Design Patterns and Principles

Exercise 1: Implementing the Singleton Pattern

Logger.java

java

```java
public class Logger {
    private static Logger instance;
    private StringBuilder logMessages;

    private Logger() {
        logMessages = new StringBuilder();
        System.out.println("Logger instance created!");
    }

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

    public void log(String message) {
        logMessages.append("[" + java.time.LocalTime.now() + "] " + message + "\n");
        System.out.println("Logged: " + message);
    }

    public void printAllLogs() {
        System.out.println("=== All Logs ===");
        System.out.println(logMessages.toString());
    }
}
```

SingletonTest.java

java

```java
public class SingletonTest {
    public static void main(String[] args) {
        Logger logger1 = Logger.getInstance();
        logger1.log("First log message");

        Logger logger2 = Logger.getInstance();
        logger2.log("Second log message");

        System.out.println("Same instance? " + (logger1 == logger2));

        logger1.log("Message from logger1");
        logger2.log("Message from logger2");

        logger1.printAllLogs();
    }
}
```

Sample Output:

```
Logger instance created!
Logged: First log message
Logged: Second log message
Same instance? true
Logged: Message from logger1
Logged: Message from logger2
=== All Logs ===
[10:15:23] First log message
[10:15:23] Second log message
[10:15:23] Message from logger1
[10:15:23] Message from logger2
```

Exercise 2: Implementing the Factory Method Pattern

Document.java

java

```java
public abstract class Document {
    protected String type;
    protected String content;

    public abstract void open();
    public abstract void save();

    public String getType() {
        return type;
    }
}
```

WordDocument.java

java

```java
public class WordDocument extends Document {
    public WordDocument() {
        this.type = "Word Document";
        this.content = "";
    }

    @Override
    public void open() {
        System.out.println("Opening Word document with Microsoft Word");
    }

    @Override
    public void save() {
        System.out.println("Saving Word document as .docx file");
    }
}
```

PdfDocument.java

java

```java
public class PdfDocument extends Document {
    public PdfDocument() {
        this.type = "PDF Document";
        this.content = "";
    }

    @Override
    public void open() {
        System.out.println("Opening PDF document with PDF Reader");
    }

    @Override
    public void save() {
        System.out.println("Saving PDF document as .pdf file");
    }
}
```

ExcelDocument.java

java

```java
public class ExcelDocument extends Document {
    public ExcelDocument() {
        this.type = "Excel Document";
        this.content = "";
    }

    @Override
    public void open() {
        System.out.println("Opening Excel document with Microsoft Excel");
    }

    @Override
    public void save() {
        System.out.println("Saving Excel document as .xlsx file");
    }
}
```

DocumentFactory.java

java

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

    public void processDocument() {
        Document doc = createDocument();
        System.out.println("Created: " + doc.getType());
        doc.open();
        doc.save();
    }
}
```

WordDocumentFactory.java

java

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

PdfDocumentFactory.java

java

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

ExcelDocumentFactory.java

java

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

FactoryTest.java

java

```java
public class FactoryTest {
    public static void main(String[] args) {
        System.out.println("=== Factory Method Pattern Demo ===\n");

        DocumentFactory wordFactory = new WordDocumentFactory();
        wordFactory.processDocument();

        System.out.println();

        DocumentFactory pdfFactory = new PdfDocumentFactory();
        pdfFactory.processDocument();

        System.out.println();

        DocumentFactory excelFactory = new ExcelDocumentFactory();
        excelFactory.processDocument();
    }
}
```

Sample Output:

```
=== Factory Method Pattern Demo ===

Created: Word Document
Opening Word document with Microsoft Word
Saving Word document as .docx file

Created: PDF Document
Opening PDF document with PDF Reader
Saving PDF document as .pdf file

Created: Excel Document
Opening Excel document with Microsoft Excel
Saving Excel document as .xlsx file
```

Exercise 3: Implementing the Builder Pattern

Computer.java

```java
public class Computer {
    private String cpu;
    private String ram;
    private String storage;
    private String gpu;
    private String motherboard;
    private boolean isWifiEnabled;

    private Computer(Builder builder) {
        this.cpu = builder.cpu;
        this.ram = builder.ram;
        this.storage = builder.storage;
        this.gpu = builder.gpu;
        this.motherboard = builder.motherboard;
        this.isWifiEnabled = builder.isWifiEnabled;
    }

    public static class Builder {
        private String cpu;
        private String ram;
        private String storage;
        private String gpu = "Integrated";
        private String motherboard = "Standard";
        private boolean isWifiEnabled = false;

        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 setGpu(String gpu) {
            this.gpu = gpu;
            return this;
        }

        public Builder setMotherboard(String motherboard) {
            this.motherboard = motherboard;
            return this;
        }
```

```
        public Builder enableWifi(boolean isWifiEnabled) {
            this.isWifiEnabled = isWifiEnabled;
            return this;
        }

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

    @Override
    public String toString() {
        return "Computer Configuration:\n" +
               "CPU: " + cpu + "\n" +
               "RAM: " + ram + "\n" +
               "Storage: " + storage + "\n" +
               "GPU: " + gpu + "\n" +
               "Motherboard: " + motherboard + "\n" +
               "WiFi: " + (isWifiEnabled ? "Enabled" : "Disabled");
    }
}
```

BuilderTest.java

java

```java
public class BuilderTest {
    public static void main(String[] args) {
        System.out.println("=== Builder Pattern Demo ===\n");

        Computer gamingPC = new Computer.Builder()
                .setCpu("Intel i9-11900K")
                .setRam("32GB DDR4")
                .setStorage("1TB NVMe SSD")
                .setGpu("RTX 3080")
                .setMotherboard("ASUS ROG Strix")
                .enableWifi(true)
                .build();

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

        System.out.println("\n" + "=".repeat(40) + "\n");

        Computer officePC = new Computer.Builder()
                .setCpu("Intel i5-11400")
                .setRam("16GB DDR4")
                .setStorage("512GB SSD")
                .build();

        System.out.println("Office PC:");
        System.out.println(officePC);

        System.out.println("\n" + "=".repeat(40) + "\n");

        Computer budgetPC = new Computer.Builder()
                .setCpu("AMD Ryzen 5 3600")
                .setRam("8GB DDR4")
                .setStorage("256GB SSD")
                .enableWifi(true)
                .build();

        System.out.println("Budget PC:");
        System.out.println(budgetPC);
    }
}
```

Sample Output:

```
=== Builder Pattern Demo ===

Gaming PC:
Computer Configuration:
CPU: Intel i9-11900K
RAM: 32GB DDR4
Storage: 1TB NVMe SSD
GPU: RTX 3080
Motherboard: ASUS ROG Strix
WiFi: Enabled

========================================

Office PC:
Computer Configuration:
CPU: Intel i5-11400
RAM: 16GB DDR4
Storage: 512GB SSD
GPU: Integrated
Motherboard: Standard
WiFi: Disabled

========================================

Budget PC:
Computer Configuration:
CPU: AMD Ryzen 5 3600
RAM: 8GB DDR4
Storage: 256GB SSD
GPU: Integrated
Motherboard: Standard
WiFi: Enabled
```

Exercise 4: Implementing the Adapter Pattern

PaymentProcessor.java

java

```java
public interface PaymentProcessor {
    boolean processPayment(double amount, String currency);
    String getPaymentStatus();
}
```

PayPalGateway.java

java

```java
public class PayPalGateway {
    public boolean makePayment(double amount) {
        System.out.println("Processing $" + amount + " through PayPal");
        System.out.println("PayPal payment successful!");
        return true;
    }

    public String checkStatus() {
        return "PayPal Transaction Completed";
    }
}
```

StripeGateway.java

java

```java
public class StripeGateway {
    public boolean charge(double amount, String curr) {
        System.out.println("Charging " + amount + " " + curr + " via Stripe");
        System.out.println("Stripe payment processed!");
        return true;
    }

    public String getTransactionStatus() {
        return "Stripe Payment Success";
    }
}
```

PayPalAdapter.java

java

```java
public class PayPalAdapter implements PaymentProcessor {
    private PayPalGateway paypalGateway;

    public PayPalAdapter(PayPalGateway paypalGateway) {
        this.paypalGateway = paypalGateway;
    }

    @Override
    public boolean processPayment(double amount, String currency) {
        return paypalGateway.makePayment(amount);
    }

    @Override
    public String getPaymentStatus() {
        return paypalGateway.checkStatus();
    }
}
```

StripeAdapter.java

java

```java
public class StripeAdapter implements PaymentProcessor {
    private StripeGateway stripeGateway;

    public StripeAdapter(StripeGateway stripeGateway) {
        this.stripeGateway = stripeGateway;
    }

    @Override
    public boolean processPayment(double amount, String currency) {
        return stripeGateway.charge(amount, currency);
    }

    @Override
    public String getPaymentStatus() {
        return stripeGateway.getTransactionStatus();
    }
}
```

AdapterTest.java

java

```java
public class AdapterTest {
    public static void main(String[] args) {
        System.out.println("=== Adapter Pattern Demo ===\n");

        PaymentProcessor paypalProcessor = new PayPalAdapter(new PayPalGateway());
        processPayment(paypalProcessor, 99.99, "USD");

        System.out.println("\n" + "-".repeat(40) + "\n");

        PaymentProcessor stripeProcessor = new StripeAdapter(new StripeGateway());
        processPayment(stripeProcessor, 149.50, "EUR");
    }

    private static void processPayment(PaymentProcessor processor, double amount, String curren
        System.out.println("Initiating payment...");
        boolean success = processor.processPayment(amount, currency);

        if (success) {
            System.out.println("Status: " + processor.getPaymentStatus());
        } else {
            System.out.println("Payment failed!");
        }
    }
}
```

Sample Output:

```
=== Adapter Pattern Demo ===

Initiating payment...
Processing $99.99 through PayPal
PayPal payment successful!
Status: PayPal Transaction Completed

----------------------------------------

Initiating payment...
Charging 149.5 EUR via Stripe
Stripe payment processed!
Status: Stripe Payment Success
```

Exercise 5: Implementing the Decorator Pattern

Notifier.java

java

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

EmailNotifier.java

java

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

NotifierDecorator.java

java

```java
public abstract class NotifierDecorator implements Notifier {
    protected Notifier notifier;

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

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

SMSNotifierDecorator.java

java

```java
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("SMS: " + message);
    }
}
```

SlackNotifierDecorator.java

java

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

    @Override
    public void send(String message) {
        super.send(message);
        sendSlack(message);
    }

    private void sendSlack(String message) {
        System.out.println("Slack: " + message);
    }
}
```

DecoratorTest.java

java

```java
public class DecoratorTest {
    public static void main(String[] args) {
        System.out.println("=== Decorator Pattern Demo ===\n");

        Notifier emailNotifier = new EmailNotifier();
        System.out.println("Basic Email Notification:");
        emailNotifier.send("Welcome to our service!");

        System.out.println("\n" + "-".repeat(40) + "\n");

        Notifier emailAndSMS = new SMSNotifierDecorator(emailNotifier);
        System.out.println("Email + SMS Notification:");
        emailAndSMS.send("Your order has been shipped!");

        System.out.println("\n" + "-".repeat(40) + "\n");

        Notifier allChannels = new SlackNotifierDecorator(
            new SMSNotifierDecorator(emailNotifier)
        );
        System.out.println("Email + SMS + Slack Notification:");
        allChannels.send("System maintenance scheduled!");
    }
}
```

Sample Output:

```
=== Decorator Pattern Demo ===

Basic Email Notification:
Email: Welcome to our service!

----------------------------------------

Email + SMS Notification:
Email: Your order has been shipped!
SMS: Your order has been shipped!

----------------------------------------

Email + SMS + Slack Notification:
Email: System maintenance scheduled!
SMS: System maintenance scheduled!
Slack: System maintenance scheduled!
```

Exercise 6: Implementing the Proxy Pattern

Image.java

java

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

RealImage.java

java

```java
public class RealImage implements Image {
    private String filename;

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

    private void loadFromServer() {
        System.out.println("Loading " + filename + " from remote server...");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        System.out.println(filename + " loaded successfully!");
    }

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

ProxyImage.java

java

```java
import java.util.HashMap;
import java.util.Map;

public class ProxyImage implements Image {
    private String filename;
    private RealImage realImage;
    private static Map<String, RealImage> cache = new HashMap<>();

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

    @Override
    public void display() {
        if (cache.containsKey(filename)) {
            System.out.println("Loading " + filename + " from cache...");
            realImage = cache.get(filename);
        } else {
            realImage = new RealImage(filename);
            cache.put(filename, realImage);
        }
        realImage.display();
    }
}
```

ProxyTest.java

java

```java
public class ProxyTest {
    public static void main(String[] args) {
        System.out.println("=== Proxy Pattern Demo ===\n");

        Image image1 = new ProxyImage("photo1.jpg");
        Image image2 = new ProxyImage("photo2.jpg");
        Image image3 = new ProxyImage("photo1.jpg");

        System.out.println("First time accessing photo1.jpg:");
        image1.display();

        System.out.println("\n" + "-".repeat(40) + "\n");

        System.out.println("First time accessing photo2.jpg:");
        image2.display();

        System.out.println("\n" + "-".repeat(40) + "\n");

        System.out.println("Second time accessing photo1.jpg:");
        image3.display();
    }
}
```

Sample Output:

```
=== Proxy Pattern Demo ===

First time accessing photo1.jpg:
Loading photo1.jpg from remote server...
photo1.jpg loaded successfully!
Displaying photo1.jpg

----------------------------------------

First time accessing photo2.jpg:
Loading photo2.jpg from remote server...
photo2.jpg loaded successfully!
Displaying photo2.jpg

----------------------------------------

Second time accessing photo1.jpg:
Loading photo1.jpg from cache...
Displaying photo1.jpg
```

Exercise 7: Implementing the Observer Pattern

Observer.java

java

```java
public interface Observer {
    void update(String stockName, double price);
}
```

Stock.java

java

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

StockMarket.java

java

```java
import java.util.ArrayList;
import java.util.List;

public class StockMarket implements Stock {
    private List<Observer> observers;
    private String stockName;
    private double stockPrice;

    public StockMarket() {
        observers = new ArrayList<>();
    }

    @Override
    public void registerObserver(Observer observer) {
        observers.add(observer);
        System.out.println("Observer registered for stock updates");
    }

    @Override
    public void removeObserver(Observer observer) {
        observers.remove(observer);
        System.out.println("Observer removed from stock updates");
    }

    @Override
    public void notifyObservers() {
        System.out.println("Notifying all observers about " + stockName + " price change to $"
        for (Observer observer : observers) {
            observer.update(stockName, stockPrice);
        }
    }

    public void setStockPrice(String stockName, double stockPrice) {
        this.stockName = stockName;
        this.stockPrice = stockPrice;
        notifyObservers();
    }
}
```

MobileApp.java

java

```java
public class MobileApp implements Observer {
    private String appName;

    public MobileApp(String appName) {
        this.appName = appName;
    }

    @Override
    public void update(String stockName, double price) {
        System.out.println("[" + appName + " Mobile App] " + stockName + " is now $" + price);
        System.out.println("[" + appName + "] Sending push notification to user");
    }
}
```

WebApp.java

java

```java
public class WebApp implements Observer {
    private String websiteName;

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

    @Override
    public void update(String stockName, double price) {
        System.out.println("[" + websiteName + " Website] " + stockName + " updated to $" + pri
        System.out.println("[" + websiteName + "] Refreshing dashboard display");
    }
}
```

ObserverTest.java

java

```java
public class ObserverTest {
    public static void main(String[] args) {
        System.out.println("=== Observer Pattern Demo ===\n");

        StockMarket stockMarket = new StockMarket();

        MobileApp tradingApp = new MobileApp("TradingPro");
        WebApp financeWebsite = new WebApp("FinanceTracker");
        MobileApp investorApp = new MobileApp("InvestSmart");

        stockMarket.registerObserver(tradingApp);
        stockMarket.registerObserver(financeWebsite);
        stockMarket.registerObserver(investorApp);

        System.out.println("\n" + "=".repeat(50) + "\n");

        stockMarket.setStockPrice("AAPL", 150.25);

        System.out.println("\n" + "=".repeat(50) + "\n");

        stockMarket.setStockPrice("GOOGL", 2750.80);

        System.out.println("\n" + "-".repeat(30) + "\n");

        stockMarket.removeObserver(investorApp);

        System.out.println("\n" + "=".repeat(50) + "\n");

        stockMarket.setStockPrice("TSLA", 890.45);
    }
}
```

Sample Output:

```
=== Observer Pattern Demo ===

Observer registered for stock updates
Observer registered for stock updates
Observer registered for stock updates

================================================

Notifying all observers about AAPL price change to $150.25
[TradingPro Mobile App] AAPL is now $150.25
[TradingPro] Sending push notification to user
[FinanceTracker Website] AAPL updated to $150.25
[FinanceTracker] Refreshing dashboard display
[InvestSmart Mobile App] AAPL is now $150.25
[InvestSmart] Sending push notification to user

================================================

Notifying all observers about GOOGL price change to $2750.8
[TradingPro Mobile App] GOOGL is now $2750.8
[TradingPro] Sending push notification to user
[FinanceTracker Website] GOOGL updated to $2750.8
[FinanceTracker] Refreshing dashboard display
[InvestSmart Mobile App] GOOGL is now $2750.8
[InvestSmart] Sending push notification to user

------------------------------

Observer removed from stock updates

================================================

Notifying all observers about TSLA price change to $890.45
[TradingPro Mobile App] TSLA is now $890.45
[TradingPro] Sending push notification to user
[FinanceTracker Website] TSLA updated to $890.45
[FinanceTracker] Refreshing dashboard display
```

Exercise 8: Implementing the Strategy Pattern

PaymentStrategy.java

java

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

CreditCardPayment.java

java

```java
public class CreditCardPayment implements PaymentStrategy {
    private String cardNumber;
    private String name;

    public CreditCardPayment(String cardNumber, String name) {
        this.cardNumber = cardNumber;
        this.name = name;
    }

    @Override
    public boolean pay(double amount) {
        System.out.println("Processing credit card payment...");
        System.out.println("Card Number: ****-****-****-" + cardNumber.substring(cardNumber.len
        System.out.println("Cardholder: " + name);
        System.out.println("Amount: $" + amount);
        System.out.println("Credit card payment successful!");
        return true;
    }
}
```

PayPalPayment.java

java

```java
public class PayPalPayment implements PaymentStrategy {
    private String email;

    public PayPalPayment(String email) {
        this.email = email;
    }

    @Override
    public boolean pay(double amount) {
        System.out.println("Processing PayPal payment...");
        System.out.println("PayPal Account: " + email);
        System.out.println("Amount: $" + amount);
        System.out.println("PayPal payment successful!");
        return true;
    }
}
```

PaymentContext.java

java

```java
public class PaymentContext {
    private PaymentStrategy paymentStrategy;

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

    public boolean executePayment(double amount) {
        if (paymentStrategy == null) {
            System.out.println("No payment method selected!");
            return false;
        }
        return paymentStrategy.pay(amount);
    }
}
```

StrategyTest.java

java

```java
public class StrategyTest {
    public static void main(String[] args) {
        System.out.println("=== Strategy Pattern Demo ===\n");

        PaymentContext paymentContext = new PaymentContext();

        System.out.println("Order Total: $299.99\n");

        System.out.println("Payment Method 1: Credit Card");
        paymentContext.setPaymentStrategy(new CreditCardPayment("1234567890123456", "John Doe")
        paymentContext.executePayment(299.99);

        System.out.println("\n" + "=".repeat(50) + "\n");

        System.out.println("Order Total: $149.50\n");

        System.out.println("Payment Method 2: PayPal");
        paymentContext.setPaymentStrategy(new PayPalPayment("john.doe@email.com"));
        paymentContext.executePayment(149.50);
    }
}
```

Sample Output:

```
=== Strategy Pattern Demo ===

Order Total: $299.99

Payment Method 1: Credit Card
Processing credit card payment...
Card Number: ****-****-****-3456
Cardholder: John Doe
Amount: $299.99
Credit card payment successful!

================================================

Order Total: $149.50

Payment Method 2: PayPal
Processing PayPal payment...
PayPal Account: john.doe@email.com
Amount: $149.5
PayPal payment successful!
```

Exercise 9: Implementing the Command Pattern

Command.java

java

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

Light.java

java

```java
public class Light {
    private String location;
    private boolean isOn;

    public Light(String location) {
        this.location = location;
        this.isOn = false;
    }

    public void turnOn() {
        isOn = true;
        System.out.println(location + " light is ON");
    }

    public void turnOff() {
        isOn = false;
        System.out.println(location + " light is OFF");
    }

    public boolean isOn() {
        return isOn;
    }
}
```

LightOnCommand.java

java

```java
public class LightOnCommand implements Command {
    private Light light;

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

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

LightOffCommand.java

java

```java
public class LightOffCommand implements Command {
    private Light light;

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

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

RemoteControl.java

java

```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 set!");
        }
    }
}
```

CommandTest.java

java

```java
public class CommandTest {
    public static void main(String[] args) {
        System.out.println("=== Command Pattern Demo ===\n");

        Light livingRoomLight = new Light("Living Room");
        Light kitchenLight = new Light("Kitchen");

        LightOnCommand livingRoomLightOn = new LightOnCommand(livingRoomLight);
        LightOffCommand livingRoomLightOff = new LightOffCommand(livingRoomLight);
        LightOnCommand kitchenLightOn = new LightOnCommand(kitchenLight);
        LightOffCommand kitchenLightOff = new LightOffCommand(kitchenLight);

        RemoteControl remote = new RemoteControl();

        System.out.println("Testing Living Room Light:");
        remote.setCommand(livingRoomLightOn);
        remote.pressButton();

        remote.setCommand(livingRoomLightOff);
        remote.pressButton();

        System.out.println("\n" + "-".repeat(30) + "\n");

        System.out.println("Testing Kitchen Light:");
        remote.setCommand(kitchenLightOn);
        remote.pressButton();

        remote.setCommand(kitchenLightOff);
        remote.pressButton();
    }
}
```

Sample Output:

```
=== Command Pattern Demo ===

Testing Living Room Light:
Living Room light is ON
Living Room light is OFF

------------------------------

Testing Kitchen Light:
```