**Week1 Hands-On-Exercise**

**Exercise 1: Implementing the Singleton Pattern**

**Logger.java**

package SingletonPattern.example;

public class Logger {

private static Logger *instance*;

private Logger()

{

System.*out*.println("Logger initialized");

}

public static Logger getinstance()

{

if(*instance*==null)

{

*instance* = new Logger();

}

return *instance*;

}

public void log(String message)

{

System.*out*.println("[LOG]"+message);

}

}

**Main.java**

package SingletonPattern.example;

public class Main {

public static void main(String[] args) {

Logger log1=Logger.*getinstance*();

System.*out*.println("First log message");

Logger log2=Logger.*getinstance*();

System.*out*.println("Second log message");

if(log1==log2)

{

System.*out*.println("Only one instance is created, singleton worked.");

}

else

{

System.*out*.println("Different instances are created, singleton failed.");

}

}

}

**Output:**

Logger initialized

First log message

Second log message

Only one instance is created, singleton worked.

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

**1. Common Document Interface**

**Document.java**

package com.example.factory;

public interface Document {

void open();

}

**2. Concrete Document Classes**

**WordDocument.java**

package com.example.factory;

public class WordDocument implements Document {

public void open() {

System.out.println("Opening a Word document.");

}

}

**PdfDocument.java**

package com.example.factory;

public class PdfDocument implements Document {

public void open() {

System.out.println("Opening a PDF document.");

}

}

**ExcelDocument.java**

package com.example.factory;

public class ExcelDocument implements Document {

public void open() {

System.out.println("Opening an Excel document.");

}

}

**3. Abstract Factory Class**

**DocumentFactory.java**

package com.example.factory;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**4. Concrete Factory Classes**

**WordDocumentFactory.java**

package com.example.factory;

public class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

package com.example.factory;

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

package com.example.factory;

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**5. Testing the Factory Method**

**Main.java**

package com.example.factory;

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Output:**

Opening a Word document.

Opening a PDF document.

Opening an Excel document.

**Exercise 3: E-commerce Platform Search Function**

**Product.java**

package com.example.search;

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;

}

}

**SearchService.java**

package com.example.search;

import java.util.Arrays;

import java.util.Comparator;

public class SearchService {

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

for (Product product : products) {

if (product.productName.equalsIgnoreCase(name)) {

return product;

}

}

return null;

}

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

int low = 0, high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

int cmp = products[mid].productName.compareToIgnoreCase(name);

if (cmp == 0) {

return products[mid];

} else if (cmp < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void sortByName(Product[] products) {

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

}

}

**Main.java**

package com.example.search;

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(1, "Laptop", "Electronics"),

new Product(2, "Shirt", "Fashion"),

new Product(3, "Mobile", "Electronics"),

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

new Product(5, "Watch", "Accessories")

};

Product found1 = SearchService.linearSearch(products, "Shoes");

System.out.println(found1 != null ? "Found (Linear): " + found1.productName : "Not found (Linear)");

SearchService.sortByName(products);

Product found2 = SearchService.binarySearch(products, "Shoes");

System.out.println(found2 != null ? "Found (Binary): " + found2.productName : "Not found (Binary)");

}

}

**Output:**

Found (Linear): Shoes

Found (Binary): Shoes

**Which is More Suitable for an E-commerce Platform?**

For an actual e-commerce platform with thousands (or millions) of products:

* Linear search is not suitable because it performs poorly on large data (O(n)).
* Binary search is better than linear but requires sorted data and is still limited.
* In real platforms, neither linear nor binary search is used directly. Instead, platforms use:
  + Database indexing (e.g., MySQL, PostgreSQL) etc.

**Exercise - 4: Financial Forecasting Using Recursion**

**1. Understanding Recursive Algorithms**

Recursion is a programming technique where a method calls itself to solve smaller subproblems.  
It helps simplify repetitive problems or can be broken into similar subproblems.

**Example:**  
Factorial using recursion:  
factorial(n) = n \* factorial(n - 1)

**2. Setup**

Create a method to calculate future value given:

* initialValue (starting amount)
* growthRate (e.g., 10% = 0.10)
* years (number of years to forecast)

**3. Implementation**

**Forecast.java**

package com.example.forecast;

public class Forecast {

public static double predictFutureValue(double currentValue, double growthRate, int years) {

if (years == 0) {

return currentValue;

}

return predictFutureValue(currentValue \* (1 + growthRate), growthRate, years - 1);

}

}

**Main.java**

package com.example.forecast;

public class Main {

public static void main(String[] args) {

double initialValue = 1000.0;

double growthRate = 0.10;

int years = 5;

double futureValue = Forecast.predictFutureValue(initialValue, growthRate, years);

System.out.printf("Predicted future value after %d years: ₹%.2f\n", years, futureValue);

}

}

**Output:**

Predicted future value after 5 years: ₹1610.51

**4. Analysis**

**Time Complexity:**

The recursive method calls itself once per year:

Time Complexity = O(n)  
(where n is the number of years)

**Optimization:**

Recursion can be inefficient for large values. Optimization techniques include:

* Converting to iteration
* Using tail recursion
* Memoization (for problems with overlapping subproblems)

**Iterative Version:**

public static double predictIteratively(double value, double rate, int years) {

for (int i = 0; i < years; i++) {

value \*= (1 + rate);

}

return value;

}