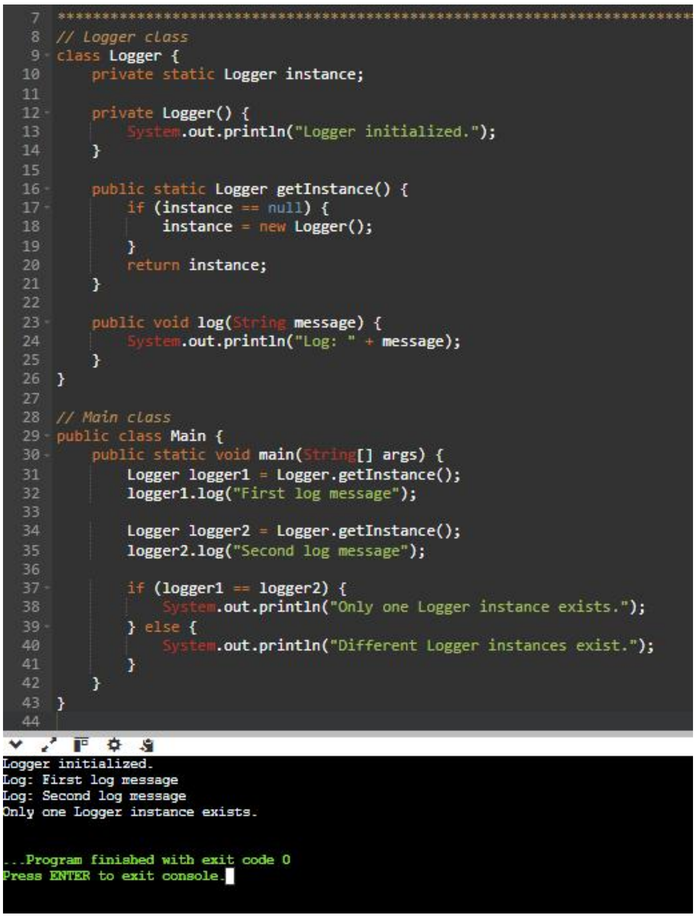
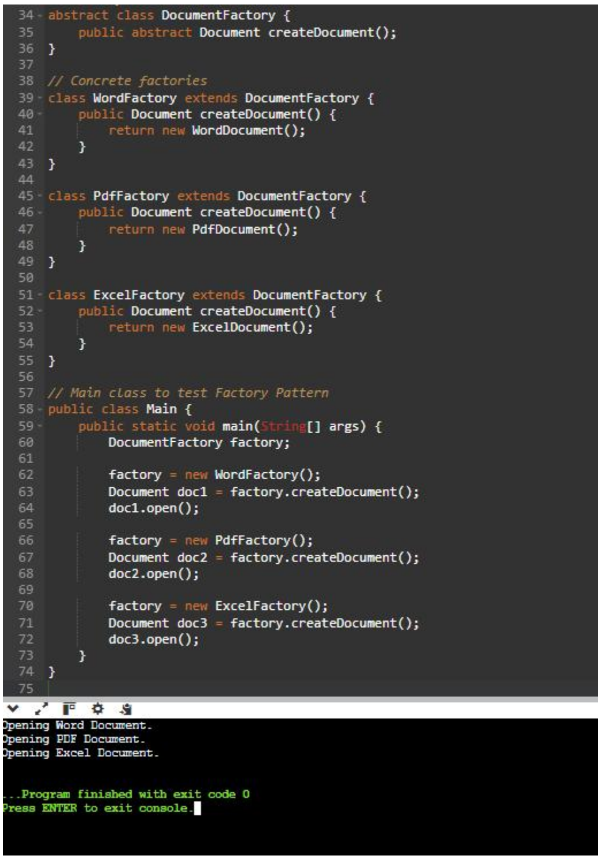
Exercise 1:



```
*******************************************************************************
// Logger class
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);
    }
}

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

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

        if (logger1 == logger2) {
            System.out.println("Only one Logger instance exists.");
        } else {
            System.out.println("Different Logger instances exist.");
        }
    }
}
```

```
Logger initialized.
Log: First log message
Log: Second log message
Only one Logger instance exists.


...Program finished with exit code 0
Press ENTER to exit console.
```

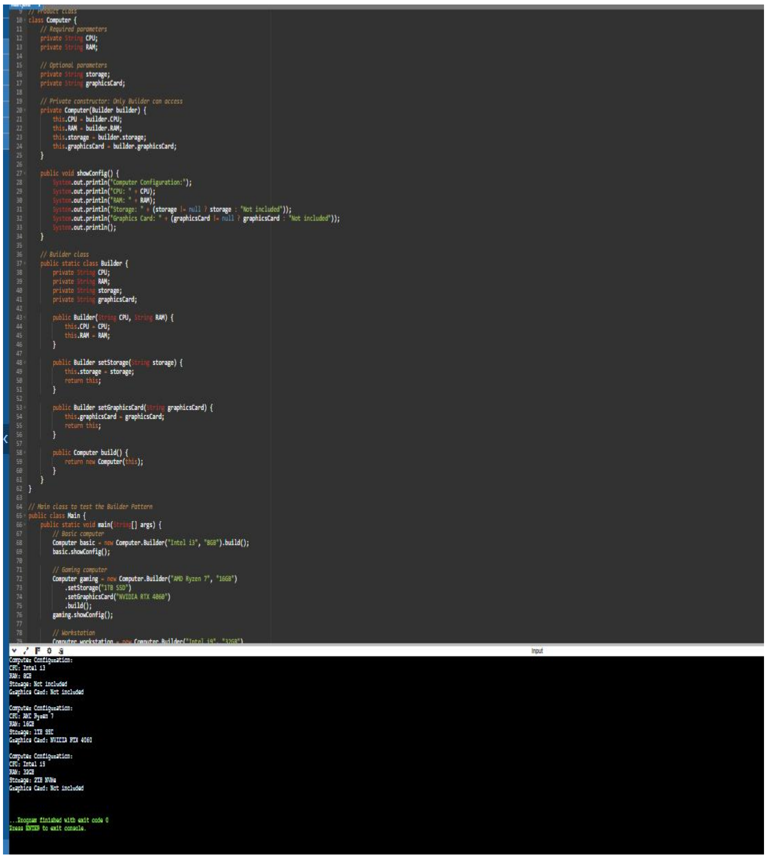
Exercise 2:



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

// Concrete factories
class WordFactory extends DocumentFactory {
    public Document createDocument() {
        return new WordDocument();
    }
}

class PdfFactory extends DocumentFactory {
    public Document createDocument() {
        return new PdfDocument();
    }
}

class ExcelFactory extends DocumentFactory {
    public Document createDocument() {
        return new ExcelDocument();
    }
}

// Main class to test Factory Pattern
public class Main {
    public static void main(String[] args) {
        DocumentFactory factory;

        factory = new WordFactory();
        Document doc1 = factory.createDocument();
        doc1.open();

        factory = new PdfFactory();
        Document doc2 = factory.createDocument();
        doc2.open();

        factory = new ExcelFactory();
        Document doc3 = factory.createDocument();
        doc3.open();
    }
}
```

```
Opening Word Document.
Opening PDF Document.
Opening Excel Document.


...Program finished with exit code 0
Press ENTER to exit console.
```

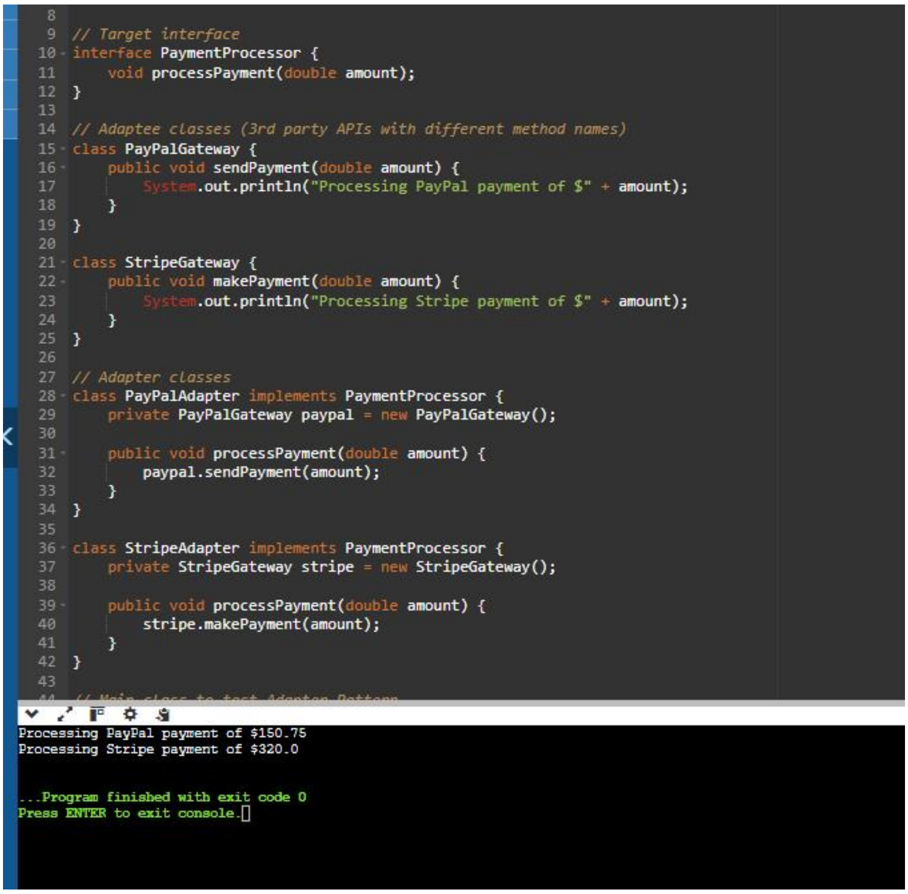
Exercise 3:



```java
// Product class
class Computer {
    // Required parameters
    private String CPU;
    private String RAM;

    // Optional parameters
    private String storage;
    private String graphicsCard;

    // Private constructor: Only Builder can access
    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 != null ? storage : "Not included"));
        System.out.println("Graphics Card: " + (graphicsCard != null ? graphicsCard : "Not included"));
        System.out.println();
    }

    // Builder class
    public static class Builder {
        private String CPU;
        private String RAM;
        private String storage;
        private String graphicsCard;

        public Builder(String CPU, String RAM) {
            this.CPU = CPU;
            this.RAM = RAM;
        }

        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);
        }
    }
}

// Main class to test the Builder Pattern
public class Main {
    public static void main(String[] args) {
        // Basic computer
        Computer basic = new Computer.Builder("Intel i3", "8GB").build();
        basic.showConfig();

        // Gaming computer
        Computer gaming = new Computer.Builder("AMD Ryzen 7", "16GB")
            .setStorage("1TB SSD")
            .setGraphicsCard("NVIDIA RTX 4060")
            .build();
        gaming.showConfig();

        // Workstation
        Computer workstation = new Computer.Builder("Intel i9", "32GB")
```

```
Computer Configuration:
CPU: Intel i3
RAM: 8GB
Storage: Not included
Graphics Card: Not included

Computer Configuration:
CPU: AMD Ryzen 7
RAM: 16GB
Storage: 1TB SSD
Graphics Card: NVIDIA RTX 4060

Computer Configuration:
CPU: Intel i9
RAM: 32GB
Storage: 2TB NVMe
Graphics Card: Not included



...Program finished with exit code 0
Press ENTER to exit console.
```

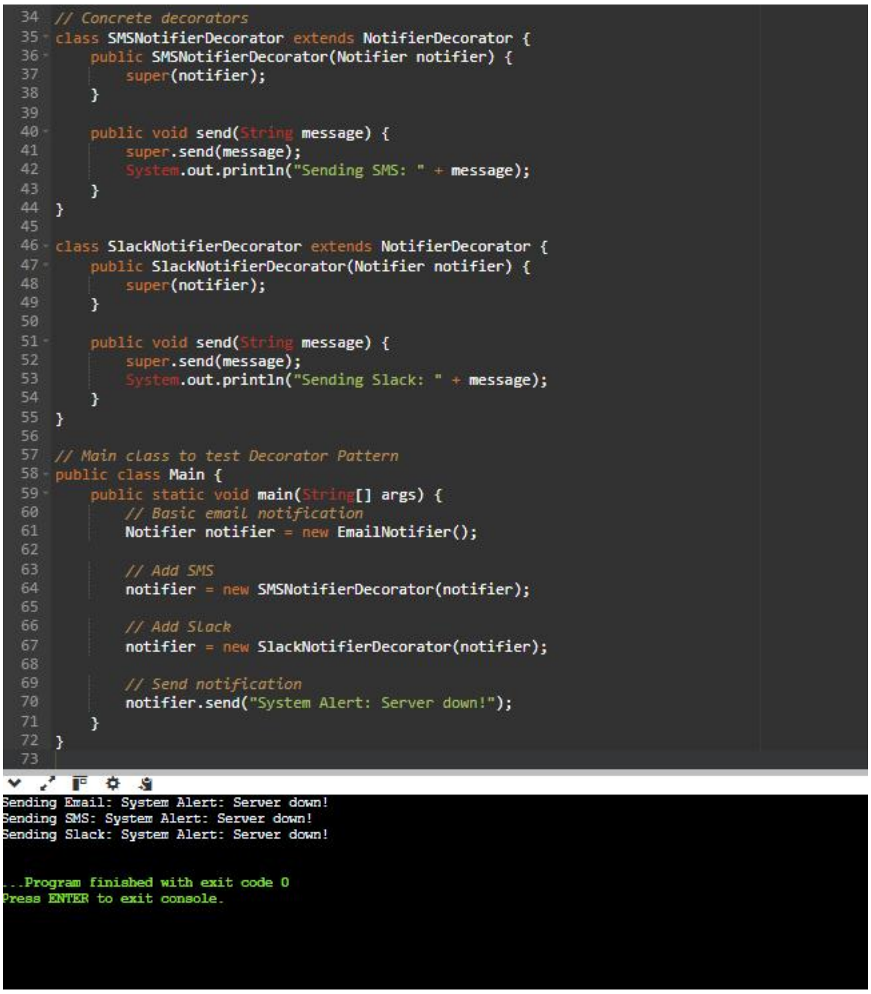
Exercise 4:



```java

// Target interface
interface PaymentProcessor {
    void processPayment(double amount);
}

// Adaptee classes (3rd party APIs with different method names)
class PayPalGateway {
    public void sendPayment(double amount) {
        System.out.println("Processing PayPal payment of $" + amount);
    }
}

class StripeGateway {
    public void makePayment(double amount) {
        System.out.println("Processing Stripe payment of $" + amount);
    }
}

// Adapter classes
class PayPalAdapter implements PaymentProcessor {
    private PayPalGateway paypal = new PayPalGateway();

    public void processPayment(double amount) {
        paypal.sendPayment(amount);
    }
}

class StripeAdapter implements PaymentProcessor {
    private StripeGateway stripe = new StripeGateway();

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

// Main class to test Adapter Pattern
```

```
Processing PayPal payment of $150.75
Processing Stripe payment of $320.0


...Program finished with exit code 0
Press ENTER to exit console.
```

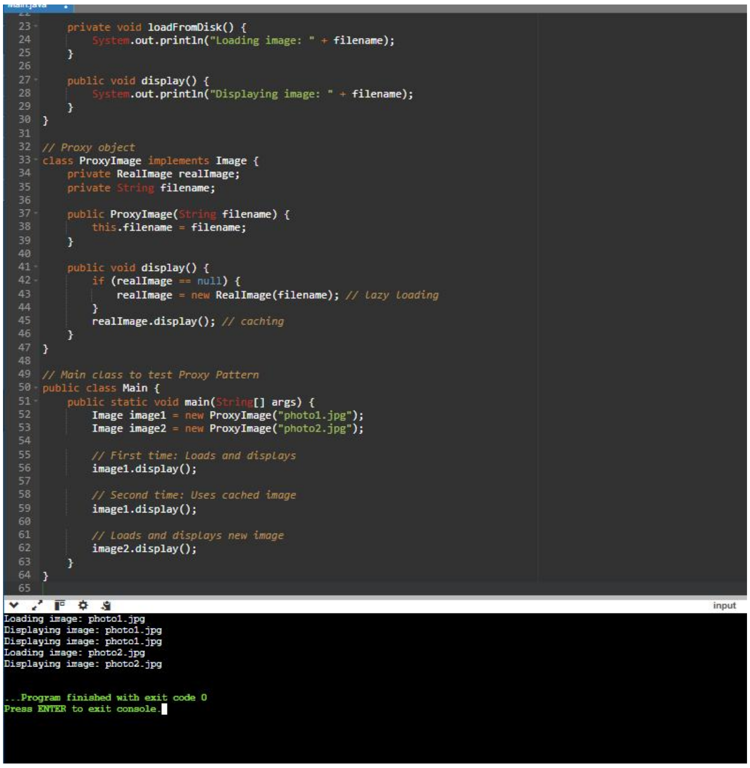
Exercise 5:



```java
// Concrete decorators
class SMSNotifierDecorator extends NotifierDecorator {
    public SMSNotifierDecorator(Notifier notifier) {
        super(notifier);
    }

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

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

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

// Main class to test Decorator Pattern
public class Main {
    public static void main(String[] args) {
        // Basic email notification
        Notifier notifier = new EmailNotifier();

        // Add SMS
        notifier = new SMSNotifierDecorator(notifier);

        // Add Slack
        notifier = new SlackNotifierDecorator(notifier);

        // Send notification
        notifier.send("System Alert: Server down!");
    }
}
```

```
Sending Email: System Alert: Server down!
Sending SMS: System Alert: Server down!
Sending Slack: System Alert: Server down!


...Program finished with exit code 0
Press ENTER to exit console.
```

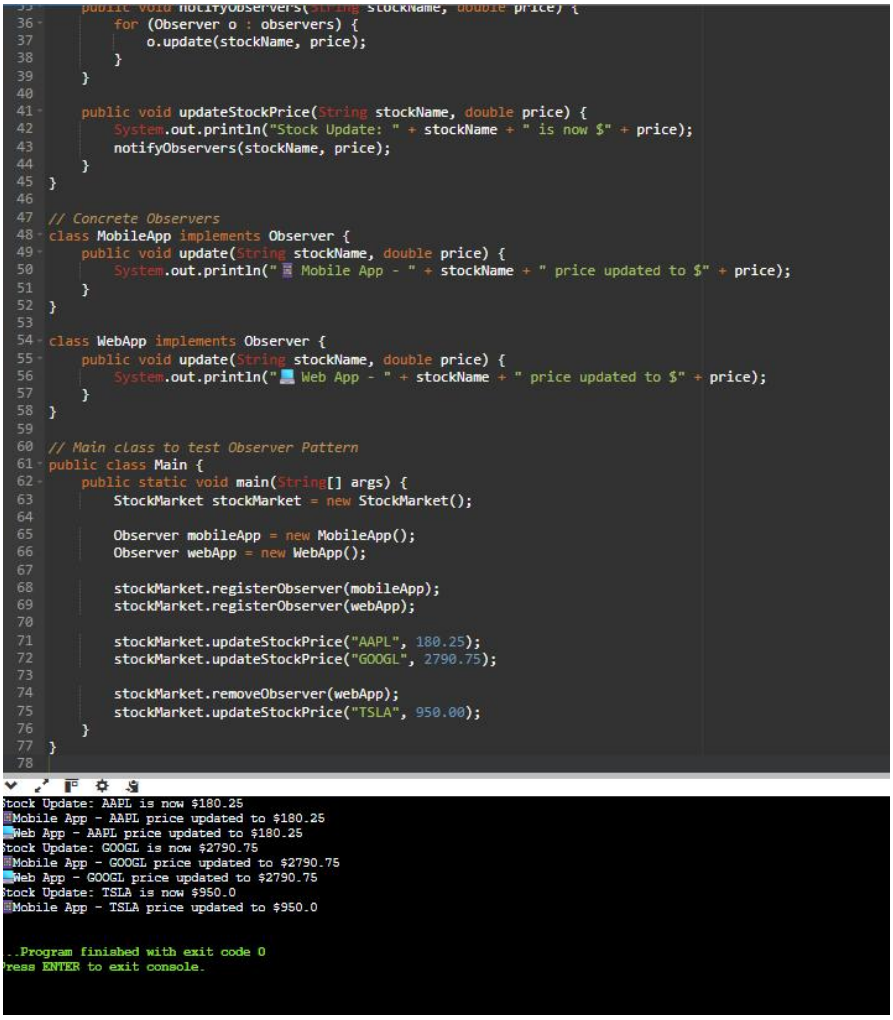
Exercise 6:



```java
    private void loadFromDisk() {
        System.out.println("Loading image: " + filename);
    }

    public void display() {
        System.out.println("Displaying image: " + filename);
    }
}

// Proxy object
class ProxyImage implements Image {
    private RealImage realImage;
    private String filename;

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

    public void display() {
        if (realImage == null) {
            realImage = new RealImage(filename); // lazy loading
        }
        realImage.display(); // caching
    }
}

// Main class to test Proxy Pattern
public class Main {
    public static void main(String[] args) {
        Image image1 = new ProxyImage("photo1.jpg");
        Image image2 = new ProxyImage("photo2.jpg");

        // First time: Loads and displays
        image1.display();

        // Second time: Uses cached image
        image1.display();

        // Loads and displays new image
        image2.display();
    }
}
```

```
Loading image: photo1.jpg
Displaying image: photo1.jpg
Displaying image: photo1.jpg
Loading image: photo2.jpg
Displaying image: photo2.jpg


...Program finished with exit code 0
Press ENTER to exit console.
```

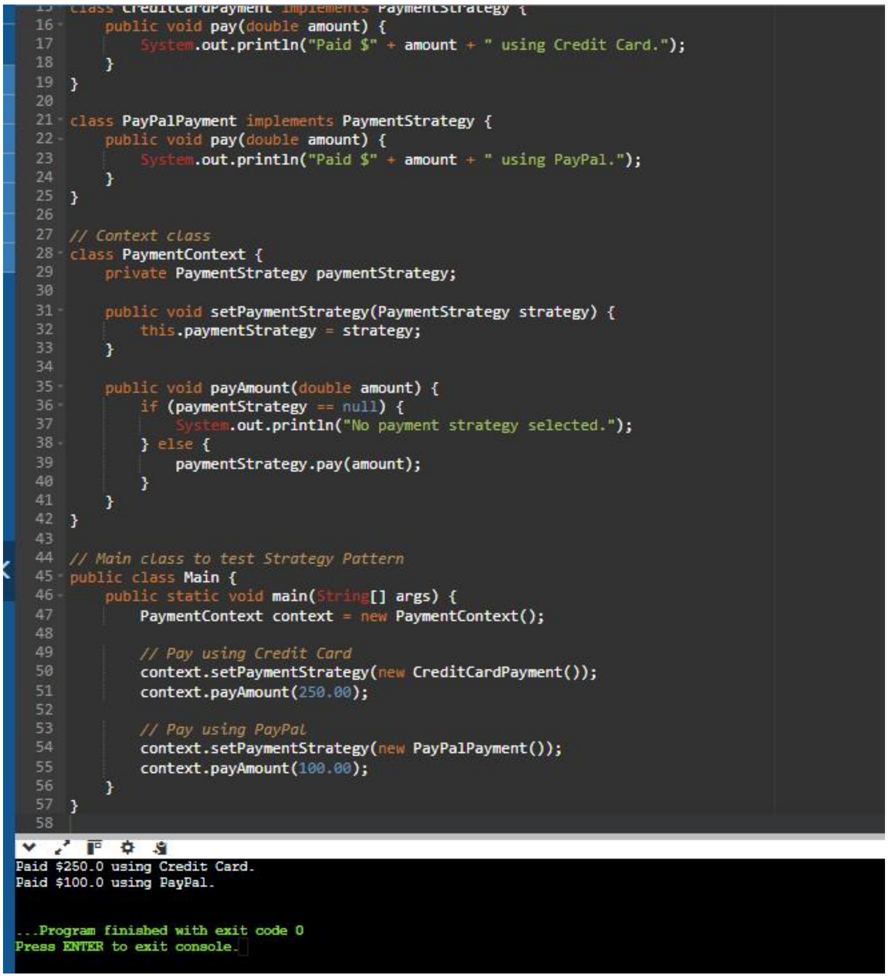
Exercise 7:



```java
    public void notifyObservers(String stockName, double price) {
        for (Observer o : observers) {
            o.update(stockName, price);
        }
    }

    public void updateStockPrice(String stockName, double price) {
        System.out.println("Stock Update: " + stockName + " is now $" + price);
        notifyObservers(stockName, price);
    }
}

// Concrete Observers
class MobileApp implements Observer {
    public void update(String stockName, double price) {
        System.out.println("📱 Mobile App - " + stockName + " price updated to $" + price);
    }
}

class WebApp implements Observer {
    public void update(String stockName, double price) {
        System.out.println("💻 Web App - " + stockName + " price updated to $" + price);
    }
}

// Main class to test Observer Pattern
public class Main {
    public static void main(String[] args) {
        StockMarket stockMarket = new StockMarket();

        Observer mobileApp = new MobileApp();
        Observer webApp = new WebApp();

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

        stockMarket.updateStockPrice("AAPL", 180.25);
        stockMarket.updateStockPrice("GOOGL", 2790.75);

        stockMarket.removeObserver(webApp);
        stockMarket.updateStockPrice("TSLA", 950.00);
    }
}
```

```
Stock Update: AAPL is now $180.25
📱Mobile App - AAPL price updated to $180.25
💻Web App - AAPL price updated to $180.25
Stock Update: GOOGL is now $2790.75
📱Mobile App - GOOGL price updated to $2790.75
💻Web App - GOOGL price updated to $2790.75
Stock Update: TSLA is now $950.0
📱Mobile App - TSLA price updated to $950.0


...Program finished with exit code 0
Press ENTER to exit console.
```

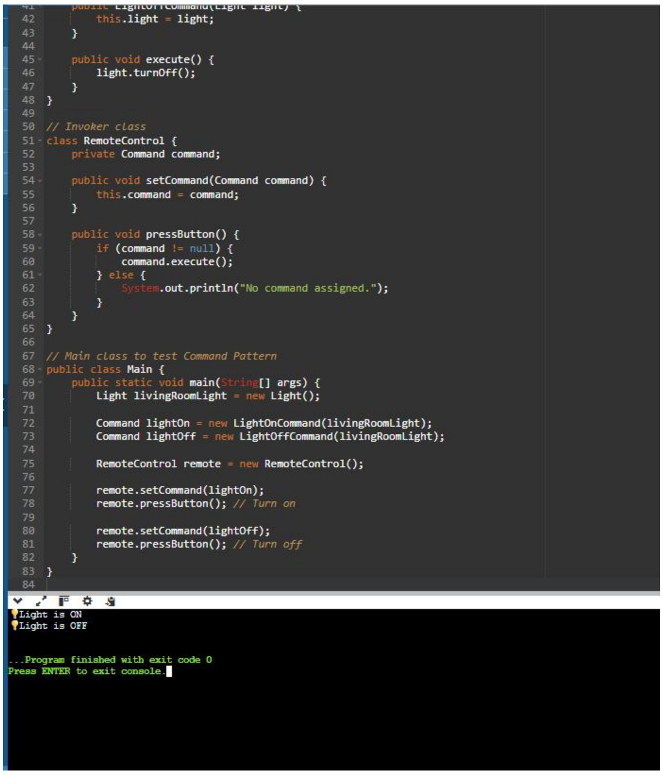
Exercise 8:



```
class CreditCardPayment implements PaymentStrategy {
    public void pay(double amount) {
        System.out.println("Paid $" + amount + " using Credit Card.");
    }
}

class PayPalPayment implements PaymentStrategy {
    public void pay(double amount) {
        System.out.println("Paid $" + amount + " using PayPal.");
    }
}

// Context class
class PaymentContext {
    private PaymentStrategy paymentStrategy;

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

    public void payAmount(double amount) {
        if (paymentStrategy == null) {
            System.out.println("No payment strategy selected.");
        } else {
            paymentStrategy.pay(amount);
        }
    }
}

// Main class to test Strategy Pattern
public class Main {
    public static void main(String[] args) {
        PaymentContext context = new PaymentContext();

        // Pay using Credit Card
        context.setPaymentStrategy(new CreditCardPayment());
        context.payAmount(250.00);

        // Pay using PayPal
        context.setPaymentStrategy(new PayPalPayment());
        context.payAmount(100.00);
    }
}
```

```
Paid $250.0 using Credit Card.
Paid $100.0 using PayPal.


...Program finished with exit code 0
Press ENTER to exit console.
```

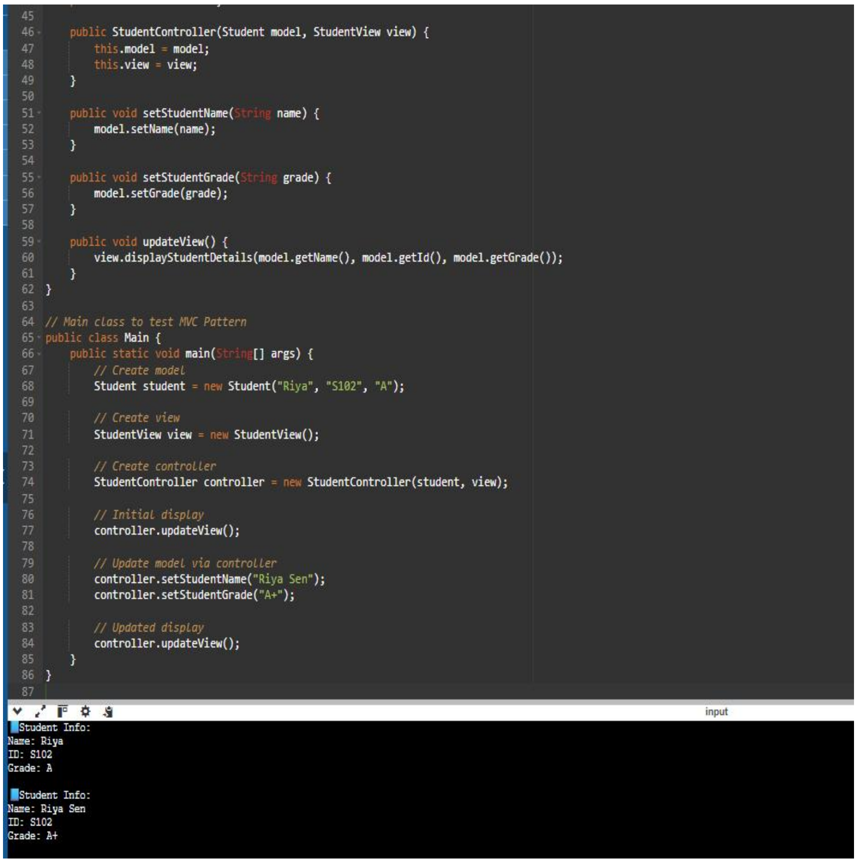
Exercise 9:



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

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

// Invoker class
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.");
        }
    }
}

// Main class to test Command Pattern
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(); // Turn on

        remote.setCommand(lightOff);
        remote.pressButton(); // Turn off
    }
}
```

```
💡Light is ON
💡Light is OFF


...Program finished with exit code 0
Press ENTER to exit console.
```

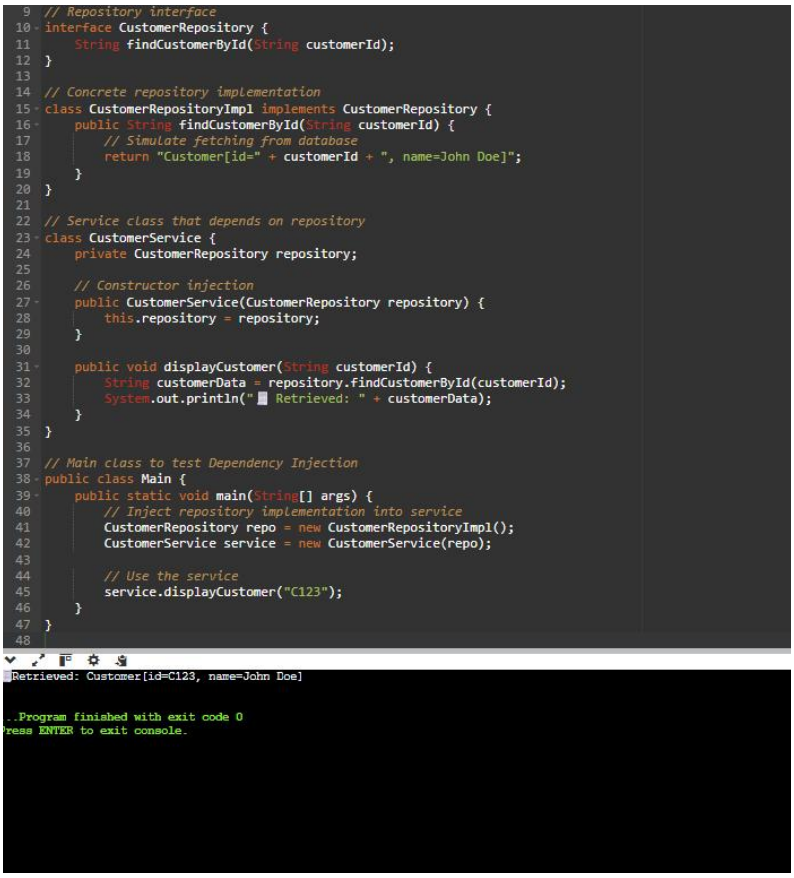
Exercise 10:



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

    public void setStudentName(String name) {
        model.setName(name);
    }

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

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

// Main class to test MVC Pattern
public class Main {
    public static void main(String[] args) {
        // Create model
        Student student = new Student("Riya", "S102", "A");

        // Create view
        StudentView view = new StudentView();

        // Create controller
        StudentController controller = new StudentController(student, view);

        // Initial display
        controller.updateView();

        // Update model via controller
        controller.setStudentName("Riya Sen");
        controller.setStudentGrade("A+");

        // Updated display
        controller.updateView();
    }
}
```

```
█Student Info:
Name: Riya
ID: S102
Grade: A

█Student Info:
Name: Riya Sen
ID: S102
Grade: A+
```

Exercise 11:



```
// Repository interface
interface CustomerRepository {
    String findCustomerById(String customerId);
}

// Concrete repository implementation
class CustomerRepositoryImpl implements CustomerRepository {
    public String findCustomerById(String customerId) {
        // Simulate fetching from database
        return "Customer[id=" + customerId + ", name=John Doe]";
    }
}

// Service class that depends on repository
class CustomerService {
    private CustomerRepository repository;

    // Constructor injection
    public CustomerService(CustomerRepository repository) {
        this.repository = repository;
    }

    public void displayCustomer(String customerId) {
        String customerData = repository.findCustomerById(customerId);
        System.out.println("█ Retrieved: " + customerData);
    }
}

// Main class to test Dependency Injection
public class Main {
    public static void main(String[] args) {
        // Inject repository implementation into service
        CustomerRepository repo = new CustomerRepositoryImpl();
        CustomerService service = new CustomerService(repo);

        // Use the service
        service.displayCustomer("C123");
    }
}
```

```
█Retrieved: Customer[id=C123, name=John Doe]


..Program finished with exit code 0
ress ENTER to exit console.
```