**SOLUTIONS WEEK 1**

**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   * Create a class named Logger that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   * Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   * Create a test class to verify that only one instance of Logger is created and used across the application.

CODE

public class Logger {

      private static Logger instance;

    private Logger()

    {

        System.out.println("Account Created");

    }

    public static Logger getInstance()

    {

        if (instance == null)

            instance = new Logger();

        else

        System.out.println("Your account is already once created.plz don't create another");

        return instance;

    }

}

public class Test {

    public static void main(String[] args)

    {

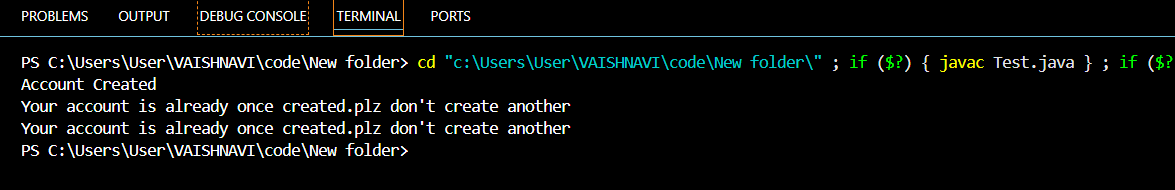
        Logger.getInstance();

        Logger.getInstance();

        Logger.getInstance();

    }

}

OUTPUT

**Exercise 2: Implementing the Factory Method Pattern**

**Scenario:**

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **FactoryMethodPatternExample**.
2. **Define Document Classes:**
   * Create interfaces or abstract classes for different document types such as **WordDocument**, **PdfDocument**, and **ExcelDocument**.
3. **Create Concrete Document Classes:**
   * Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.
4. **Implement the Factory Method:**
   * Create an abstract class **DocumentFactory** with a method **createDocument()**.
   * Create concrete factory classes for each document type that extends DocumentFactory and implements the **createDocument()** method.
5. **Test the Factory Method Implementation:**
   * Create a test class to demonstrate the creation of different document types using the factory method.

CODE

public abstract class Document {

    public abstract void display();

}

// Concrete Products

class WordDocument extends Document {

    @Override

    public void display() {

        System.out.println("This is Concrete WordDocument");

    }

}

class PDFDocument extends Document {

    @Override

    public void display() {

        System.out.println("This is Concrete PDFDocument.");

    }

}

class ExcelDocument extends Document {

    @Override

    public void display() {

        System.out.println("This is Concrete ExcelDocument.");

    }

}

public class DocumentFactory {

     public Document createDocument(String type) {

        if (type.equalsIgnoreCase("word")) {

            return new WordDocument();

        } else if (type.equalsIgnoreCase("pdf")) {

            return new PDFDocument();

        }

        else{

            return new ExcelDocument();

        }

    }

    public static void main(String[] args){

        DocumentFactory factory = new DocumentFactory();

        Document word = factory.createDocument("word");

        word.display(); // Outputs: Driving a car...

        Document pdf = factory.createDocument("pdf");

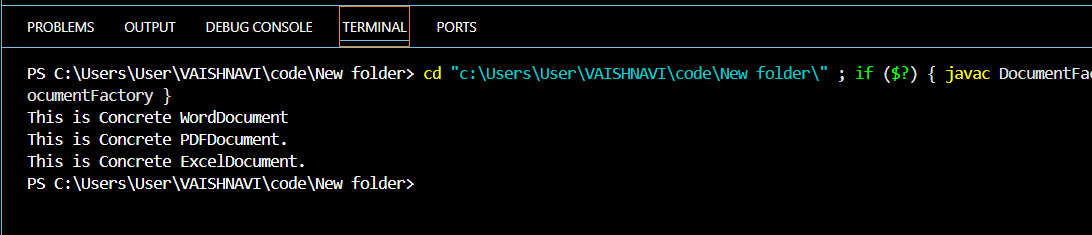
        pdf.display();

        Document excel = factory.createDocument("excel");

        excel.display();

            }

}

OUTPUT