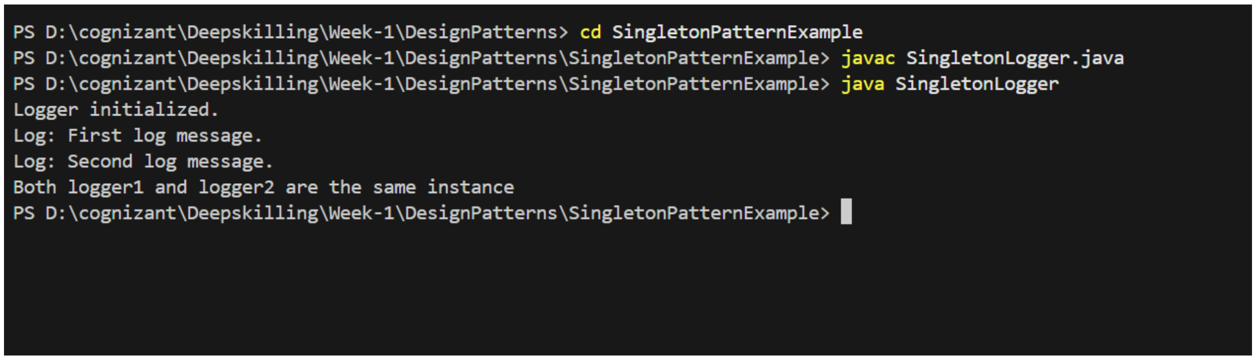
Exercise 1: Implementing the Singleton Pattern

CODE:

```
public class SingletonLogger {
    static 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("First log message.");
        Logger logger2 = Logger.getInstance();
        logger2.log("Second log message.");
        if (logger1 == logger2) {
            System.out.println("Both logger1 and logger2 are the same instance ");
        } else {
            System.out.println("Different instances");
        }
    }
}
```

OUTPUT :



```
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns> cd SingletonPatternExample
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns\SingletonPatternExample> javac SingletonLogger.java
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns\SingletonPatternExample> java SingletonLogger
Logger initialized.
Log: First log message.
Log: Second log message.
Both logger1 and logger2 are the same instance
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns\SingletonPatternExample>
```

Exercise 2: Implementing the Factory Method Pattern

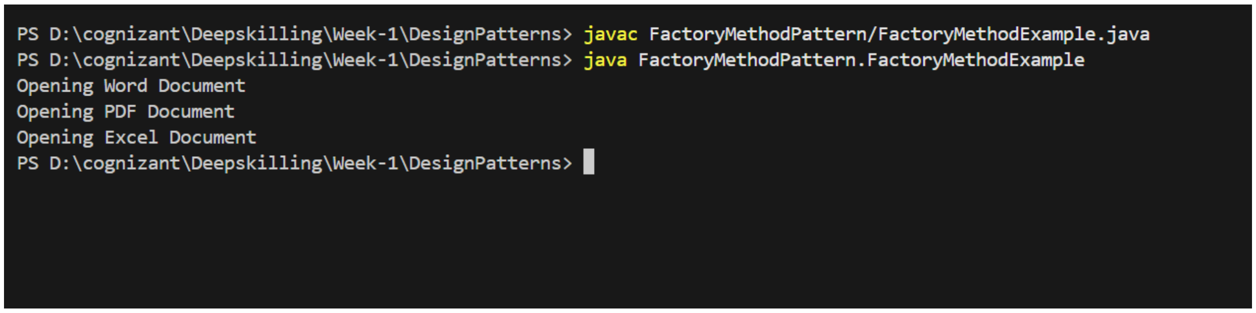
CODE :

```
package FactoryMethodPattern;
public class FactoryMethodExample {
    interface Document {
        void open();
    }
    static class WordDocument implements Document {
        public void open() {
            System.out.println("Opening Word Document");
        }
    }
    static class PdfDocument implements Document {
        public void open() {
            System.out.println("Opening PDF Document");
        }
    }
```

```
static class ExcelDocument implements Document {

    public void open() {

        System.out.println("Opening Excel Document");

    }

}

static abstract class DocumentFactory {

    public abstract Document createDocument();

}

static class WordDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new WordDocument();

    }

}

static class PdfDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new PdfDocument();

    }

}

static class ExcelDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new ExcelDocument();

    }

}

public static void main(String[] args) {

    DocumentFactory wordFactory = new WordDocumentFactory();

    Document word = wordFactory.createDocument();

    word.open();

    DocumentFactory pdfFactory = new PdfDocumentFactory();
```

```
        Document pdf = pdfFactory.createDocument();

        pdf.open();

        DocumentFactory excelFactory = new ExcelDocumentFactory();

        Document excel = excelFactory.createDocument();

        excel.open();

    }

}
```

OUTPUT :



```
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns> javac FactoryMethodPattern/FactoryMethodExample.java
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns> java FactoryMethodPattern.FactoryMethodExample
Opening Word Document
Opening PDF Document
Opening Excel Document
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns>
```

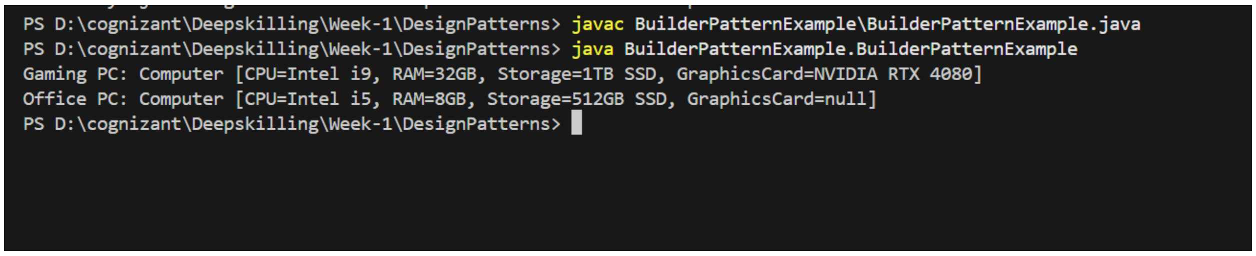
Exercise 3: Implementing the Builder Pattern

CODE :

```
package BuilderPatternExample;

public class BuilderPatternExample {

    static class Computer {

        private String CPU;

        private String RAM;

        private String storage;

        private String graphicsCard;

        private Computer(Builder builder) {

            this.CPU = builder.CPU;

            this.RAM = builder.RAM;
        this.storage = builder.storage;
        this.graphicsCard = builder.graphicsCard;
    }
public 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);
        } }
```

```
  @Override
      public String toString() {
          return "Computer [CPU=" + CPU + ", RAM=" + RAM + ", Storage=" + storage + ", GraphicsCard=" +
graphicsCard + "]";
      }
  }
  public static void main(String[] args) {
      Computer gamingPC = new Computer.Builder()
              .setCPU("Intel i9")
              .setRAM("32GB")
              .setStorage("1TB SSD")
              .setGraphicsCard("NVIDIA RTX 4080")
              .build();
      Computer officePC = new Computer.Builder()
              .setCPU("Intel i5")
              .setRAM("8GB")
              .setStorage("512GB SSD")
              .build();
      System.out.println("Gaming PC: " + gamingPC);
      System.out.println("Office PC: " + officePC);
  }
}
```

OUTPUT :



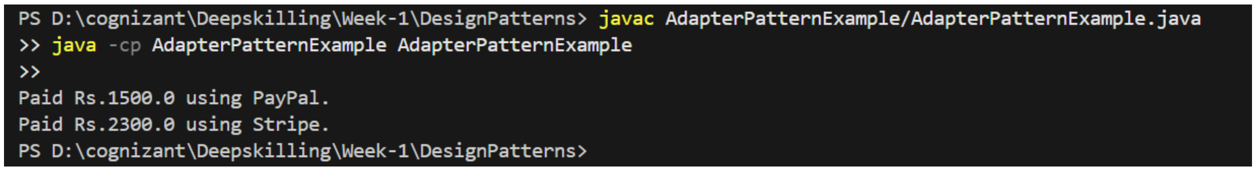
```
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns> javac BuilderPatternExample\BuilderPatternExample.java
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns> java BuilderPatternExample.BuilderPatternExample
Gaming PC: Computer [CPU=Intel i9, RAM=32GB, Storage=1TB SSD, GraphicsCard=NVIDIA RTX 4080]
Office PC: Computer [CPU=Intel i5, RAM=8GB, Storage=512GB SSD, GraphicsCard=null]
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns>
```

Exercise 4: Implementing the Adapter Pattern

CODE :

```
public class AdapterPatternExample {

  interface PaymentProcessor {

    void processPayment(double amount);

  }

  static class PayPal {

    public void sendPayment(double amount) {

      System.out.println("Paid Rs." + amount + " using PayPal.");

    }

  }

  static class Stripe {

    public void makePayment(double amount) {

      System.out.println("Paid Rs." + amount + " using Stripe.");

    }

  }

  static class PayPalAdapter implements PaymentProcessor {

    private PayPal payPal;

    public PayPalAdapter(PayPal payPal) {

      this.payPal = payPal;

    }

    public void processPayment(double amount) {

      payPal.sendPayment(amount);

    }

  }
  static class StripeAdapter implements PaymentProcessor {
    private Stripe stripe;
    public StripeAdapter(Stripe stripe) {
      this.stripe = stripe;
    }
    public void processPayment(double amount) {
      stripe.makePayment(amount);
    }
  }
  public static void main(String[] args) {
    PaymentProcessor paypalProcessor = new PayPalAdapter(new PayPal());
    paypalProcessor.processPayment(1500);
    PaymentProcessor stripeProcessor = new StripeAdapter(new Stripe());
    stripeProcessor.processPayment(2300);
  }
}
```

OUTPUT :



```
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns> javac AdapterPatternExample/AdapterPatternExample.java
>> java -cp AdapterPatternExample AdapterPatternExample
>>
Paid Rs.1500.0 using PayPal.
Paid Rs.2300.0 using Stripe.
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns>
```

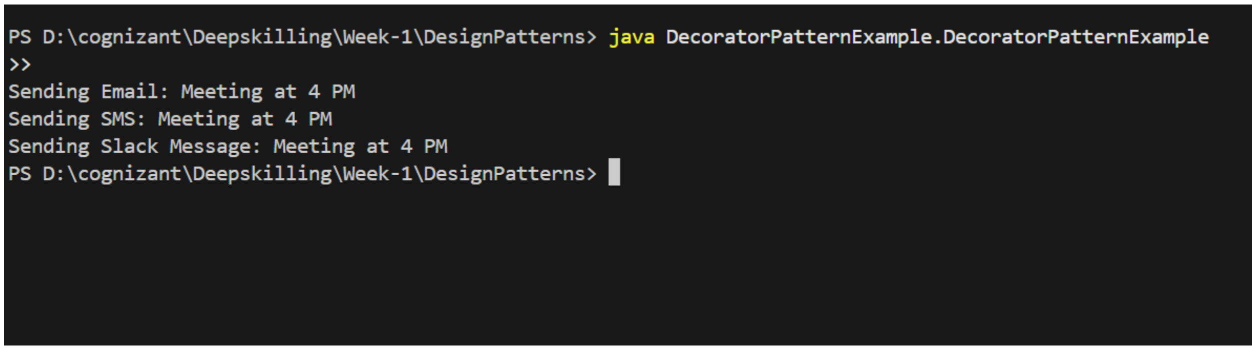
Exercise 5: Implementing the Decorator Pattern

CODE:

```
package DecoratorPatternExample;

public class DecoratorPatternExample {

  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 baseNotifier = new EmailNotifier();

        Notifier multiChannelNotifier = new SlackNotifierDecorator(new SMSNotifierDecorator(baseNotifier));

        multiChannelNotifier.send("Meeting at 4 PM");

    }

}
```

OUTPUT :



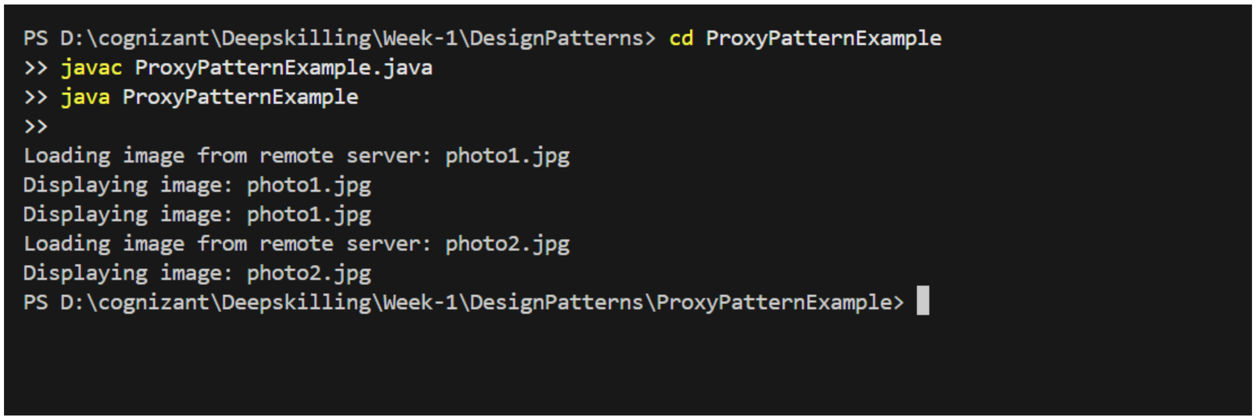
```
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns> java DecoratorPatternExample.DecoratorPatternExample
>>
Sending Email: Meeting at 4 PM
Sending SMS: Meeting at 4 PM
Sending Slack Message: Meeting at 4 PM
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns>
```

Exercise 6: Implementing the Proxy Pattern

CODE :

```
package ProxyPatternExample;

public class ProxyPatternExample {

  interface Image {

    void display();

  }

  static class RealImage implements Image {

    private String fileName;

    public RealImage(String fileName) {

      this.fileName = fileName;

      loadFromRemoteServer();

    }

    private void loadFromRemoteServer() {

      System.out.println("Loading image from remote server: " + fileName);

    }

    public void display() {

      System.out.println("Displaying image: " + fileName);

    }

  }

  static 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);
          }
          realImage.display();
        }
      }
    public static void main(String[] args) {
      Image image1 = new ProxyImage("photo1.jpg");
      Image image2 = new ProxyImage("photo2.jpg");
      image1.display();
      image1.display();
      image2.display();
    }
}
```

OUTPUT :



```
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns> cd ProxyPatternExample
>> javac ProxyPatternExample.java
>> java ProxyPatternExample
>>
Loading image from remote server: photo1.jpg
Displaying image: photo1.jpg
Displaying image: photo1.jpg
Loading image from remote server: photo2.jpg
Displaying image: photo2.jpg
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns\ProxyPatternExample>
```

Exercise 7: Implementing the Observer Pattern

CODE:

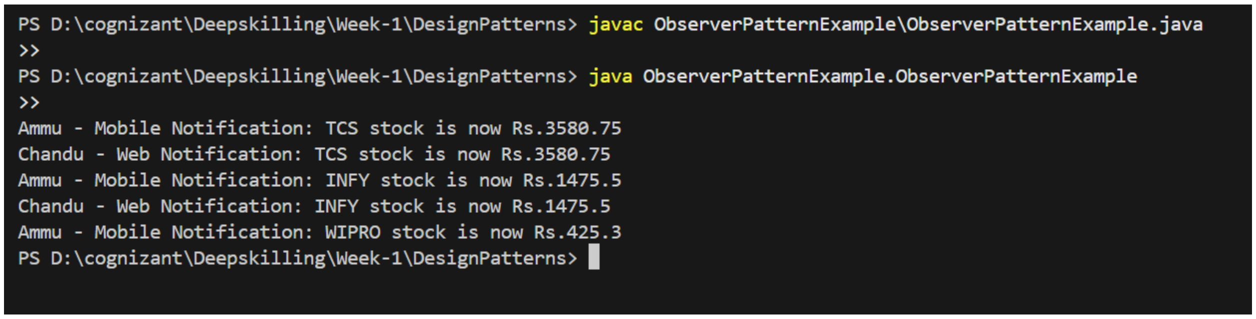
```
package ObserverPatternExample;
import java.util.*;
public class ObserverPatternExample {
```

```
interface Observer {
   void update(String stockName, double price);    }
interface Stock {
   void registerObserver(Observer observer);
   void removeObserver(Observer observer);
   void notifyObservers();
}
static class StockMarket implements Stock {
   private List<Observer> observers = new ArrayList<>();
   private String stockName;
   private double stockPrice;
   public void setStockPrice(String stockName, double price) {
      this.stockName = stockName;
      this.stockPrice = price;
      notifyObservers();
   }
   public void registerObserver(Observer observer) {
      observers.add(observer);
   }
   public void removeObserver(Observer observer) {
      observers.remove(observer);
   }
   public void notifyObservers() {
      for (Observer observer : observers) {
         observer.update(stockName, stockPrice);
      }
   }
}
```

```
static class MobileApp implements Observer {
    private String user;
    public MobileApp(String user) {
        this.user = user;
    }
    public void update(String stockName, double price) {
        System.out.println(user + " - Mobile Notification: " + stockName + " stock is now Rs." + price);
    }
}
static class WebApp implements Observer {
    private String user;
    public WebApp(String user) {
        this.user = user;
    }
    public void update(String stockName, double price) {
        System.out.println(user + " - Web Notification: " + stockName + " stock is now Rs." + price);
    }
}
public static void main(String[] args) {
    StockMarket market = new StockMarket();
    Observer mobileUser = new MobileApp("Ammu");
    Observer webUser = new WebApp("Chandu");
    market.registerObserver(mobileUser);
    market.registerObserver(webUser);

    market.setStockPrice("TCS", 3580.75);
    market.setStockPrice("INFY", 1475.50);
    market.removeObserver(webUser);
```

```
        market.setStockPrice("WIPRO", 425.30);
    }
}
```

OUTPUT :



```
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns> javac ObserverPatternExample\ObserverPatternExample.java
>>
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns> java ObserverPatternExample.ObserverPatternExample
>>
Ammu - Mobile Notification: TCS stock is now Rs.3580.75
Chandu - Web Notification: TCS stock is now Rs.3580.75
Ammu - Mobile Notification: INFY stock is now Rs.1475.5
Chandu - Web Notification: INFY stock is now Rs.1475.5
Ammu - Mobile Notification: WIPRO stock is now Rs.425.3
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns>
```

Exercise 8: Implementing the Strategy Pattern

CODE :

```
package StratergyPatternExample;

public class StratergyPatternExample {

    interface PaymentStrategy {

        void pay(double amount);

    }

    static class CreditCardPayment implements PaymentStrategy {

        private String cardNumber;

        public CreditCardPayment(String cardNumber) {

            this.cardNumber = cardNumber;

        }

        public void pay(double amount) {

            System.out.println("Paid Rs." + amount + " using Credit Card: " + cardNumber);

        }

    }
```

```
static class PayPalPayment implements PaymentStrategy {
    private String email;
    public PayPalPayment(String email) {
        this.email = email;
    }
    public void pay(double amount) {
        System.out.println("Paid Rs." + amount + " using PayPal account: " + email);
    }
}
static class PaymentContext {
    private PaymentStrategy paymentStrategy;
    public void setPaymentStrategy(PaymentStrategy paymentStrategy) {
        this.paymentStrategy = paymentStrategy;
    }
    public void payAmount(double amount) {
        if (paymentStrategy == null) {
            System.out.println("Please select a payment method first.");
        } else {
            paymentStrategy.pay(amount);
        }
    }
}
public static void main(String[] args) {
    PaymentContext context = new PaymentContext();
    context.setPaymentStrategy(new CreditCardPayment("1234-5678-9012-3456"));
    context.payAmount(2500);
    context.setPaymentStrategy(new PayPalPayment("ammu@example.com"));
    context.payAmount(3500);
    }
}
```

OUTPUT :

Exercise 9: Implementing the Command Pattern

CODE :

```
package CommandPatternExample;

public class CommandPatternExample {

 interface Command {

        void execute();

    }

    static class Light {

        public void turnOn() {

            System.out.println(" Light is ON");

        }

        public void turnOff() {

            System.out.println(" Light is OFF");

        }

    }

    static class LightOnCommand implements Command {

        private Light light;

        public LightOnCommand(Light light) {

            this.light = light;

        }

        public void execute() {

            light.turnOn();

        }

    }
```

```
static class LightOffCommand implements Command {
    private Light light;
    public LightOffCommand(Light light) {
        this.light = light;
    }
    public void execute() {
        light.turnOff();
    }
}
static 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.");
        }
    }
}
public static void main(String[] args) {
    Light light = new Light();
    Command lightOn = new LightOnCommand(light);
    Command lightOff = new LightOffCommand(light);
    RemoteControl remote = new RemoteControl();
    remote.setCommand(lightOn);
```

```
        remote.pressButton();

        remote.setCommand(lightOff);

        remote.pressButton();

    }

}
```

OUTPUT :

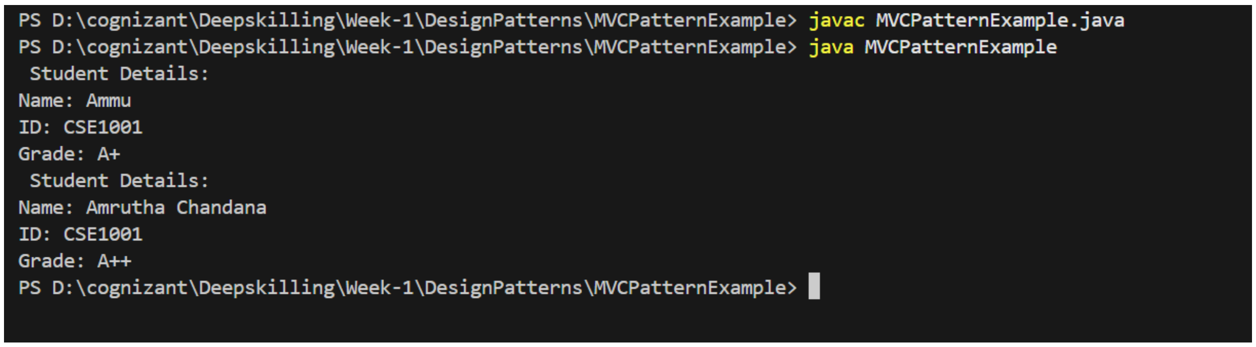
Exercise 10: Implementing the MVC Pattern

CODE :

```
package MVCPatternExample;

public class MVCPatternExample {

    static class Student {

        private String name;

        private String id;

        private String grade;


        public Student(String name, String id, String grade) {

            this.name = name;

            this.id = id;

            this.grade = 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; }

}

static 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);

    }

}

static 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 setStudentGrade(String grade) {

        model.setGrade(grade);

    }

    public void updateView() {

        view.displayStudentDetails(model.getName(), model.getId(), model.getGrade());

    }

}
```

```
  public static void main(String[] args) {
    Student student = new Student("Ammu", "CSE1001", "A+");
    StudentView view = new StudentView();
    StudentController controller = new StudentController(student, view);
    controller.updateView();
    controller.setStudentName("Amrutha Chandana");
    controller.setStudentGrade("A++");
    controller.updateView();
  }
}
```

OUTPUT :



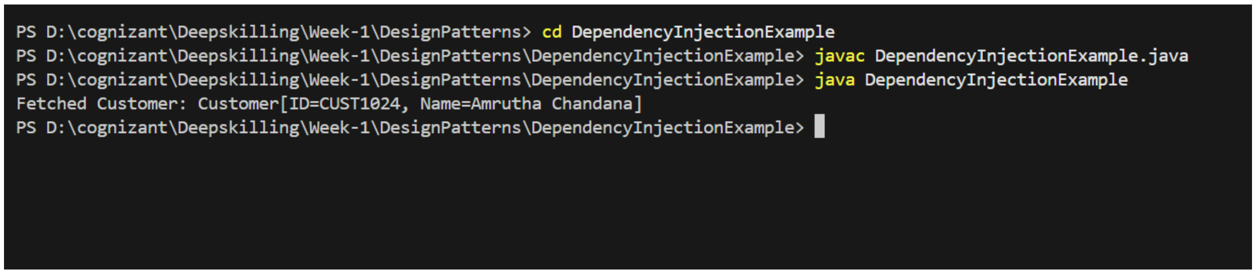
```
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns\MVCPatternExample> javac MVCPatternExample.java
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns\MVCPatternExample> java MVCPatternExample
 Student Details:
Name: Ammu
ID: CSE1001
Grade: A+
 Student Details:
Name: Amrutha Chandana
ID: CSE1001
Grade: A++
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns\MVCPatternExample>
```

Exercise 11: Implementing Dependency Injection

CODE :

```
package DependencyInjectionExample;
public class DependencyInjectionExample {
  interface CustomerRepository {
    String findCustomerById(String customerId);
  }
  static class CustomerRepositoryImpl implements CustomerRepository {
    public String findCustomerById(String customerId) {
            return "Customer[ID=" + customerId + ", Name=Amrutha Chandana]";
        }
    }
    static class CustomerService {
        private CustomerRepository repository;
        public CustomerService(CustomerRepository repository) {
            this.repository = repository;
        }
        public void showCustomer(String customerId) {
            String customerDetails = repository.findCustomerById(customerId);
            System.out.println("Fetched Customer: " + customerDetails);
        }
    }
    public static void main(String[] args) {
        CustomerRepository repository = new CustomerRepositoryImpl();
        CustomerService service = new CustomerService(repository);
        service.showCustomer("CUST1024");
  }
  }
```

OUTPUT :



```
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns> cd DependencyInjectionExample
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns\DependencyInjectionExample> javac DependencyInjectionExample.java
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns\DependencyInjectionExample> java DependencyInjectionExample
Fetched Customer: Customer[ID=CUST1024, Name=Amrutha Chandana]
PS D:\cognizant\Deepskilling\Week-1\DesignPatterns\DependencyInjectionExample>
```