**WEEK 1**

**DESIGN PATTERNS AND PRINCIPLES:**

#Exercise 1: Implementing the Singleton Pattern

Code:

Logger.java

package SingletonPattern;

public class Logger {

private static Logger *instance*;

private Logger() {

System.***out***.println("Logger instance is created");

}

public static Logger getInstance() {

if(*instance*==null)

{

*instance*=new Logger();

}

return *instance*;

}

public void getMessage(String input)

{

System.***out***.println("Message: "+input);

}

}

TestLogger.java

package SingletonPattern;

public class TestLogger {

public static void main(String[] args)

{

Logger test1=Logger.getInstance();

System.out.println("1st message");

Logger test2=Logger.getInstance();

System.out.println("2nd message");

if(test1==test2){

System.out.println("Singleton pattern is successful");

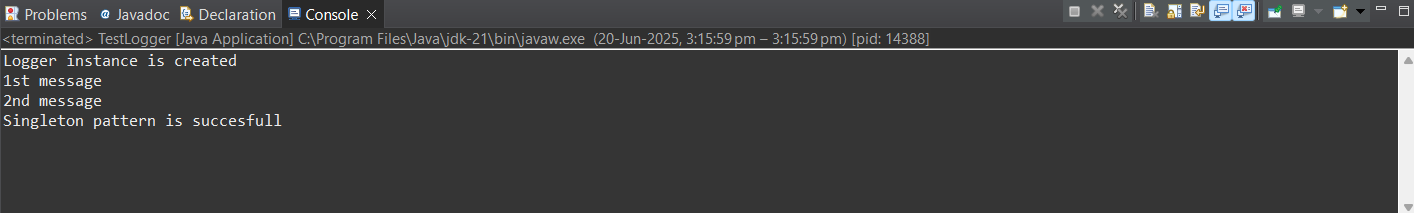
}else

System.out.println("More instances are created");

}

}

Output:



#Exercise 2: Implementing the Factory Method Pattern

Code:

Document.java

public interface Document {

void open();

}

DocumentFactory.java

public abstract class DocumentFactory {

public abstract Document createDocument();

}

WordDocument.java

public class WordDocument implements Document{

*@Override*

public void open()

{

System.***out***.println("Word document opening.");

}

}

ExcelDocument.java

public class ExcelDocument implements Document{

*@Override*

public void open()

{

System.***out***.println("Excel document opening.");

}

}

PdfDocument.java

public class PdfDocument implements Document{

*@Override*

public void open()

{

System.***out***.println("Pdf document opening.");

}

}

WordFactory.java

public class WordFactory extends DocumentFactory{

*@Override*

public Document createDocument()

{

return new WordDocument();

}

}

ExcelFactory.java

public class ExcelFactory extends DocumentFactory{

*@Override*

public Document createDocument()

{

return new ExcelDocument();

}

}

PdfFactory.java

public class PdfFactory extends DocumentFactory{

*@Override*

public Document createDocument()

{

return new PdfDocument();

}

}

TestFactoryPattern.java

public class TestFacoryPattern {

public static void main(String[] args)

{

DocumentFactory wFactory=new WordFactory();

Document word=wFactory.createDocument();

word.open();

DocumentFactory pFactory=new PdfFactory();

Document pdf=pFactory.createDocument();

pdf.open();

DocumentFactory eFactory=new ExcelFactory();

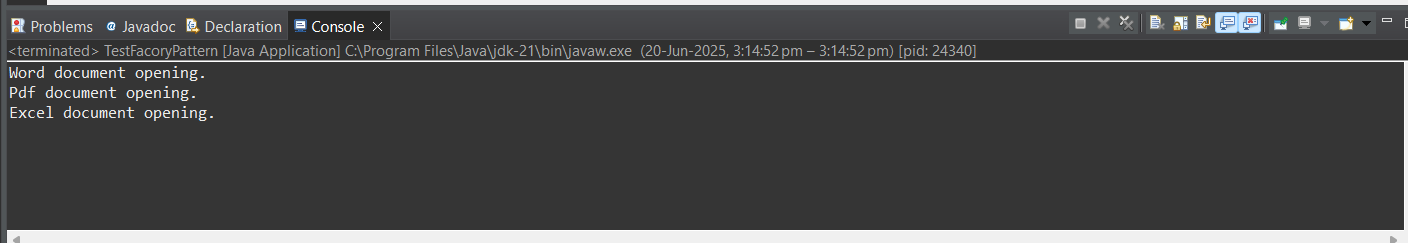
Document excel=eFactory.createDocument();

excel.open();

}

}

Output:



**DATA STRUCTURES AND ALGORITHMS:**

#Exercise 2: E-commerce Platform Search Function

Code:

Product.java

package Searching;

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;

}

public String toString() {

return productId+" "+productName+" "+category;

}

}

BinarySearch.java

package Searching;

import java.util.Arrays;

import java.util.Comparator;

public class BinarySearch {

public static Product binarysearch(Product[] products,String Name) {

Arrays.sort(products,Comparator.comparing(p->p.productName.toLowerCase()));

int left=0;

int right=products.length-1;

while(left<=right) {

int mid=(left+right)/2;

int comp=(products[mid].productName.compareToIgnoreCase(Name));

if(comp==0) {

return products[mid];

}else if(comp<0) {

left=mid+1;

}else {

right=mid-1;

}

}

return null;

}

}

LinearSearch.java

package Searching;

public class LinearSearch {

public Product ls(Product[] products,String Name) {

for(Product p:products) {

if(p.productName.equalsIgnoreCase(Name)) {

return p;

}

}

return null;

}

}

Main.java

package Searching;

public class Main {

public static void main(String args[]) {

Product[] products = new Product[4];

products[0] = new Product(101, "Shoes", "Footwear");

products[1] = new Product(102, "Laptop", "Electronics");

products[2] = new Product(103, "Camera", "Electronics");

products[3] = new Product(104, "Watch", "Accessories");

String search="Camera";

LinearSearch a=new LinearSearch();

Product lans=a.ls(products,search);

Product bans=BinarySearch.*binarysearch*(products, "Watch");

System.***out***.println("Output from Linear Search");

System.***out***.println(lans);

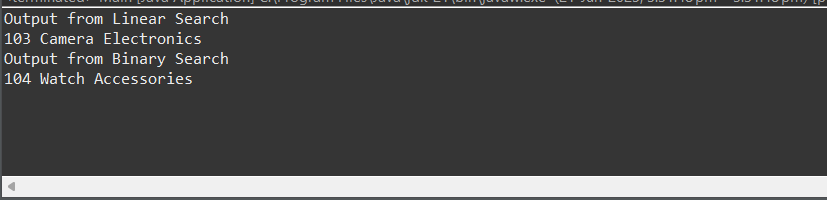
System.***out***.println("Output from Binary Search");

System.***out***.println(bans);

}

}

Output:



#Exercise 7: Financial Forecasting

Code:

Forecast.java

package FinancialForecast;

public class Forecast {

public double forecast (double amount, double rate, Int year) {

if(year==0) {

return amount;

}

return forecast (amount, rate, year-1) \*(1+rate);

}

}

Main.java

package FinancialForecast;

import java.util.Scanner;

public class Main {

public static void main (String[] args) {

Scanner in=new Scanner (System.***in***);

double price=in.nextDouble();

double rate=in.nextDouble();

int year=in.nextInt();

Forecast fo=new Forecast();

double forecastprice=fo.forecast(price, rate, year);

System.***out***.print(forecastprice);

}

}

Output:

