

DESIGN PRINCIPLES AND DESIGN PATTERNS EXERCISES

Exercise 1: Implementing the Singleton Pattern

```
public class Logger {  
  
    private static Logger instance;  
    private Logger() {  
        System.out.println("Logger initialized.");  
    }  
  
    public static Logger getInstance() {  
        if (instance == null) {  
            instance = new Logger();  
        }  
        return instance;  
    }  
    public void log(String message) {  
        System.out.println("Log: " + message);  
    }  
  
    public static void main(String[] args) {  
  
        Logger logger1 = Logger.getInstance();  
        logger1.log("Logging from logger1");  
  
        Logger logger2 = Logger.getInstance();  
        logger2.log("Logging from logger2");  
  
        if (logger1 == logger2) {  
            System.out.println("✓ logger1 and logger2 are the same instance  
(Singleton works)");  
        } else {  
            System.out.println("✗ Different instances detected (Singleton failed)");  
        }  
    }  
}
```

OUTPUT:

The screenshot shows the IntelliJ IDEA interface with the code editor and run tool window.

Code Editor (Logger.java):

```
1  public class Logger {
2      ...
3      public static Logger getInstance() { 2 usages
4          ...
5          instance = new Logger();
6      }
7      ...
8      return instance;
9  }
10 public void log(String message) { 2 usages
11     System.out.println("Log: " + message);
12 }
13
14 public static void main(String[] args) {
15
16     ...
17     ...
18     ...
19     ...
20     ...
21     ...
22     ...
23     ...
24     ...
25     ...
26     ...
27     ...
28     ...
29 }
```

Run Tool Window (Output):

```
C:\Users\LENOVO\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=53632" -Dfile.encoding=UTF-8
Logger initialized.
Log: Logging from logger1
Log: Logging from logger2
logger1 and logger2 are the same instance (Singleton works)
Process finished with exit code 0
```

22:49 CRLF UTF-8 4 spaces

Exercise 2: Implementing the Factory Method Pattern

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

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

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

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

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

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

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

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

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:

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Bar:** Shows "Main.java" as the current file.
- Code Editor:** Displays Java code for the Factory Method pattern. It includes classes like `ExcelDocument`, `WordDocumentFactory`, and `PdfDocumentFactory`.
- Run Tab:** Shows the command used to run the application: `C:\Users\LENOVO\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=53670" -Dfile.encoding=UTF-8`.
- Output Tab:** Shows the output of the application: "Open Word Document..." and "Process finished with exit code 0".
- Status Bar:** Shows file statistics: 291 CRLF, UTF-8, 4 spaces.

Exercise 3: Implementing the Builder Pattern

```
public class Main {

    static class Computer {
        private final String CPU;
        private final String RAM;
        private final String storage;
        private final String graphicsCard;

        private Computer(Builder builder) {
            this.CPU = builder.CPU;
            this.RAM = builder.RAM;
            this.storage = builder.storage;
            this.graphicsCard = builder.graphicsCard;
        }

        public void showConfig() {
            System.out.println("Computer Configuration:");
            System.out.println("CPU: " + CPU);
            System.out.println("RAM: " + RAM);
            System.out.println("Storage: " + storage);
            System.out.println("Graphics Card: " + graphicsCard);
            System.out.println();
        }
    }

    static class Builder {

```

```
private String CPU;
private String RAM;
private String storage;
private String graphicsCard;

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;
}

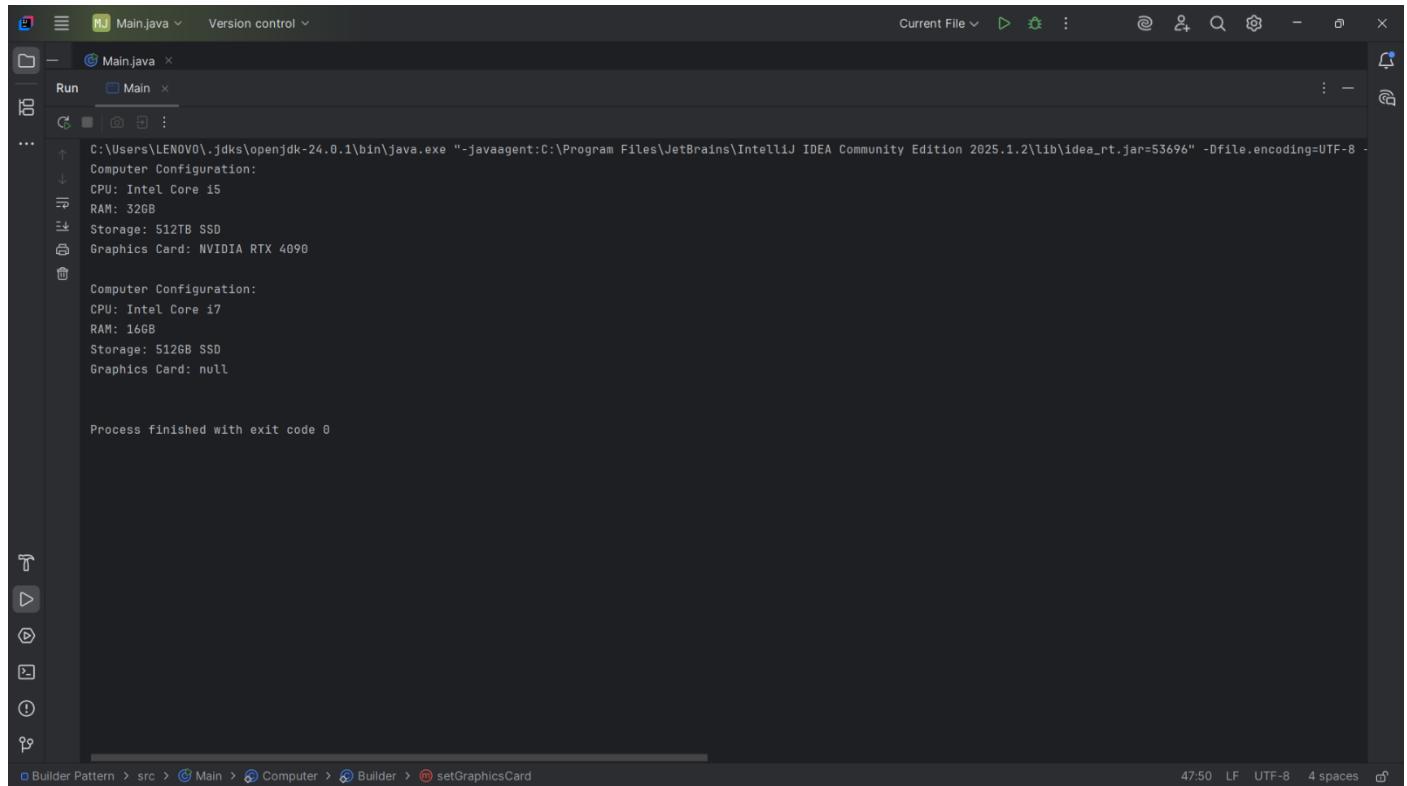
public Computer build() {
    return new Computer(this);
}
}

public static void main(String[] args) {
    Computer gamingPC = new Computer.Builder()
        .setCPU("Intel Core i5")
        .setRAM("32GB")
        .setStorage("512TB SSD")
        .setGraphicsCard("NVIDIA RTX 4090")
        .build();

    Computer officePC = new Computer.Builder()
        .setCPU("Intel Core i7")
        .setRAM("16GB")
        .setStorage("512GB SSD")
        .build();

    gamingPC.showConfig();
    officePC.showConfig();
}
}
```

OUTPUT:



```
C:\Users\LENOVO\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=53696" -Dfile.encoding=UTF-8 -  
Computer Configuration:  
CPU: Intel Core i5  
RAM: 32GB  
Storage: 512TB SSD  
Graphics Card: NVIDIA RTX 4090  
  
Computer Configuration:  
CPU: Intel Core i7  
RAM: 16GB  
Storage: 512GB SSD  
Graphics Card: null  
  
Process finished with exit code 0
```

Exercise 4: Implementing the Adapter Pattern

```
public class Main {  
  
    interface PaymentProcessor {  
        void processPayment(double amount);  
    }  
  
    static class GPayGateway {  
        void makePayment(double amount) {  
            System.out.println("Processing payment of ₹" + amount + " through GPay.");  
        }  
    }  
  
    static class PhonePeGateway {  
        void doTransaction(double amount) {  
            System.out.println("Processing payment of ₹" + amount + " through PhonePe.");  
        }  
    }  
  
    static class GPayAdapter implements PaymentProcessor {  
        private GPayGateway gPayGateway;  
  
        public GPayAdapter(GPayGateway gPayGateway) {  
            this.gPayGateway = gPayGateway;  
        }  
  
        public void processPayment(double amount) {  
            gPayGateway.makePayment(amount);  
        }  
    }  
}
```

```

        gPayGateway.makePayment(amount);
    }

}

static class PhonePeAdapter implements PaymentProcessor {
    private PhonePeGateway phonePeGateway;

    public PhonePeAdapter(PhonePeGateway phonePeGateway) {
        this.phonePeGateway = phonePeGateway;
    }

    public void processPayment(double amount) {
        phonePeGateway.doTransaction(amount);
    }
}

public static void main(String[] args) {
    PaymentProcessor gpay = new GPayAdapter(new GPayGateway());
    gpay.processPayment(320.00);

    PaymentProcessor phonePe = new PhonePeAdapter(new PhonePeGateway());
    phonePe.processPayment(474.50);
}
}

```

OUTPUT:

The screenshot shows the IntelliJ IDEA interface with the Main.java file open. The terminal window displays the following output:

```

C:\Users\LENOVO\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=53753" -Dfile.encoding=UTF-8 -jar C:\Users\LENOVO\IdeaProjects\Adapter Pattern\src\Main.jar
Processing payment of ₹320.0 through GPay.
Processing payment of ₹474.5 through PhonePe.

Process finished with exit code 0

```

Exercise 5: Implementing the Decorator Pattern

```
public class Main {

    interface Notifier {
        void send(String message);
    }

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

    static abstract class NotifierDecorator implements Notifier {
        protected Notifier notifier;

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

        public void send(String message) {
            notifier.send(message);
        }
    }

    static class SMSNotifierDecorator extends NotifierDecorator {
        public SMSNotifierDecorator(Notifier notifier) {
            super(notifier);
        }

        public void send(String message) {
            super.send(message);
            System.out.println("Sending SMS: " + message);
        }
    }

    static class SlackNotifierDecorator extends NotifierDecorator {
        public SlackNotifierDecorator(Notifier notifier) {
            super(notifier);
        }

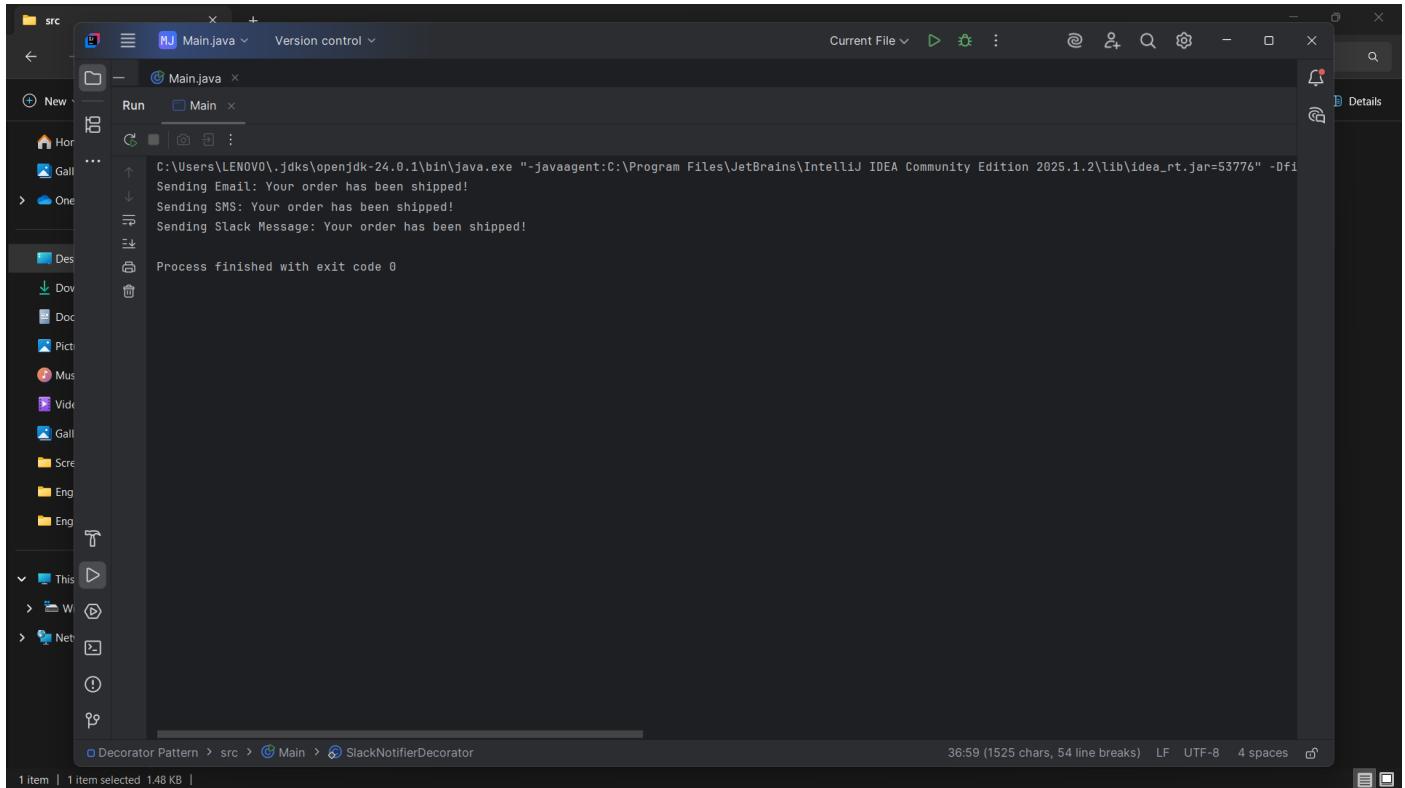
        public void send(String message) {
            super.send(message);
            System.out.println("Sending Slack Message: " + message);
        }
    }

    public static void main(String[] args) {
        Notifier notifier = new EmailNotifier();
        Notifier smsNotifier = new SMSNotifierDecorator(notifier);
        Notifier slackAndSmsNotifier = new SlackNotifierDecorator(smsNotifier);

        slackAndSmsNotifier.send("Your order has been shipped!");
    }
}
```

```
}
```

OUTPUT:



The screenshot shows the IntelliJ IDEA interface with a dark theme. The left sidebar displays a file tree with a 'src' folder containing 'Main.java'. The main editor window shows the Java code for 'Main.java'. The bottom status bar indicates the file has 36.59 lines, 1525 characters, and is saved in UTF-8 with 4 spaces.

```
C:\Users\LENOVO\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=53776" -Dfile.encoding=UTF-8
Sending Email: Your order has been shipped!
Sending SMS: Your order has been shipped!
Sending Slack Message: Your order has been shipped!

Process finished with exit code 0
```

Exercise 6: Implementing the Proxy Pattern

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

    static class RealImage implements Image {
        private final String filename;

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

        private void loadFromServer() {
            System.out.println("Loading " + filename + " from server...");
        }

        @Override
        public void display() {
            System.out.println("Displaying " + filename);
        }
    }
}
```

```

        }

    static class ProxyImage implements Image {
        private final String filename;
        private RealImage realImage;

        public ProxyImage(String filename) {
            this.filename = filename;
        }

        @Override
        public void display() {
            if (realImage == null) {
                realImage = new RealImage(filename);
            }
            realImage.display();
        }
    }

    public static void main(String[] args) {
        Image image1 = new ProxyImage("image1.jpg");
        image1.display(); // Loads from server
        image1.display(); // Uses cache

        Image image2 = new ProxyImage("image2.jpg");
        image2.display(); // Loads from server
    }
}

```

OUTPUT:

The screenshot shows the IntelliJ IDEA interface with the Run tool window open. The tool window has a title bar with tabs for 'Main.java' and 'Version control'. Below the title bar is a toolbar with various icons. The main area of the tool window displays the execution log for the 'Main' run configuration. The log shows the following output:

```

...
↑ C:\Users\LENOVO\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=53806" -Dfile.encoding=UTF-8
↓ Loading image1.jpg from server...
→ Displaying image1.jpg
→ Displaying image1.jpg
↓ Loading image2.jpg from server...
→ Displaying image2.jpg
Process finished with exit code 0
|
```

The 'Run' tab is selected in the left sidebar of the tool window. The bottom status bar indicates the current file path as 'Proxy Pattern Example > src > Main' and the file encoding as 'UTF-8'.

Exercise 7: Implementing the Observer Pattern

```
import java.util.*;  
  
public class Main {  
    interface Observer {  
        void update(String stockName, double price);  
    }  
  
    interface Stock {  
        void register(Observer o);  
        void deregister(Observer o);  
        void notifyObservers();  
        void setPrice(double price);  
    }  
  
    static class StockMarket implements Stock {  
        private final String stockName;  
        private double price;  
        private final List<Observer> observers = new ArrayList<>();  
  
        public StockMarket(String stockName, double initialPrice) {  
            this.stockName = stockName;  
            this.price = initialPrice;  
        }  
  
        @Override  
        public void register(Observer o) {  
            observers.add(o);  
        }  
  
        @Override  
        public void deregister(Observer o) {  
            observers.remove(o);  
        }  
  
        @Override  
        public void notifyObservers() {  
            for (Observer o : observers) {  
                o.update(stockName, price);  
            }  
        }  
  
        @Override  
        public void setPrice(double price) {  
            this.price = price;  
            notifyObservers();  
        }  
    }  
  
    static class MobileApp implements Observer {  
        private final String user;
```

```
public MobileApp(String user) {
    this.user = user;
}

@Override
public void update(String stockName, double price) {
    System.out.println("MobileApp " + user + " notified: " + stockName + " price
is now ₹" + price);
}

static class WebApp implements Observer {
    private final String user;

    public WebApp(String user) {
        this.user = user;
    }

    @Override
    public void update(String stockName, double price) {
        System.out.println("WebApp " + user + " notified: " + stockName + " price is
now ₹" + price);
    }
}

public static void main(String[] args) {
    StockMarket appleStock = new StockMarket("Apple", 150.0);

    Observer mobileUser = new MobileApp("Dharmesh");
    Observer webUser = new WebApp("Sriram");

    appleStock.register(mobileUser);
    appleStock.register(webUser);

    appleStock.setPrice(152.5);
    appleStock.setPrice(155.0);

    appleStock.deregister(webUser);
    appleStock.setPrice(158.0);
}
}
```

OUTPUT:

```
import java.util.*;  
public class Main {  
    interface Observer {  
        void update(String stockName, double price);  
    }  
    interface Stock {  
        void register(Observer o);  
        void deregister(Observer o);  
        void notifyObservers();  
        void setPrice(double price);  
    }  
    static class StockMarket implements Stock {  
        private final String stockName;  
        private double price;  
        private final List<Observer> observers = new ArrayList<>();  
        public StockMarket(String stockName, double initialPrice) {  
            this.stockName = stockName;  
        }  
        void register(Observer o) {  
            observers.add(o);  
        }  
        void deregister(Observer o) {  
            observers.remove(o);  
        }  
        void notifyObservers() {  
            for (Observer observer : observers) {  
                observer.update(stockName, price);  
            }  
        }  
        void setPrice(double price) {  
            this.price = price;  
            notifyObservers();  
        }  
    }  
}  
MobileApp Dharmesh notified: Apple price is now ₹152.5  
WebApp Sriram notified: Apple price is now ₹152.5  
MobileApp Dharmesh notified: Apple price is now ₹155.0  
WebApp Sriram notified: Apple price is now ₹155.0  
MobileApp Dharmesh notified: Apple price is now ₹158.0  
Process finished with exit code 0
```

Exercise 8: Implementing the Strategy Pattern

```
interface PaymentStrategy {  
    void pay(int amount);  
}  
  
class CreditCardPayment implements PaymentStrategy {  
    private String cardNumber;  
  
    public CreditCardPayment(String cardNumber) {  
        this.cardNumber = cardNumber;  
    }  
  
    public void pay(int amount) {  
        System.out.println("Paid ₹" + amount + " using Credit Card: " + cardNumber);  
    }  
}  
  
class GPayPayment implements PaymentStrategy {  
    private String email;  
  
    public GPayPayment(String email) {  
        this.email = email;  
    }  
  
    public void pay(int amount) {  
        System.out.println("Paid ₹" + amount + " using GPay account: " + email);  
    }  
}
```

```

class PaymentContext {
    private PaymentStrategy strategy;

    public void setPaymentStrategy(PaymentStrategy strategy) {
        this.strategy = strategy;
    }

    public void pay(int amount) {
        strategy.pay(amount);
    }
}

public class Main {
    public static void main(String[] args) {
        PaymentContext context = new PaymentContext();

        context.setPaymentStrategy(new CreditCardPayment("1148 7458 6321 4862"));
        context.pay(500);

        context.setPaymentStrategy(new GPayPayment("ditto143@gmail.com"));
        context.pay(300);
    }
}

```

OUTPUT:

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure with a file named "Main.java" selected.
- Main.java Content:**

```

class PaymentContext {
    private PaymentStrategy strategy;

    public void setPaymentStrategy(PaymentStrategy strategy) {
        this.strategy = strategy;
    }

    public void pay(int amount) {
        strategy.pay(amount);
    }
}

public class Main {
    public static void main(String[] args) {
        PaymentContext context = new PaymentContext();

        context.setPaymentStrategy(new CreditCardPayment("1148 7458 6321 4862"));
        context.pay(500);

        context.setPaymentStrategy(new GPayPayment("ditto143@gmail.com"));
        context.pay(300);
    }
}

```
- Run Tab:** Displays the command-line output of the application's execution.

Exercise 9: Implementing the Command Pattern

```

interface Command {
    void execute();
}

```

```
}

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

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

class LightOnCommand implements Command {
    private Light light;

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

    public void execute() {
        light.turnOn();
    }
}

class LightOffCommand implements Command {
    private Light light;

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

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

class RemoteControl {
    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }

    public void pressButton() {
        command.execute();
    }
}

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();

        remote.setCommand(lightOn);
        remote.pressButton();

        remote.setCommand(lightOff);
        remote.pressButton();
    }
}

```

OUTPUT:

The screenshot shows the IntelliJ IDEA interface with the Main.java file open. The code implements the Command design pattern. The run output window shows the expected behavior: the light is turned on and then turned off. The status bar at the bottom right shows the process finished with an exit code of 0.

```

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();

        remote.setCommand(lightOn);
        remote.pressButton();

        remote.setCommand(lightOff);
        remote.pressButton();
    }
}

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

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

class LightOnCommand implements Command {
    private Light light;

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

    public void execute() {
        light.turnOn();
    }
}

class LightOffCommand implements Command {
    private Light light;

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

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

interface Command {
    void execute();
}

```

Exercise 10: Implementing the MVC Pattern

```

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;
}
}

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);
    }
}

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 String getStudentName() {
        return model.getName();
    }

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

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

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

```

}

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

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

public class Main {
    public static void main(String[] args) {
        Student model = new Student();
        model.setName("Dharmesh S");
        model.setId("ST1711");
        model.setGrade("A");

        StudentView view = new StudentView();
        StudentController controller = new StudentController(model, view);

        controller.updateView();

        controller.setStudentGrade("A+");
        controller.setStudentName("Dharmesh S ");

        controller.updateView();
    }
}

```

OUTPUT:

```

C:\Users\LENOVO\jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=55735" -Dfile.encoding=UTF-8
Student Details:
Name: Dharmesh S
ID: ST1711
Grade: A
Student Details:
Name: Dharmesh S
ID: ST1711
Grade: A+
Process finished with exit code 0

```

Exercise 11: Implementing Dependency Injection

```
interface CustomerRepository {  
    String findCustomerById(String id);  
}  
  
class CustomerRepositoryImpl implements CustomerRepository {  
    public String findCustomerById(String id) {  
        return "Customer ID: " + id + ", Name: Dharmesh Sriram";  
    }  
}  
  
class CustomerService {  
    private CustomerRepository customerRepository;  
  
    public CustomerService(CustomerRepository customerRepository) {  
        this.customerRepository = customerRepository;  
    }  
  
    public void displayCustomer(String id) {  
        String customer = customerRepository.findCustomerById(id);  
        System.out.println(customer);  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        CustomerRepository repository = new CustomerRepositoryImpl();  
        CustomerService service = new CustomerService(repository);  
        service.displayCustomer("CUST1711");  
    }  
}
```

OUTPUT:

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure for "Dependency Injection Example" located at C:\Users\LENOVO\OneDrive\... . It includes folders like .idea, out, src, .gitignore, and files like Dependency Injection Example.iml, External Libraries, and Scratches and Consoles.
- Main.java Editor:** Displays the Java code for the Main class:

```
11  class CustomerService { 12      public void displayCustomer(String id) { 13          String customer = customerRepository.findCustomerById(id); 14          System.out.println(customer); 15      } 16  } 17  public class Main { 18      public static void main(String[] args) { 19          CustomerRepository repository = new CustomerRepositoryImpl(); 20          CustomerService service = new CustomerService(repository); 21          service.displayCustomer("CUST1711"); 22      } 23  } 24 }
```
- Run Tab:** Shows the run configuration for "Main".
- Terminal Output:** Shows the execution results:

```
C:\Users\LENOVO\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=55761" -Dfile.encoding=UTF-8
Customer ID: CUST1711, Name: Dharmesh Srinam
Process finished with exit code 0
```
- Status Bar:** Shows the file path "Dependency Injection Example > src > Main.java > main", and the status "28:42 LF UTF-8 4 spaces".