JAVA FSE Mandatory Module:

MODULE1:

Exercise1: Implementing theSingleton Pattern:

Code:

Singleton.java:

public class Singleton {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

         Logger logger2 = Logger.getInstance();

        if (logger1 == logger2) {

            System.out.println(" same instance");

        } else {

            System.out.println(" multiple instances");

        }

    }

}

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();

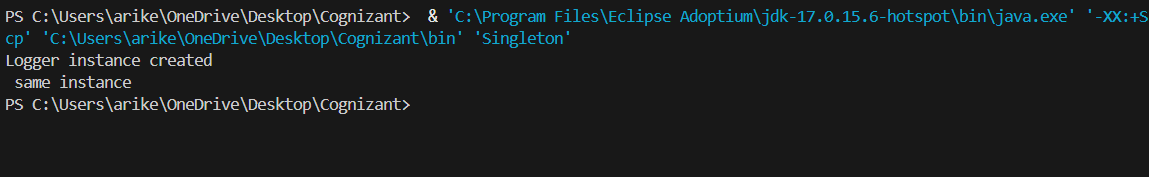
        }

        return instance;

    }

}

Output:



Exercise 2: Implementing the Factory Method Pattern:

Code:

Document package:

1.Document.java:

package documents;

public interface Document {

void openFile();

}

2.ExcelDocument.java:

package documents;

public class ExcelDocument implements Document {

@Override

public void openFile(){

System.out.println("exelsheet");

}

}

3.PdfDocument.java:

package documents;

public class PdfDocument implements Document {

@Override

public void openFile() {

    System.out.println("Opening PDF file");

    }

}

4.WordDocument.java:

package documents;

public class WordDocument implements Document {

@Override

public void openFile() {

System.out.println("Opening Word file");

}

}

factory package:

1.DocumentFactory.java:

package factory;

import documents.Document;

public abstract class DocumentFactory {

public abstract Document createDocument();

public void processDocument() {

Document doc1 = createDocument();

doc1.openFile();

}

}

2.PdfFactory.java:

package factory;

import documents.Document;

import documents.PdfDocument;

public class PdfFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

3.ExcelFactory.java:

package factory;

import documents.Document;

import documents.ExcelDocument;

public class ExcelFactory extends DocumentFactory {

     @Override

    public Document createDocument() {

        return new ExcelDocument();

}

}

4.WordFactory.java:

package factory;

import documents.WordDocument;

import documents.Document;

public class WordFactory extends DocumentFactory{

@Override

public Document createDocument() {

return new WordDocument();

}

}

Test.java:

import factory.\*;

public class Test{

public static void main(String[] args) {

DocumentFactory wordFactory = new WordFactory();

DocumentFactory pdfFactory = new PdfFactory();

DocumentFactory excelFactory = new ExcelFactory();

System.out.println(" Word document:");

wordFactory.processDocument();

System.out.println("\nPDF document:");

pdfFactory.processDocument();

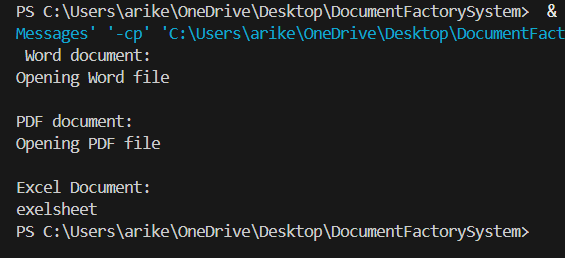
System.out.println("\nExcel Document:");

excelFactory.processDocument();

}

}

Output:



MODULE2:

Exercise 2: E-commerce Platform Search Function:

Big-0 Notation:

Two functions f(n) and g(n), we say that f(n) is O(g(n)) if there exist constants c > 0 and n0 >= 0 such that f(n) <= c\*g(n) for all n >= n0.

Performance:

Linear and Binary comparision: Best case: both has O(1), in average case O(n) for linear search and O(logn) for binary search. Coming to worst cases O(n) for linear and O(logn) for binary search.

* For making an array to implement sorting techniques. I thought of taking productid as integer and then searching technique is implemented.

Code:

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;

    }

    public static Product linear(Product[] products, int targetId) {

        for (int i = 0; i < products.length; i++) {

            if (products[i].productId == targetId) {

                return products[i];

            }

        }

        return null;

    }

public static void sortProducts(Product[] products) {

        int n = products.length;

        for (int i = 0; i < n - 1; i++) {

            for (int j = 0; j < n - i - 1; j++) {

                if (products[j].productId > products[j + 1].productId) {

                    Product temp = products[j];

                    products[j] = products[j + 1];

                    products[j + 1] = temp;

}

}

   }

 }

    public static Product binary(Product[] products, int targetId) {

        int low = 0;

        int high = products.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            if (products[mid].productId == targetId) {

                return products[mid];

            } else if (products[mid].productId < targetId) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return null;

    }

    // Main method

    public static void main(String[] args) {

        Product[] products = new Product[5];

        products[0] = new Product(105, "Keyboard", "Electronics");

        products[1] = new Product(101, "Pen", "Stationery");

        products[2] = new Product(103, "Notebook", "Stationery");

        products[3] = new Product(102, "Mouse", "Electronics");

        products[4] = new Product(104, "Water Bottle", "Kitchen");

        int target = 100;

        Product result = Product.linear(products, target);

        System.out.println("linear search:");

        if (result != null) {

            System.out.print("Product Found ");

        } else {

            System.out.println("Product not found");

        }

        sortProducts(products);

        int target1 = 101;

        System.out.println("Binary search:");

        Product result1 = Product.binary(products, target1);

        if (result1 != null) {

            System.out.print("Product Found ");

        } else {

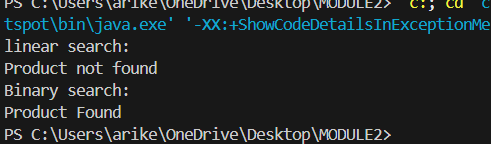
            System.out.println("Product not found ");

        }

    }

}

Output:



Performance Analysis:

Binary Search is more optimal compared to linear search as it has time complexity of O(log n). Usually the best way for searching the elements is Binary search as they are differed in Time complexity and comparing it with space complexity both has same space complexity.

Exercise 7: Financial Forecasting:

Code:

public class Interest {

    public static double power(double base, int exp) {

        if (exp == 0) {

            return 1;

        } else {

            return base \* power(base, exp - 1);

        }

    }

    public static double interest(double P, double annual, int years, int N) {

        double rate = (annual / 100) / N;

        int totaltime = N \* years;

        double growth = power(1 + rate, totaltime);

        return P \* growth;

    }

    public static void main(String[] args) {

        double P = 1000.0;

        double annual = 6;

        int years = 4;

        int N = 365;

        double futureValue = interest(P, annual, years, N);

        System.out.printf("The growth is:%f", futureValue);

    }

}

Output:

