**WEEK 1 HANDS ON**

**R DARWIN RAJA KUMAR (312422205026)**

**St. Joseph’s Institute of Technology**

**Design Principles 1.Implementing the singleton pattern Code:**

package com.deepskilling;

public class MyLogger { private static MyLogger *instance*; public static MyLogger getInstance() {

if(*instance*==null){

*instance*=new MyLogger();

}

return *instance*;

} }

package com.deepskilling;

//TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or // click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter. public class Main {

public static void main(String[] args) { MyLogger log1=MyLogger.*getInstance*(); MyLogger log2=MyLogger.*getInstance*(); if(log1==log2){

System.*out*.println("Same logger used");

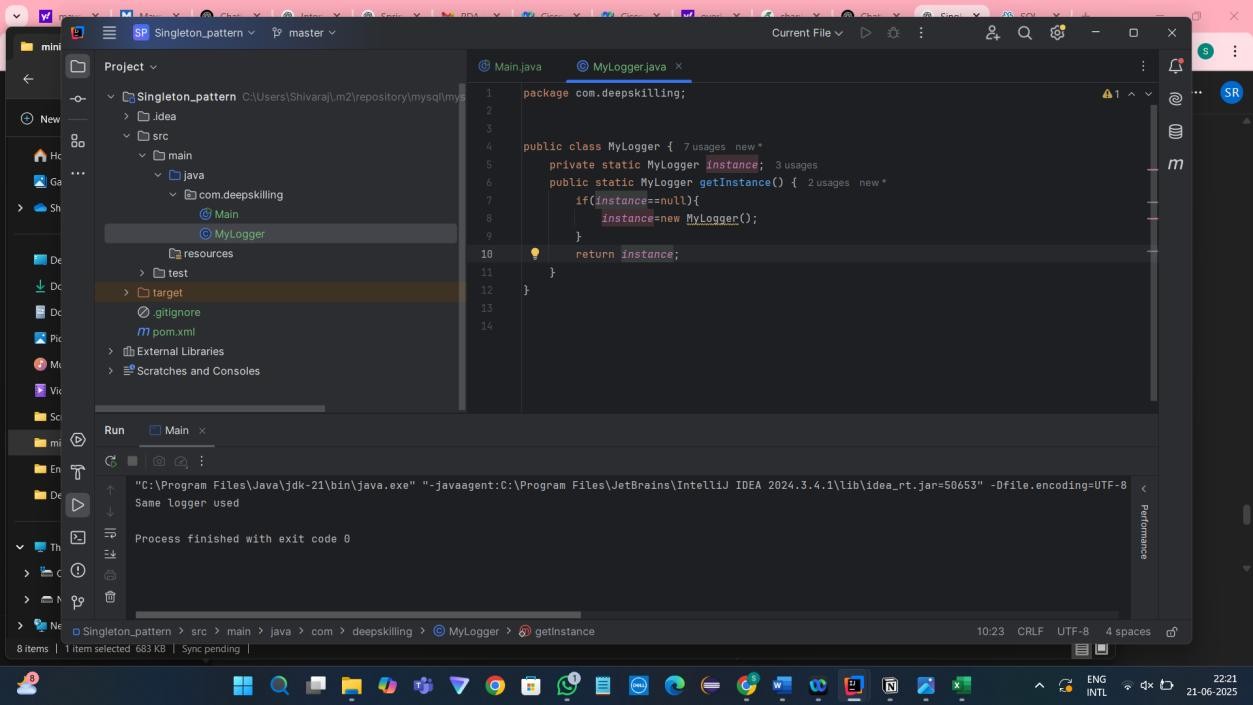
}else{

System.*out*.println("Different loggers used");

}

} }

**Output:**



**2.Factory Method Pattern**

**Code:**

package com.deepskilling;

//TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or // click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter.

public class Main {

public static void main(String[] args) {

DocFactory wordFactory=new WordFactory(); Doc word=wordFactory.create();

word.open();

DocFactory pdfFactory=new PdfFactory(); Doc pdf=pdfFactory.create();

pdf.open();

DocFactory excelFactory=new ExcelFactory(); Doc excel=excelFactory.create();

excel.open();

} }

package com.deepskilling;

public interface Doc {

void open();

}

package com.deepskilling;

public class Word implements Doc {

public void open() {

System.*out*.println("opening word");

} }

package com.deepskilling;

public class Pdf implements Doc { public void open() {

System.*out*.println("opening pdf");

}

}

package com.deepskilling;

public class Excel implements Doc {

public void open() {

System.*out*.println("opening excel");

}

}

package com.deepskilling;

public abstract class DocFactory {

public abstract Doc create();

}

package com.deepskilling;

public class WordFactory extends DocFactory {

public Doc create() {

return new Word();

}

}

package com.deepskilling;

public class PdfFactory extends DocFactory {

public Doc create() { return new Pdf();

}

}

package com.deepskilling;

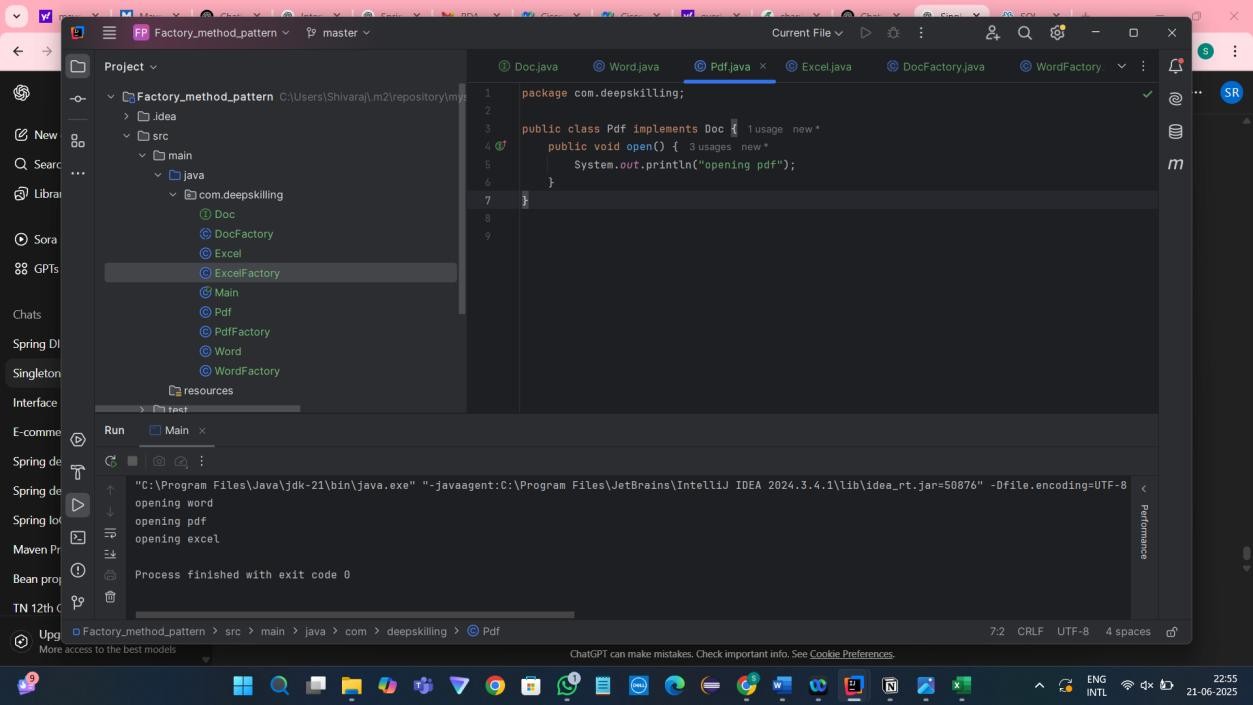
public class ExcelFactory extends DocFactory {

public Doc create() {

return new Excel();

} }

**Output:**



**Data Structure and Algorithm Hands On**

**3. E\_Commerce Platform Search Function**

Big O notation: Big O notation describes the upper bound of an algorithms space and time complexity with the respective input size n. It helps us to understand the performance of the algorithm with respective to time and space there are three cases here best,average and worst cases.

Best case: When the element is found immediately Average case: when the element is found in middle region.

Worst Case: when the element is found at the last region.

**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;

} }

//TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or // click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter. import java.util.Arrays;

import java.util.Comparator;

public class ECommerceSearch {

public static Product linearSearch(Product[] products,String targetName) {

for(Product product:products) {

if(product.productname.equals(targetName)) { return product;

}

}

return null;

}

public static Product binarySearch(Product[] products,String targetName) {

int left=0,right=products.length-1;

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

result=products[mid].productname.compareTo(targetName);

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

}

return null;

}

public static void main(String[] args) {

Product[] products={ new Product(1,"Laptop","Electronics"),new

Product(2,"Phone","Electronics"),new

Product(3,"Shirt","Apparel"),new

Product(4,"Shoes","Footwear"),new

Product(5,"Watch","Accessories")};

Product result1=*linearSearch*(products,"Shoes"); if(result1!=null) System.*out*.println("Linear Search

Result:"+result1.productname);

Arrays.*sort*(products,Comparator.*comparing*(p-

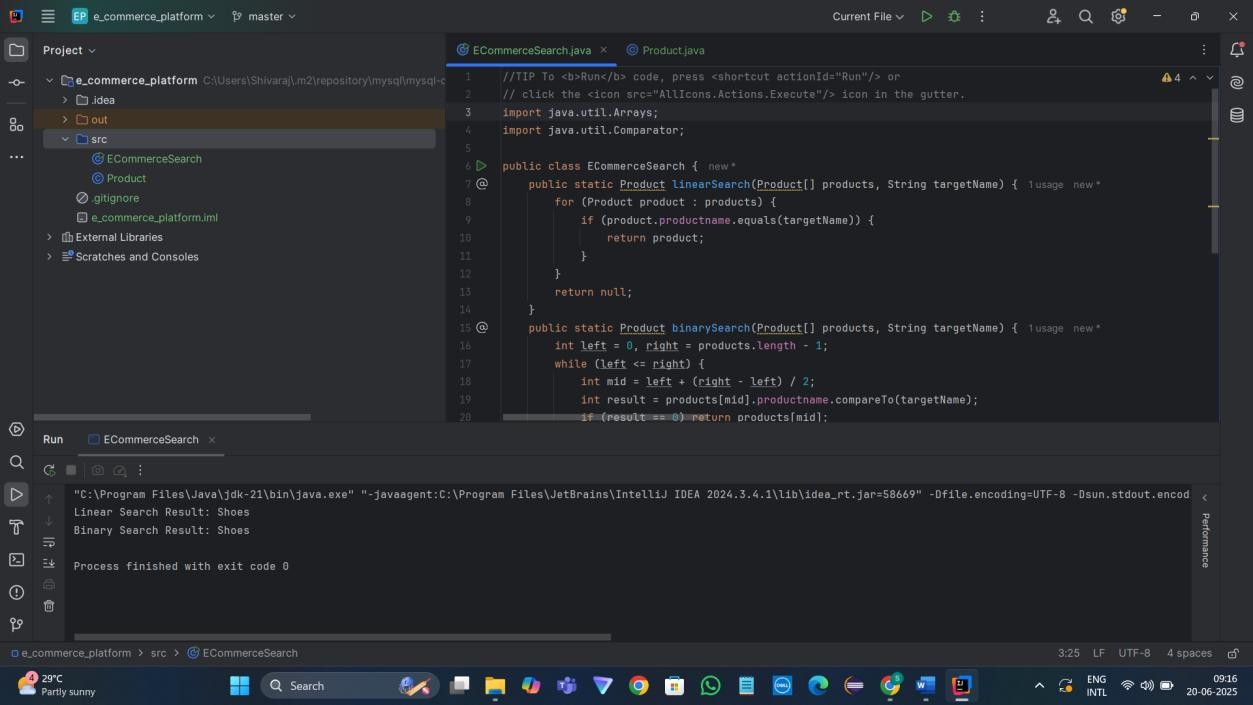
>p.productname));

Product result2=*binarySearch*(products,"Shoes"); if(result2!=null) System.*out*.println("Binary Search

Result:"+result2.productname);

} }

Output:



Time Complexity of linear search and binary search

Linear Search

Best case: O(1)

Average case: O(n/2)

Worst case: O(n)

Binary Search:

Best case: O(1)

Average case: O(log n)

Worst case: O(log n)

Best Algorithm that fits this program

In this program E\_Commerce Application the most suitable Algorithm is Binary Search because linear search is suitable for small data sets but e\_commerce application is a growing application so for large data sets so binary search is the most suitable algorithm

**4.Financial Forecasting**

Recursion

Recursion is a concept where function call itself to solve the specific problem where the code logic need to be executed repeatedly.

**Code:**

public class FinancialForecasting {

public static double predict(double currentvalue,double growthrate,int years) {

if(years==0) return currentvalue;

return *predict*(currentvalue\*(1+growthrate),growthrate,years-1);

}

public static void main(String[] args) { double currentvalue = 10000; double growthrate = 0.08;

int years = 5;

double futurevalue = *predict*(currentvalue,growthrate,years);

System.*out*.println("Future Value:"+futurevalue);

// Can optimize this recurssion using mathematical formula where we can avoid unnecessary recurssion and excessive computation

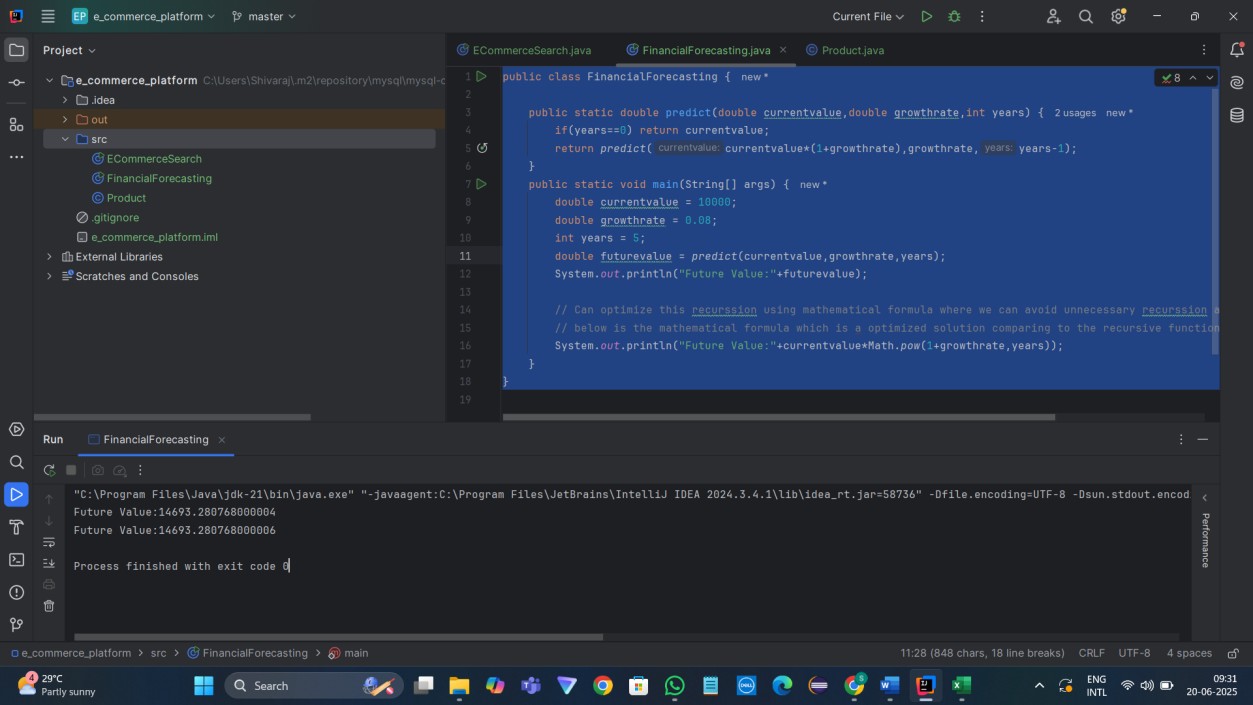
// below is the mathematical formula which is a optimized solution comparing to the recursive function

System.*out*.println("Future

Value:"+currentvalue\*Math.*pow*(1+growthrate,years));

} }

**Output:**



Time Complexity:

Here in this java code the time complexity for recursion is O(n) where the recursion call is happening base on the no of years.

Optimization Approach:

The optimization approach for this problem is to use mathematical formula where we can get in output in O(1) time complexity which avoid unnecessary recursive calls.

The formula used in this problem is:

Future Value = Current Value\*(1+growth rate)^years.