
Customer.java
CustomerRepository.j...
CustomerRepositoryI...
CustomerService.java
Main.java
Welcome
Customer.java
CustomerRepository.java
CustomerRepositoryImpl.java
CustomerService.java
Main.java
Main.java > ...
1 public class Main {
Run | Debug
2 public static void main(String[] args) {
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Run: Main
PS C:\Users\samin\DependencyInjectionExample> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\samin\AppData\Roaming\Code\User\workspaceStorage\fb6ed9f4f89e13de09768b17d713b09b\redhat.java\jdt_ws\DependencyInjectionExample_228ff5c8\bin' 'Main'
Customer Found:
ID: C101
Name: John Doe
PS C:\Users\samin\DependencyInjectionExample>