WEEK-1 HANDS ON

Design Patterns and principles

Exercise 1: Implementing the Singleton Pattern

- Created a java project named “SingletonPatternExample” in VS Code IDE.
- Created two java files named “Logger.java” and “LoggerMain.java”.

Logger.java

```
class Logger{
    // Declaring the instance of the class
    // private because only one instance per class
    private static Logger instance;
    // private constructor so that new instance cannot be instantiated
directy
    private Logger() {}
    // method to get the instance
    public static Logger getInstance(){
        if (instance==null){
            synchronized (Logger.class){
                if(instance==null){
                    instance = new Logger();
                }
            }
        }
        return instance;
    }
}
```

LoggerMain.java

```
public class LoggerMain{
    public static void main(String[] args) {
        Logger l1 = Logger.getInstance();
        Logger l2 = Logger.getInstance();
        System.out.println(l1==l2);
    }
}
```

Output

```
Single Instance Created: True
```

Exercise 2: Implementing the Factory Method Pattern

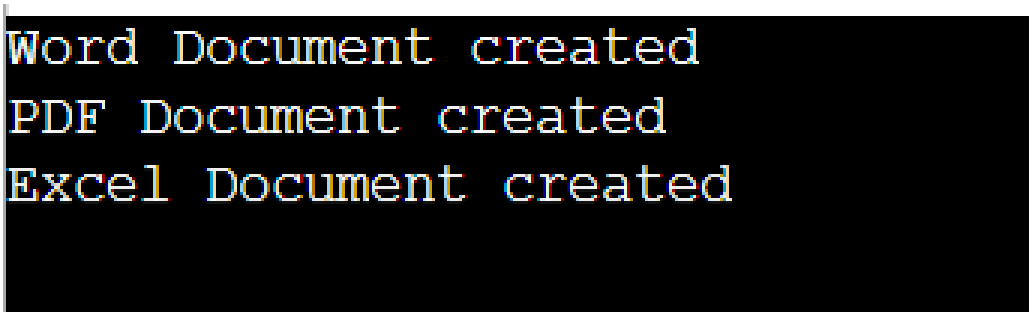
- Created a java interface named “Document.java” in VS Code.
- Create respective interface and classes for handling pdf, word and excel files.

FactoryMethodPattern.java

```java
package Week1.FactoryMethod;
interface Document{
    void createDocument();
}

class WordDocument implements Document{
    public void createDocument(){
        System.out.println("Word Document created");
    }
}

class PDFDocument implements Document{
    public void createDocument(){
        System.out.println("PDF Document created");
    }
}

class ExcelDocument implements Document{
    public void createDocument(){
        System.out.println("Excel Document created");
    }
}

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

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

class PDFDocumentFactory extends DocumentFactory {
    public Document createDocument() {
        return new PDFDocument();
    }
}

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

public class FactoryMethodPattern {
    public static void main(String[] args) {
        DocumentFactory factory;

        factory = new WordDocumentFactory();
        Document word = factory.createDocument();
        word.createDocument();

        factory = new PDFDocumentFactory();
        Document pdf = factory.createDocument();
        pdf.createDocument();

        factory = new ExcelDocumentFactory();
        Document excel = factory.createDocument();
        excel.createDocument();
    }
}
```

Output:



Algorithms_Data Structures

Exercise 2: E-commerce Platform Search Function

Big O Notation: Big O notation is used to describe the upper bound of an algorithm's time and space complexity. It helps us understand how the algorithm's performance scales with the size of the input.

Linear Search:

- Best Case – O(1): The element is found at the very beginning of the list or array.
- Average Case – O(n/2): The element is located somewhere in the middle of the list.
- Worst Case – O(n): The element is either at the end of the list or not present at all.

Binary Search:

- Best Case – O(1): The element is found right at the middle of the list or array.
- Average Case – O(log n): The list is repeatedly divided in half until the element is found.
- Worst Case – O(log n): Even in the worst scenario, the list is divided logarithmically until the element is found or determined to be absent.

Product.java

```
class Product{
    String productId;
    String productName;
    String category;
    Product(String id, String name, String category){
        this.productId = id;
        this.productName = name;
        this.category = category;
    }
    void setProductId(String id){
        this.productId = id;
    }
    void setProductName(String name){
        this.productName = name;
    }
    void setCategory(String cat){
```

```
        this.category = cat;
    }
    String getProductId(){
        return this.productId;
    }
    String getProductName(){
        return this.productName;
    }
    String getCategory(){
        return this.category;
    }
    String inString(){
        return
this.productId+"-----"+this.productName+"-----"+this.category;
    }
}
```

ECommerce.java

```
class ECommerce{
    static String name = "Amazon";
    ArrayList<Product> products;
    ECommerce(){
        this.products = new ArrayList<Product>();
    }
    ArrayList<Product> getProducts(){
        return this.products;
    }
    int equals(String s1, String s2){
        // returns 1 - if s1<s2
        // -1 - if s1>s2
        // 0 if s1==s2
        int n1 = s1.length();
        int n2 = s2.length();
        for (int i=0; i<n1 && i<n2; i++){
            if (s1.charAt(i)<s2.charAt(i)){
                return -1;
            }else if(s1.charAt(i)>s2.charAt(i)){
                return 1;
            }
```

```
        }
        return 0;
    }
    void add(Product product){
        int i = products.size()-1;
        while (i>=0 && this.equals(products.get(i).productId,
product.productId)==1){
            i-=1;
        }
        this.products.add(i+1, product);
    }
    void display(){
        for(int i=0; i<this.products.size(); i++){
            Product product = this.products.get(i);
            System.out.println(product.productId+",
"+product.productName+", "+ product.category);
        }
    }
    int linearSearch(String id){
        for(int i=0; i<products.size(); i++){
            Product product = products.get(i);
            if (product.productId.equals(id)){
                return i;
            }
        }
        return -1;
    }
    int binarySearch(String id){
        int low = 0;
        int high = this.products.size()-1;
        while (low<=high){
            int mid = (low+high)/2;
            Product product = this.products.get(mid);
            if (product.productId.equals(id)){
                return mid;
            }else if(this.equals(product.productId, id)==1){
                high = mid-1;
            }else{
                low = mid+1;
            }
```

```
        }
        return -1;
    }
}
```

ECommerceMain.java

```
public class ECommerceMain{
    public static void main(String args[]){
        ECommerce eCommerce = new ECommerce();
        Scanner scn = new Scanner(System.in);
        eCommerce.add(new Product("P102", "Laptop", "Electronics"));
        eCommerce.add(new Product("P101", "Book", "Stationery"));
        eCommerce.add(new Product("P103", "Smartphone", "Electronics"));
        eCommerce.add(new Product("P105", "Shoes", "Footwear"));
        eCommerce.add(new Product("P104", "T-shirt", "Apparel"));

        System.out.println("Enter the id to be searched");
        String key = scn.nextLine();

        System.out.println("Linear Search: ");
        int ind = eCommerce.linearSearch(key);
        if (ind!=-1){
            System.out.println("\tProduct found:
"+eCommerce.products.get(ind).inString());
        }else{
            System.out.println("\tProduct not found");
        }

        System.out.println("Binary Search: ");
        ind = eCommerce.binarySearch(key);
        if (ind!=-1){
            System.out.println("\tProduct found:
"+eCommerce.products.get(ind).inString());
        }else{
            System.out.println("\tProduct not found");
        }
    }
}
```

Output:

```
Enter the id to be searched
P101
Linear Search:
        Product found: P101-----Book-----Stationery
Binary Search:
        Product found: P101-----Book-----Stationery
```

For this example, since the size of the list is small, it is preferred to use linear search.

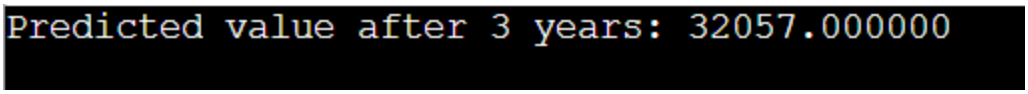
Exercise 7: Financial Forecasting

Recursion: Recursion is a programming technique where a function calls itself to solve smaller instances of a problem. It is particularly useful for breaking complex tasks into simpler subproblems. By doing so, recursion allows us to handle the logic in a clean and structured way without repeatedly writing the same code.

FinancialForecasting.java

```
public class FinancialForecasting {
   public static double predict(int currVal, int time) {
       if (time == 0) {
           return currVal;
       }
       return predict(currVal*4+time, time-1);
   }
   public static void main(String[] args) {
       int currVal = 500;
       int time = 3;
       double futureValue = predict(currVal, time);
       System.out.printf("Predicted value after %d years: %f", time,
futureValue);
   }
}
```

Output:



Time complexity = O(n)
We can optimize the recursive solution by simply following an iterative process or storing the values from previous calls into an array.