WEEK-1

[ENGINEERING CONCEPTS]

DESIGN PATTERNS AND PRINCIPALS

**Exercise 1: Implementing the Singleton Pattern**

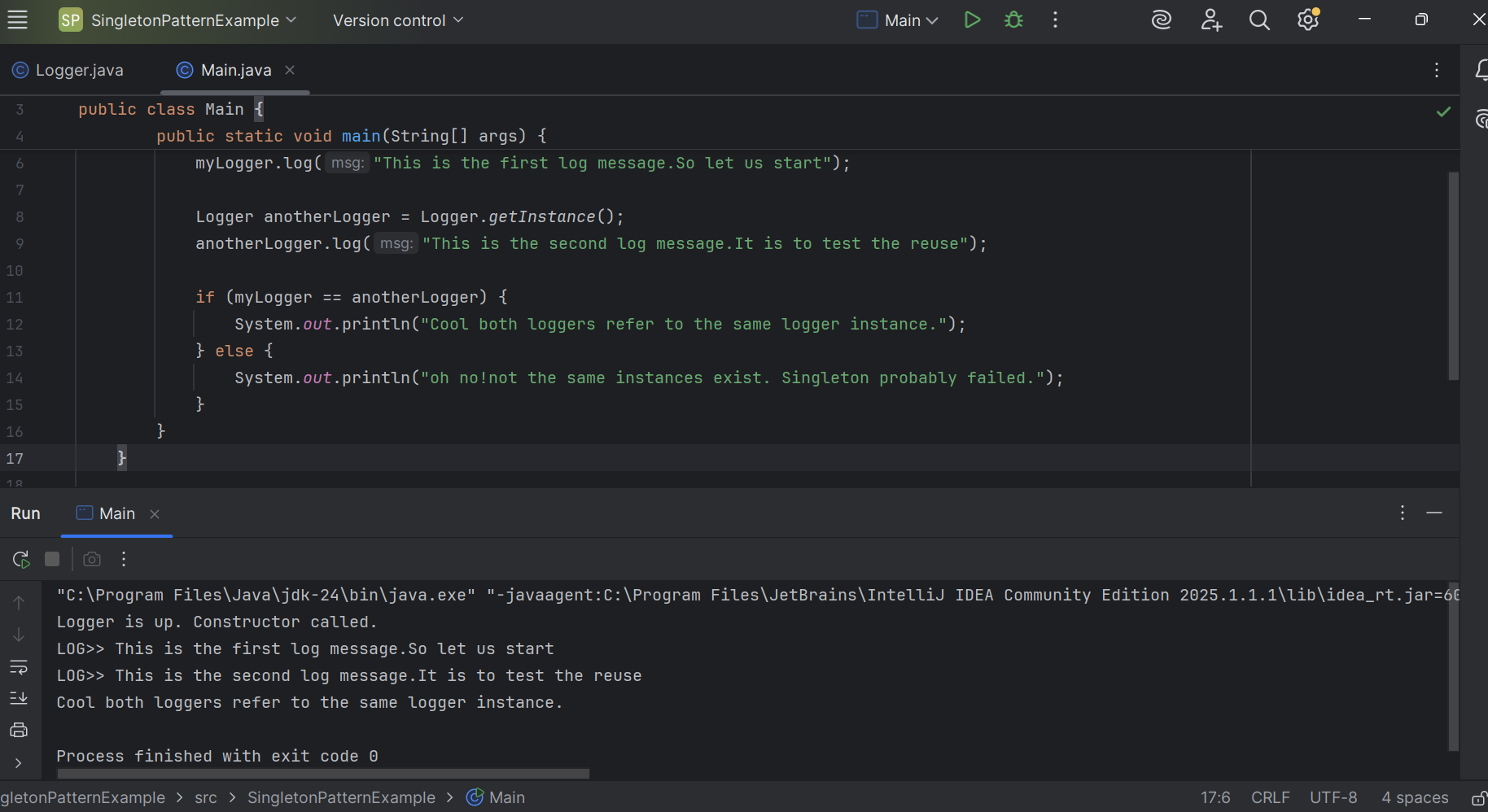
**->Logger.java class**

package SingletonPatternExample;  
  
// Logger class using a Singleton setup  
public class Logger {  
  
 // Keep track of the one-and-only instance (or so we hope)  
 private static Logger *singleLogger*;  
  
 // Constructor stays private — no direct access allowed  
 private Logger() {  
 System.*out*.println("Logger is up. Constructor called.");  
 }  
  
 // Here's how to get the one instance. Simple lazy init.  
 public static Logger getInstance() {  
 if (*singleLogger* == null) {  
 // First time? Make it.  
 *singleLogger* = new Logger();  
 }  
 return *singleLogger*;  
 }  
  
 // Logging utility  
 public void log(String msg) {  
 System.*out*.println("LOG>> " + msg); // log prefix just for flair  
 }  
  
}

->**Main.java class**

package SingletonPatternExample;  
  
public class Main {  
 public static void main(String[] args) {  
 Logger myLogger = Logger.*getInstance*();  
 myLogger.log("This is the first log message.So let us start");  
  
 Logger anotherLogger = Logger.*getInstance*();  
 anotherLogger.log("This is the second log message.It is to test the reuse");  
  
 if (myLogger == anotherLogger) {  
 System.*out*.println("Cool both loggers refer to the same logger instance.");  
 } else {  
 System.*out*.println("oh no!not the same instances exist. Singleton probably failed.");  
 }  
 }  
 }

**OUTPUT SCREEN:**



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

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

**->Document.java**

package factorymethod;  
  
public interface Document {  
 void open(); // No loops here. It's like a method signature.  
}

**->DocumentFactory.java**

package factorymethod;  
  
public abstract class DocumentFactory {  
 public abstract Document createDocument(); // Abstract method, no logic here  
}

**->ExcelDoc.java**

package factorymethod;  
  
public class ExcelDoc implements Document {  
 @Override  
 public void open() {  
 System.*out*.println("Opening an Excel document.");  
 }  
}

**->ExcelDocumentFactory.java**

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

**->PdfDocument.java**

package factorymethod;  
  
public class PdfDocument implements Document {  
 @Override  
 public void open() {  
 System.*out*.println("Opening a PDF document.");  
 }  
}

**->PdfDocumentFactory.java**

package factorymethod;  
  
public class PdfDocumentFactory extends DocumentFactory {  
 @Override  
 public Document createDocument() {  
 return new PdfDocument(); // No loop here, just object creation  
 }  
}

**->WordDocument.java**

package factorymethod;  
  
public class WordDocument implements Document {  
 @Override  
 public void open() {  
 System.*out*.println("Opening a Word document."); // Inside method body, not a loop  
 }  
}

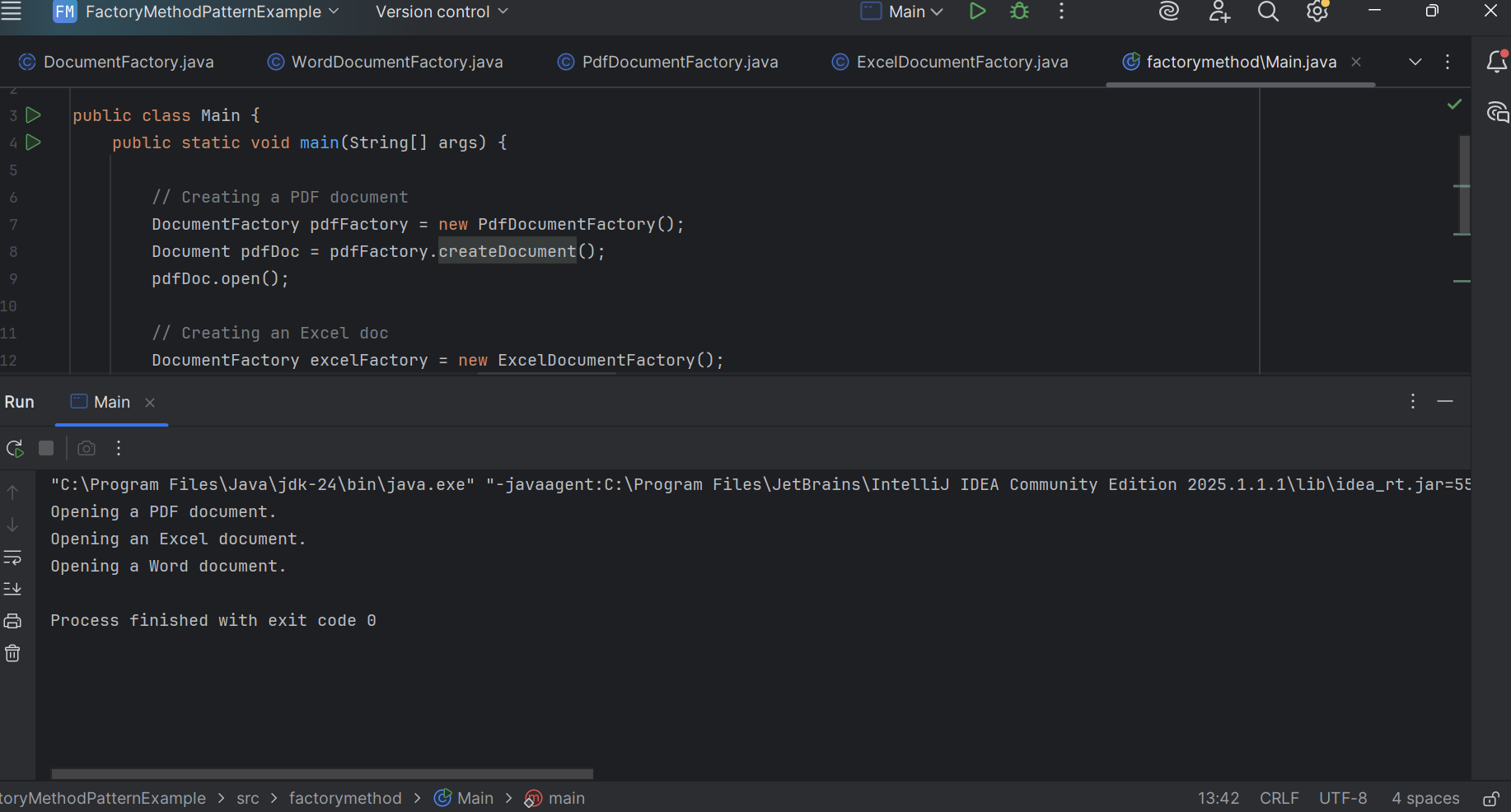
**->WordDocumentFactory.java**

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

**->Main.java**

public class Main {  
 public static void main(String[] args) {  
  
 // Creating a PDF document  
 DocumentFactory pdfFactory = new PdfDocumentFactory();  
 Document pdfDoc = pdfFactory.createDocument();  
 pdfDoc.open();  
  
 // Creating an Excel doc  
 DocumentFactory excelFactory = new ExcelDocumentFactory();  
 Document excelDoc = excelFactory.createDocument();  
 excelDoc.open();  
  
 //creating word doc  
 DocumentFactory wordFactory = new WordDocumentFactory();  
 Document wordDoc = wordFactory.createDocument();  
 wordDoc.open();  
 }  
}

**OUTPUT SCREEN:**

****

ALGORITHMS DATA STRUCTURES

**Exercise 2: E-commerce Platform Search Function**

**->Product.java class**

package search;  
  
public class Product {  
 private int productId;  
 private String productName;  
 private String category;  
  
 public Product(int productId, String productName, String category) {  
 this.productId = productId;  
 this.productName = productName;  
 this.category = category;  
 }  
  
 public int getProductId() {  
 return productId;  
 }  
  
 public String getProductName() {  
 return productName;  
 }  
  
 public String getCategory() {  
 return category;  
 }  
  
 @Override  
 public String toString() {  
 return productId + " - " + productName + " (" + category + ")";  
 }  
 }

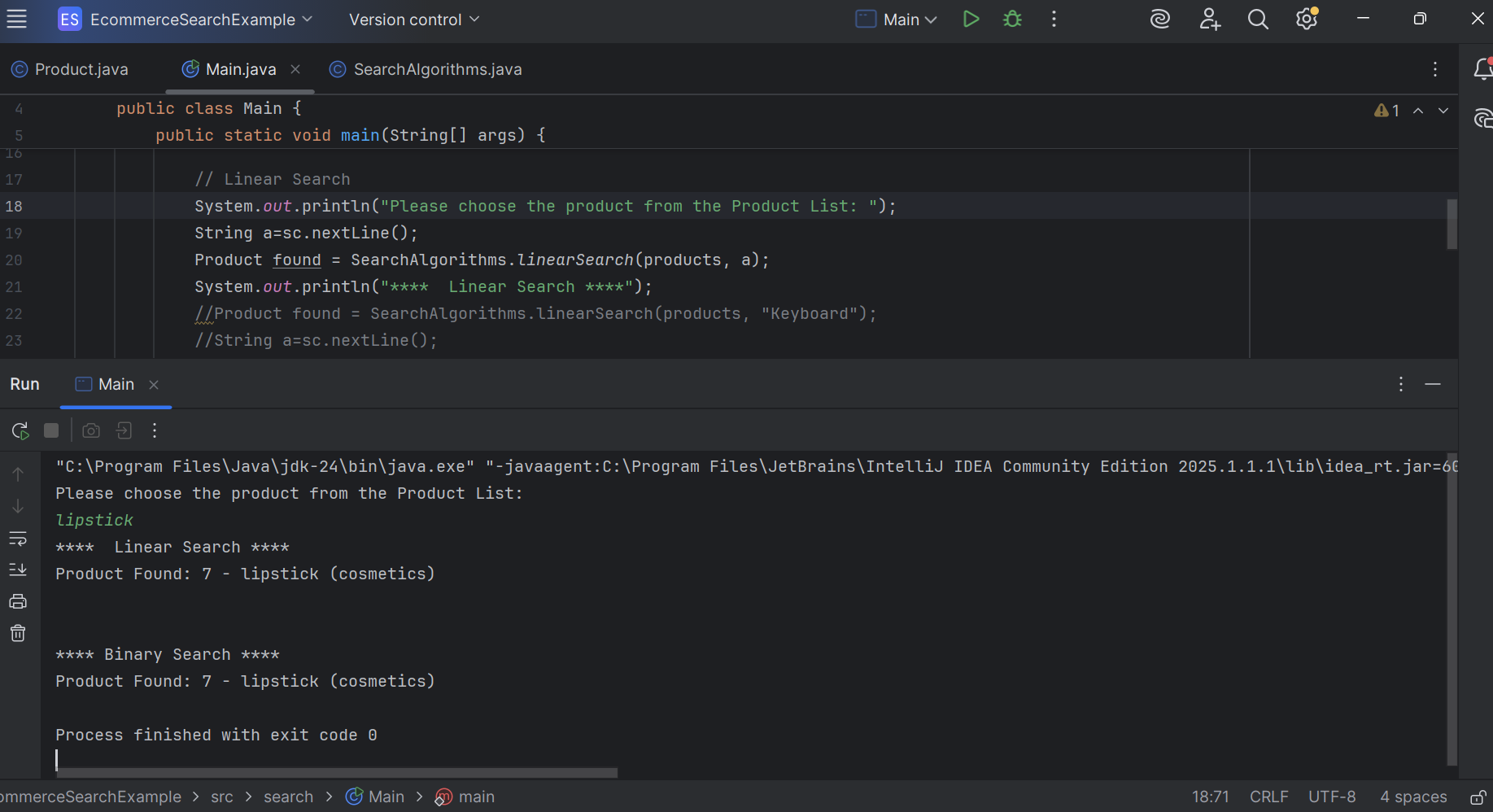
**->SearchAlgorithms.java class**

package search;  
  
import java.util.\*;  
  
 public class SearchAlgorithms {  
  
 // Linear Search  
 public static Product linearSearch(Product[] products, String name) {  
 for (Product product : products) {  
 if (product.getProductName().equalsIgnoreCase(name)) {  
 return product;  
 }  
 }  
 return null;  
 }  
  
 // Binary Search - assumes sorted by productName  
 public static Product binarySearch(Product[] products, String name) {  
 int left = 0;  
 int right = products.length - 1;  
  
 while (left <= right) {  
 int mid = (left + right) / 2;  
 int result = products[mid].getProductName().compareToIgnoreCase(name);  
  
 if (result == 0) {  
 return products[mid];  
 } else if (result < 0) {  
 left = mid + 1;  
 } else {  
 right = mid - 1;  
 }  
 }  
 return null;  
 }  
  
 // Sort helper  
 public static void sortByName(Product[] products) {  
 Arrays.*sort*(products, Comparator.*comparing*(Product::getProductName, String.*CASE\_INSENSITIVE\_ORDER*));  
 }  
 }

**->Main.java class**

package search;  
import java.util.\*;  
  
 public class Main {  
 public static void main(String[] args) {  
 Scanner sc=new Scanner(System.*in*);  
 Product[] products = {  
 new Product(1, "Laptop", "Electronics"),  
 new Product(2, "Shirt", "Clothing"),  
 new Product(3, "Comic Book", "Entertainment"),  
 new Product(4, "Keyboard", "Electronics"),  
 new Product(5, "Book", "Education"),  
 new Product(6, "Mouse", "Electronics"),  
 new Product(7, "lipstick", "cosmetics"),  
 };  
  
 // Linear Search  
 System.*out*.println("Please choose the product from the Prduct List: ");  
 String a=sc.nextLine();  
 Product found = SearchAlgorithms.*linearSearch*(products, a);  
 System.*out*.println("\*\*\*\* Linear Search \*\*\*\*");  
 //Product found = SearchAlgorithms.linearSearch(products, "Keyboard");  
 //String a=sc.nextLine();  
 //Product found = SearchAlgorithms.linearSearch(products, a);  
 System.*out*.println(found != null ? "Product Found: " + found : "Product not found");  
  
 // first we should sort the element to perform for Binary Search  
 SearchAlgorithms.*sortByName*(products);  
  
 // Binary Search  
 System.*out*.println("\n\n\*\*\*\* Binary Search \*\*\*\*");  
 found = SearchAlgorithms.*binarySearch*(products, a);  
 System.*out*.println(found != null ? "Product Found: " + found : "Product is not present sorry!");  
 }  
 }

**OUTPUT SCREEN:**

****

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

**Exercise 7: Financial Forecasting**

**->FinancialForecast.java class**

package forecast;  
  
public class FinancialForecast {  
  
 // Here I used recursive method to simplify the prediction of future value  
 public static double predictFutureValue(double presentValue, double growth, int yrs) {  
 if (yrs == 0) {  
 return presentValue;  
 }  
 return *predictFutureValue*(presentValue \* (1 + growth), growth, yrs - 1);  
 }  
 //the above method uses recursion to calculate the future values based on past growth rates.  
 // And it can be done in other ways.  
  
 // This is the more optimized version using iterative approach to avoid excessive computation as mentioned.  
 public static double FutureValueOptimized(double presentValue, double growth, int yrs) {  
 for (int i = 0; i < yrs; i++) {  
 presentValue \*= (1 + growth);  
 }  
 return presentValue;  
 }  
}

->**Main.java**

package forecast;  
import java.util.\*;  
public class Main {  
 public static void main(String[] args) {  
 Scanner sc=new Scanner(System.*in*);  
 System.*out*.println("please enter the initial amount: ");  
 double initialAmount = sc.nextDouble(); //this is how much we start with  
 System.*out*.println("please enter the yearly growth rate: ");  
 double yearlyGrowthRate = sc.nextDouble(); //guessing yearly growth  
 System.*out*.println("please enter the total years that we want to calculate amount: ");  
 int totalYears = sc.nextInt(); //after 5 year from now  
  
//the below statements prints  
 System.*out*.println("Recursive Forecast:");  
 double futureValue = FinancialForecast.*predictFutureValue*(initialAmount, yearlyGrowthRate, totalYears);  
 System.*out*.printf("Value after %d years: rupees %.2f\n\n", totalYears, futureValue);  
  
//The below statements prints the more optimized version to predict future values based on past growth rates.  
 System.*out*.println("Optimized Iterative Forecast version:");  
 double optimizedValue = FinancialForecast.*FutureValueOptimized*(initialAmount, yearlyGrowthRate, totalYears);  
 System.*out*.printf("Value after %d years: rupees %.2f\n", totalYears, optimizedValue);  
 }  
}

**OUTPUT SCREEN:**

