WEEK -01

1> DESIGN PATTERN

- **Exercise 1: Implementing the Singleton Pattern**

➔CODE:

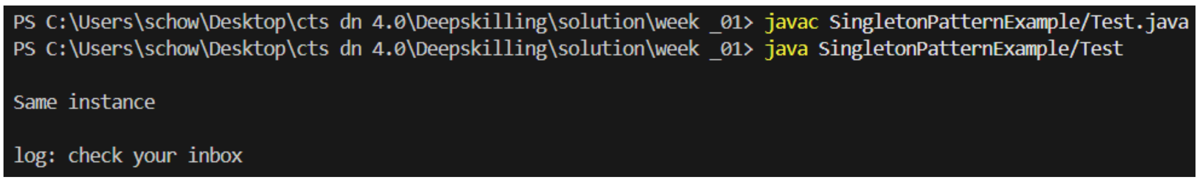
Logger.java

```
public class Logger {
  private static Logger instance;

  private Logger() {

  }

  public static Logger getInstance() {
    if (instance == null) {
      instance = new Logger();
    }
    return instance;
  }

  public void log(String msg) {
    System.out.println("log: " + msg);
  }
}
```

Test.java

```
public class Test {
  public static void main(String[] args) {
    Logger lg1 = Logger.getInstance();

    Logger lg2 = Logger.getInstance();

    if(lg1 == lg2){
      System.out.println("\nSame instance\n");
    } else {
      System.out.println("\ndifferent instance\n");
    }

    lg1.log("check your inbox\n");
  }
}
```

O/P:



```
PS C:\Users\schow\Desktop\cts dn 4.0\Deepskilling\solution\week _01> javac SingletonPatternExample/Test.java
PS C:\Users\schow\Desktop\cts dn 4.0\Deepskilling\solution\week _01> java SingletonPatternExample/Test

Same instance

log: check your inbox
```

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

➔CODE:

Document.java

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

WordDocument.java

```
public class WordDocument implements Document {
  @Override
  public void open() {
    System.out.println("Word is opening\n");
  }

  @Override
  public void close() {
    System.out.println("Word is closing\n");
  }
}
```

PdfDocument.java

```
public class PdfDocument implements Document{
  @Override
  public void open() {
    System.out.println("Pdf is opening\n");
  }

  @Override
  public void close() {
    System.out.println("Pdf is closing\n");
  }
}
```

ExcelDocument.java

```
public class ExcelDocument implements Document{
  @Override
  public void open() {
    System.out.println("Excel is opening\n");
  }

  @Override
  public void close() {
    System.out.println("Excel is closing\n");
  }
}
```

DocumentFactory.java

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

WordDocumentFactory.java

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

PdfDocumentFactory.java

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

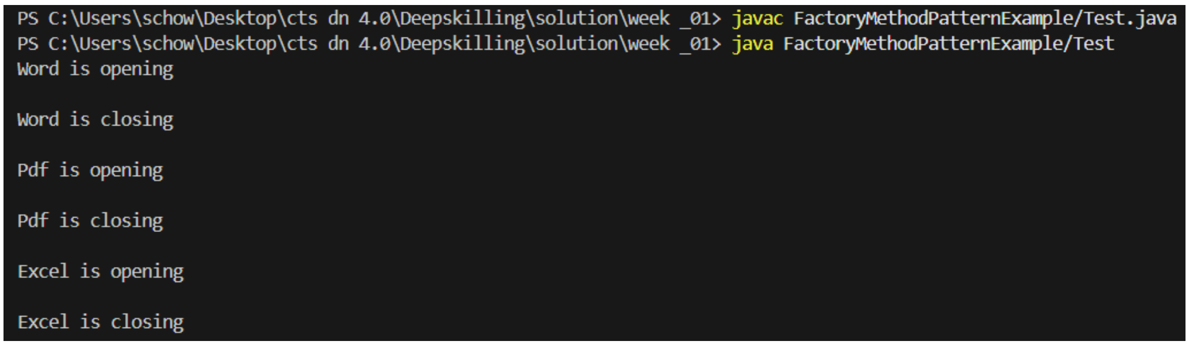
ExcelDocumentFactory.java

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

Test.java

```
public class Test {
  public static void main(String[] args) {
    WordDocumentFactory wordFactory = new WordDocumentFactory();
    Document word = wordFactory.createDocument();
    word.open();
    word.close();

    PdfDocumentFactory pdfFactory = new PdfDocumentFactory();
    Document pdf = pdfFactory.createDocument();
    pdf.open();
    pdf.close();

    ExcelDocumentFactory excelFactory = new ExcelDocumentFactory();
    Document excel = excelFactory.createDocument();
    excel.open();
    excel.close();
  }
}
```

O/P:



```
PS C:\Users\schow\Desktop\cts dn 4.0\Deepskilling\solution\week _01> javac FactoryMethodPatternExample/Test.java
PS C:\Users\schow\Desktop\cts dn 4.0\Deepskilling\solution\week _01> java FactoryMethodPatternExample/Test
Word is opening

Word is closing

Pdf is opening

Pdf is closing

Excel is opening

Excel is closing
```

2> **Algorithms and Data Structures:**

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

➔Solution:

- Big-oh: describes the upper bound (worst-case) of a function when the argument tends towards larger values. It is used to classify algo according to how their run-time or space requirement grow as input(n) grows.

Name	Best-case	Avg-case	Worst-case
Linear Search	O(1)	O(n)	O(n)
Binary Search	O(1)	O(logn)	O(logn)

Code:

BinarySearch.java

```
public class BinarySearch {
  public static Product binSearch(Product[] pdts, int id) {
    int s=0;
    int e = pdts.length - 1;
    while (s <= e) {
      int mid = (s + e) / 2;
      if (pdts[mid].productId == id) {
        return pdts[mid];
      } else if (pdts[mid].productId < id) {
        s = mid + 1;
      } else {
        e = mid - 1;
      }
    }
    return null;
  }
}
```

LinearSearch.java

```
public class LinearSearch {
  public static Product linSearch(Product[] pdts, int id){
    for (Product pdt : pdts) {
      if (pdt.productId == id) {
        return pdt;
      }
    }
    return null;
  }
}
```

BubbleSort.java

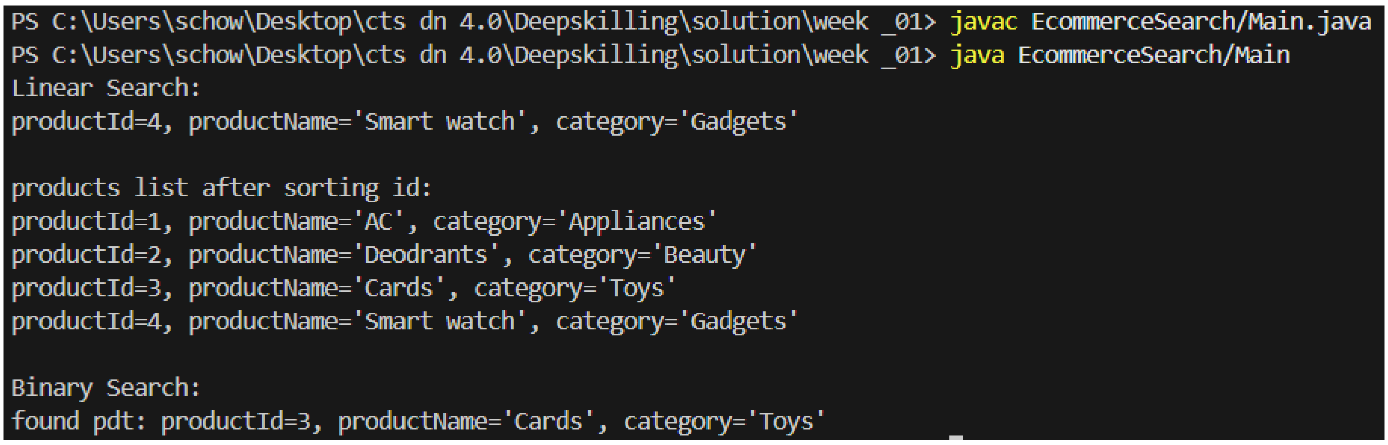
```
public class BubbleSort {
  static void sort(Product[] arr) {
    int n = arr.length;
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n - i - 1; j++) {
        if (arr[j].productId > arr[j + 1].productId) {
          Product p = (arr[j+1]);
          arr[j+1] = arr[j];
          arr[j] = p;
        }
      }
    }
  }

  static void printArr(Product[] arr) {
    for (int i = 0; i < arr.length; i++) {
      System.out.println(arr[i]);
    }
  }
}
```

Main.java

```
public class Main {
  public static void main(String[] args) {
    Product[] pdts = {
      new Product(1, "AC", "Appliances"),
      new Product(3, "Cards", "Toys"),
      new Product(2, "Deodrants", "Beauty"),
      new Product(4, "Smart watch", "Gadgets")
    };
    System.out.println("Linear Search: ");
    Product pdt = LinearSearch.linSearch(pdts, 4);
    System.out.println(pdt);
    System.out.println();
```

```
        BubbleSort.sort(pdts);
        System.out.println("products list after sorting id:");
        BubbleSort.printArr(pdts);
        System.out.println();

        System.out.println("Binary Search: ");
        Product pdt2 = BinarySearch.binSearch(pdts, 3);
        System.out.println("found pdt: "+ pdt2);
    }
}
```

O/P:



```
PS C:\Users\schow\Desktop\cts dn 4.0\Deepskilling\solution\week _01> javac EcommerceSearch/Main.java
PS C:\Users\schow\Desktop\cts dn 4.0\Deepskilling\solution\week _01> java EcommerceSearch/Main
Linear Search:
productId=4, productName='Smart watch', category='Gadgets'

products list after sorting id:
productId=1, productName='AC', category='Appliances'
productId=2, productName='Deodrants', category='Beauty'
productId=3, productName='Cards', category='Toys'
productId=4, productName='Smart watch', category='Gadgets'

Binary Search:
found pdt: productId=3, productName='Cards', category='Toys'
```

- Both linear and binary search have their pros and cons.
- In case of linear search no sorting is required but it's worst case time complexity[O(n)] is more than binary search[O(logn)] . So, it may perform bad to large number of datasets.
- In case of binary search a sorting is needed beforehand. But it's avg and worst case time complexity is better than linear search. So, it may perform good to large number of datasets, but if frequent updation occurs in database then it may take more time due to running consequent sorting.
- So, the searching algo is solely dependent on the application requirement. In my case, E-commerce will need frequent updation so I will go for linear search.

- **Exercise 7: Financial Forecasting**

➔ **Solution:**

- Recursion is a technique of programming where on function calls itself again and again until a base case is satisfied. It breaks larger problem into smaller number of problems using recursion stack.

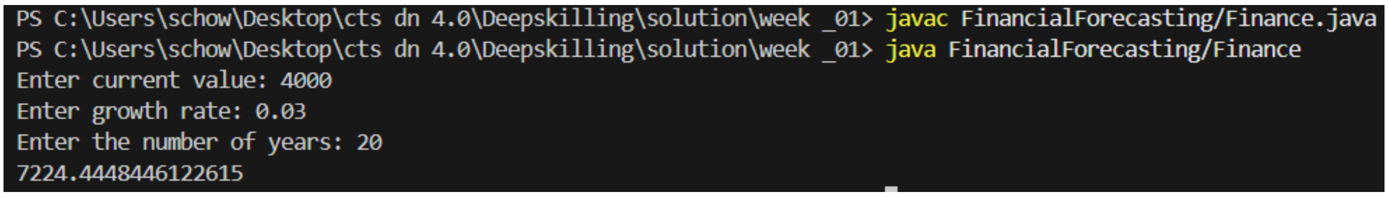
  Two step is there: i> selecting a base case: a condition to stop infinite looping and return certain outputs, ii> selecting a recursive case where function calls itself with modified arguments.
- Recursion simplifies problems which have natural recursive structures(e.g, factorial, tree traversal) by breaking down the problem in smaller subproblems and solve the problem with ease.

Code:

Finance.java

```
import java.util.Scanner;

public class Finance {
  public static double calculate(double val, double rate, int year) {
    if (year == 0) {
      return val;
    }
    return calculate((val + val * rate), rate, year - 1);
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter current value: ");
    float currentVal = sc.nextFloat();

    System.out.print("Enter growth rate: ");
    float rate = sc.nextFloat();

    System.out.print("Enter the number of years: ");
    int year = sc.nextInt();

    double val = calculate(currentVal, rate, year);
    System.out.println(val);
  }
}
```

O/P:



```
PS C:\Users\schow\Desktop\cts dn 4.0\Deepskilling\solution\week _01> javac FinancialForecasting/Finance.java
PS C:\Users\schow\Desktop\cts dn 4.0\Deepskilling\solution\week _01> java FinancialForecasting/Finance
Enter current value: 4000
Enter growth rate: 0.03
Enter the number of years: 20
7224.4448446122615
```

- Time complexity: O(n), where n is the number of years up to which growth rate is performed

- We can use memoization where we save function call for a specific input and when that function is called with the same input we can retrieve it in constant time.