

WEEK-1

Design Principles And Patterns

(The implementation is done in VS CODE)

Exercise 1: Implementing the Singleton Pattern

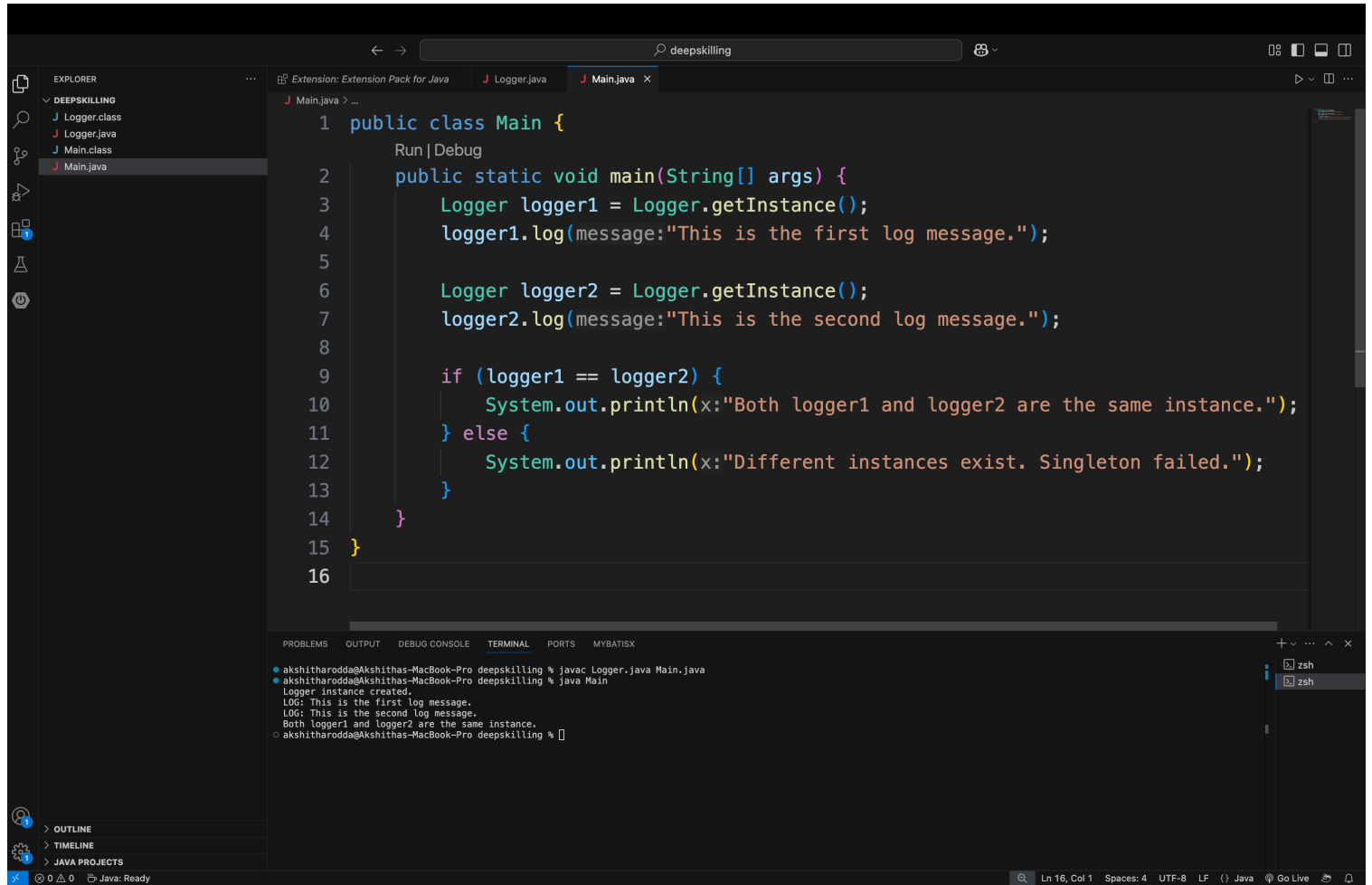
Logger.java

```
public class Logger {  
    private static Logger instance;  
  
    private Logger() {  
        System.out.println("Logger instance created.");  
    }  
  
    public static Logger getInstance() {  
        if (instance == null) {  
            instance = new Logger();  
        }  
        return instance;  
    }  
  
    public void log(String message) {  
        System.out.println("LOG: " + message);  
    }  
}
```

Main.java

```
public class Main {  
    public static void main(String[] args) {  
        Logger logger1 = Logger.getInstance();  
        logger1.log("This is the first log message.");  
  
        Logger logger2 = Logger.getInstance();  
        logger2.log("This is the second log message.");  
  
        if (logger1 == logger2) {  
            System.out.println("Both logger1 and logger2 are the same instance.");  
        } else {  
            System.out.println("Different instances exist. Singleton failed.");  
        }  
    }  
}
```

OUTPUT:



The screenshot shows an IDE with a dark theme. The Explorer panel on the left shows a project named 'DEEPSKILLING' with files 'Logger.class', 'Logger.java', 'Main.class', and 'Main.java'. The Main.java file is open in the editor, showing the following code:

```
1 public class Main {
2     public static void main(String[] args) {
3         Logger logger1 = Logger.getInstance();
4         logger1.log(message:"This is the first log message.");
5
6         Logger logger2 = Logger.getInstance();
7         logger2.log(message:"This is the second log message.");
8
9         if (logger1 == logger2) {
10             System.out.println(x:"Both logger1 and logger2 are the same instance.");
11         } else {
12             System.out.println(x:"Different instances exist. Singleton failed.");
13         }
14     }
15 }
16
```

The Output panel at the bottom shows the following output:

```
akshitharodda@Akshithas-MacBook-Pro deepskilling % javac Logger.java Main.java
akshitharodda@Akshithas-MacBook-Pro deepskilling % java Main
Logger instance created.
LOG: This is the first log message.
LOG: This is the second log message.
Both logger1 and logger2 are the same instance.
akshitharodda@Akshithas-MacBook-Pro deepskilling %
```

The output is:

Logger instance created.

LOG: This is the first log message.

LOG: This is the second log message.

Both logger1 and logger2 are the same instance.

Exercise 2: Implementing the Factory Method Pattern

Document.java

```
public interface Document {  
    void open();  
}
```

WordDocument.java

```
public class WordDocument implements Document {  
    public void open() {  
        System.out.println("Opening Word Document.");  
    }  
}
```

PdfDocument.java

```
public class PdfDocument implements Document {  
    public void open() {  
        System.out.println("Opening PDF Document.");  
    }  
}
```

ExcelDocument.java

```
public class ExcelDocument implements Document {  
    public void open() {  
        System.out.println("Opening Excel Document.");  
    }  
}
```

DocumentFactory.java

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

WordDocumentFactory.java

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

```
}  
}
```

PdfDocumentFactory.java

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

ExcelDocumentFactory.java

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

Main.java

```
public class Main {  
    public static void main(String[] args) {  
        DocumentFactory wordFactory = new WordDocumentFactory();  
        Document wordDoc = wordFactory.createDocument();  
        wordDoc.open();  
  
        DocumentFactory pdfFactory = new PdfDocumentFactory();  
        Document pdfDoc = pdfFactory.createDocument();  
        pdfDoc.open();  
  
        DocumentFactory excelFactory = new ExcelDocumentFactory();  
        Document excelDoc = excelFactory.createDocument();  
        excelDoc.open();  
    }  
}
```

OUTPUT:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         DocumentFactory wordFactory = new WordDocumentFactory();  
4         Document wordDoc = wordFactory.createDocument();  
5         wordDoc.open();  
6  
7         DocumentFactory pdfFactory = new PdfDocumentFactory();  
8         Document pdfDoc = pdfFactory.createDocument();  
9         pdfDoc.open();  
10  
11        DocumentFactory excelFactory = new ExcelDocumentFactory();  
12        Document excelDoc = excelFactory.createDocument();  
13        excelDoc.open();  
14    }  
15 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS MYBATISX

```
akshitharodda@Akshithas-MacBook-Pro factory % javac *.java  
akshitharodda@Akshithas-MacBook-Pro factory % java Main  
Opening Word Document.  
Opening PDF Document.  
Opening Excel Document.  
akshitharodda@Akshithas-MacBook-Pro factory %
```

Restart Visual Studio Code to apply the latest update.

Update Now Later Release Notes

The output is:
Opening Word Document.
Opening PDF Document.
Opening Excel Document.

Data structures and Algorithms

(The implementation is done in VS CODE)

Exercise 2: E-commerce Platform Search Function

Product.java

```
public class Product {
    int productId;
    String productName;
    String category;

    public Product(int productId, String productName, String category) {
        this.productId = productId;
        this.productName = productName;
        this.category = category;
    }
}
```

Search.java

```
public class Search {

    // Linear Search
    public static Product linearSearch(Product[] products, String name) {
        for (Product p : products) {
            if (p.productName.equalsIgnoreCase(name)) {
                return p;
            }
        }
        return null;
    }

    // Binary Search (array must be sorted)
    public static Product binarySearch(Product[] products, String name) {
        int left = 0, right = products.length - 1;

        while (left <= right) {
            int mid = (left + right) / 2;
```

```

        int compare = products[mid].productName.compareToIgnoreCase(name);

        if (compare == 0) return products[mid];
        else if (compare < 0) left = mid + 1;
        else right = mid - 1;
    }
    return null;
}
}

```

Main.java

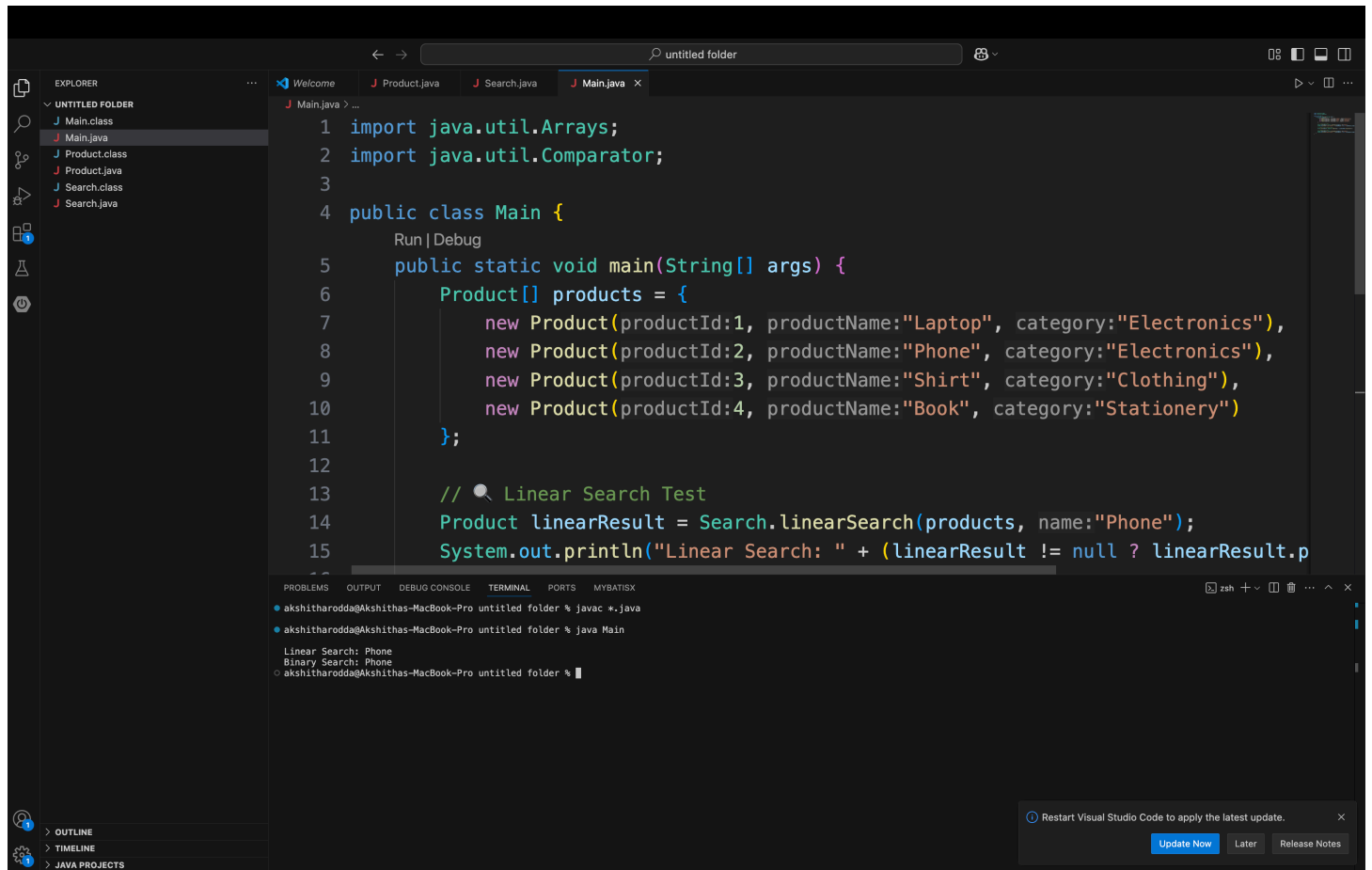
```

import java.util.Arrays;
import java.util.Comparator;

public class Main {
    public static void main(String[] args) {
        Product[] products = {
            new Product(1, "Laptop", "Electronics"),
            new Product(2, "Phone", "Electronics"),
            new Product(3, "Shirt", "Clothing"),
            new Product(4, "Book", "Stationery")
        };
        Product linearResult = Search.linearSearch(products, "Phone");
        System.out.println("Linear Search: " + (linearResult != null ?
linearResult.productName : "Not found"));
        Arrays.sort(products, Comparator.comparing(p
->p.productName.toLowerCase()));
        Product binaryResult = Search.binarySearch(products, "Phone");
        System.out.println("Binary Search: " + (binaryResult != null ?
binaryResult.productName : "Not found"));
    }
}

```

OUTPUT:



The screenshot shows the Visual Studio Code interface with a Java project. The Explorer panel on the left shows a folder named 'untitled folder' containing files: Main.class, Main.java, Product.class, Product.java, Search.class, and Search.java. The Main.java file is open in the editor, showing the following code:

```
1 import java.util.Arrays;
2 import java.util.Comparator;
3
4 public class Main {
5     Run | Debug
6     public static void main(String[] args) {
7         Product[] products = {
8             new Product(productId:1, productName:"Laptop", category:"Electronics"),
9             new Product(productId:2, productName:"Phone", category:"Electronics"),
10            new Product(productId:3, productName:"Shirt", category:"Clothing"),
11            new Product(productId:4, productName:"Book", category:"Stationery")
12        };
13
14        // 🔍 Linear Search Test
15        Product linearResult = Search.linearSearch(products, name:"Phone");
16        System.out.println("Linear Search: " + (linearResult != null ? linearResult.p
```

The terminal panel at the bottom shows the output of the program:

```
akshitharodda@Akshithas-MacBook-Pro: ~/untitled folder % javac *.java
akshitharodda@Akshithas-MacBook-Pro: ~/untitled folder % java Main
Linear Search: Phone
Binary Search: Phone
akshitharodda@Akshithas-MacBook-Pro: ~/untitled folder %
```

An update notification is visible in the bottom right corner: "Restart Visual Studio Code to apply the latest update." with buttons for "Update Now", "Later", and "Release Notes".

The output is:

Linear Search: Phone

Binary Search: Phone

Exercise 7: Financial Forecasting

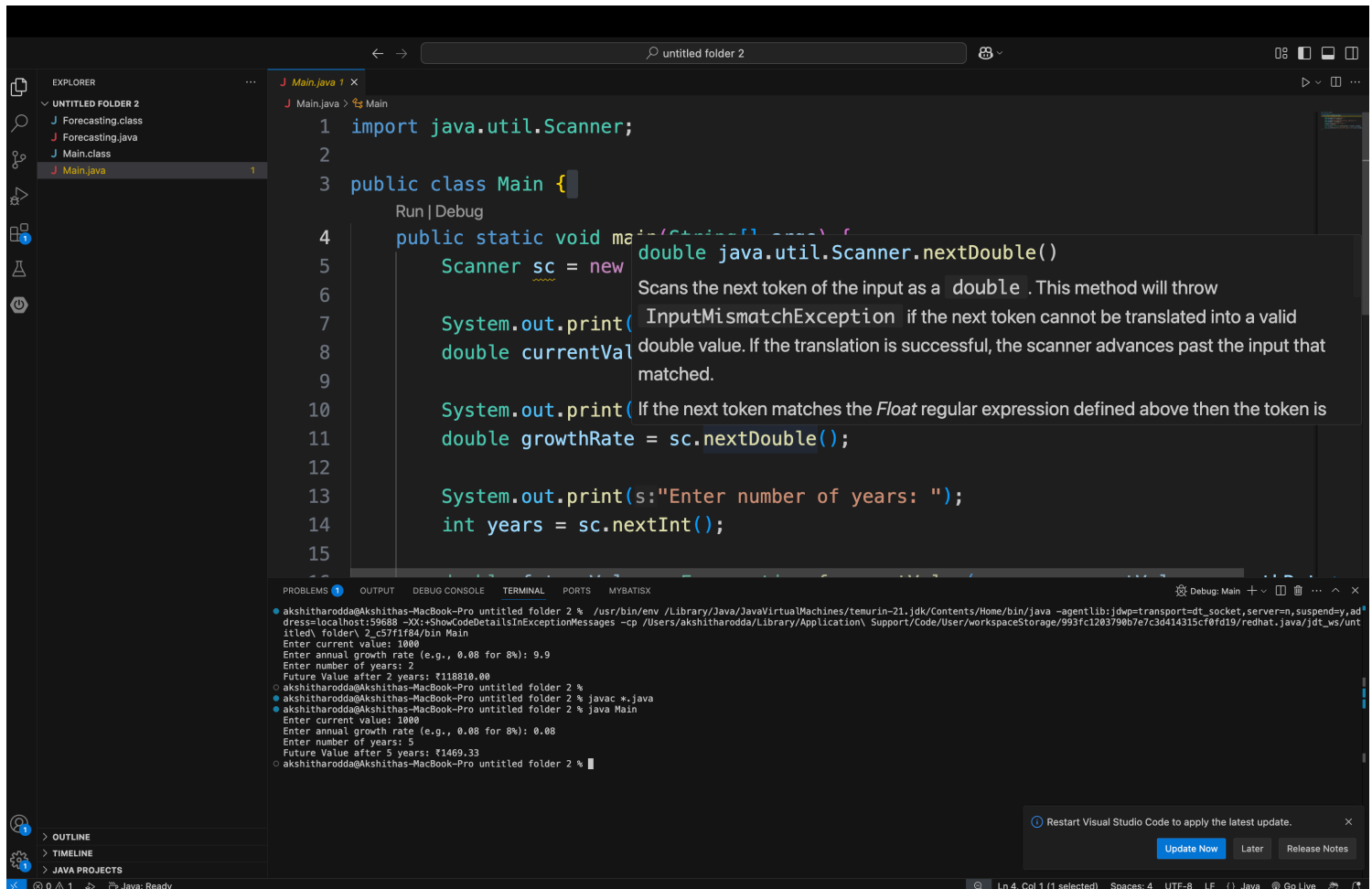
Forecasting.java

```
public class Forecasting {  
    public static double forecastValue(int years, double currentValue, double  
growthRate) {  
        if (years == 0) {  
            return currentValue;  
        }  
        return forecastValue(years - 1, currentValue, growthRate) * (1 +  
growthRate);  
    }  
}
```

Main.java

```
import java.util.Scanner;  
  
public class Main {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
  
        System.out.print("Enter current value: ");  
        double currentValue = sc.nextDouble();  
  
        System.out.print("Enter annual growth rate (e.g., 0.08 for 8%): ");  
        double growthRate = sc.nextDouble();  
  
        System.out.print("Enter number of years: ");  
        int years = sc.nextInt();  
  
        double futureValue = Forecasting.forecastValue(years, currentValue,  
growthRate);  
  
        System.out.printf("Future Value after %d years: ₹%.2f\n", years,  
futureValue);  
    }  
}
```

OUTPUT:



The screenshot shows the Visual Studio Code editor with a Java file named `Main.java` open. The code defines a `Main` class with a `main` method that prompts the user for a current value, an annual growth rate, and a number of years, then calculates and prints the future value. The output window at the bottom shows the execution results for three different input sets.

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         System.out.print("Enter current value: ");
7         double currentValue = sc.nextDouble();
8         System.out.print("Enter annual growth rate (e.g., 0.08 for 8%): ");
9         double growthRate = sc.nextDouble();
10        System.out.print("Enter number of years: ");
11        int years = sc.nextInt();
12    }
13 }
14
15
```

Output:

```
akshitharodda@Akshithas-MacBook-Pro: ~/untitled folder 2 % /usr/bin/env /Library/Java/JavaVirtualMachines/temurin-21.jdk/Contents/Home/bin/java -agentLib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:59688 -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/akshitharodda/Library/Application\ Support/Code/User/workspaceStorage/993fc1203790b7e7c3d414315cf0fd19/redhat.java/jdt_ws/untitled folder 2_c57f1f84/bin Main
Enter current value: 1000
Enter annual growth rate (e.g., 0.08 for 8%): 9.9
Enter number of years: 2
Future Value after 2 years: ₹118810.00
akshitharodda@Akshithas-MacBook-Pro: ~/untitled folder 2 % javac *.java
akshitharodda@Akshithas-MacBook-Pro: ~/untitled folder 2 % java Main
Enter current value: 1000
Enter annual growth rate (e.g., 0.08 for 8%): 0.08
Enter number of years: 5
Future Value after 5 years: ₹1469.33
akshitharodda@Akshithas-MacBook-Pro: ~/untitled folder 2 %
```

The output is:

Enter current value: 1000

Enter annual growth rate (e.g., 0.08 for 8%): 0.08

Enter number of years: 5

Future Value after 5 years: ₹1469.33