**Super Set id** : 4991632

**Name** : Suggula Eshwar Durga Prasad

**SKILL : Design principles & Patterns**

Exercise 1: Implementing the Singleton Pattern

**Scenario:**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   * Create a class named Logger that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   * Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   * Create a test class to verify that only one instance of Logger is created and used across the application.

**Solution :**

**Logger.java**

package com.wtn.week1;

public class Logger {

private static Logger *w1*;

private Logger() {

}

public static Logger getInstance() {

if(*w1*==null) {

*w1*=new Logger();

}

return *w1*;

}

public void displayLog(String msg) {

System.***out***.println("LOG :"+msg);

}

}

**Test.java**

package com.wtn.week1;

public class Test {

public static void main(String[] args) {

// TODO Auto-generated method stub

Logger l1=Logger.getInstance();

l1.displayLog("first logger");

Logger l2=Logger.getInstance();

l2.displayLog("second logger");

if(l1==l2) {

System.out.println("Both instances are same");

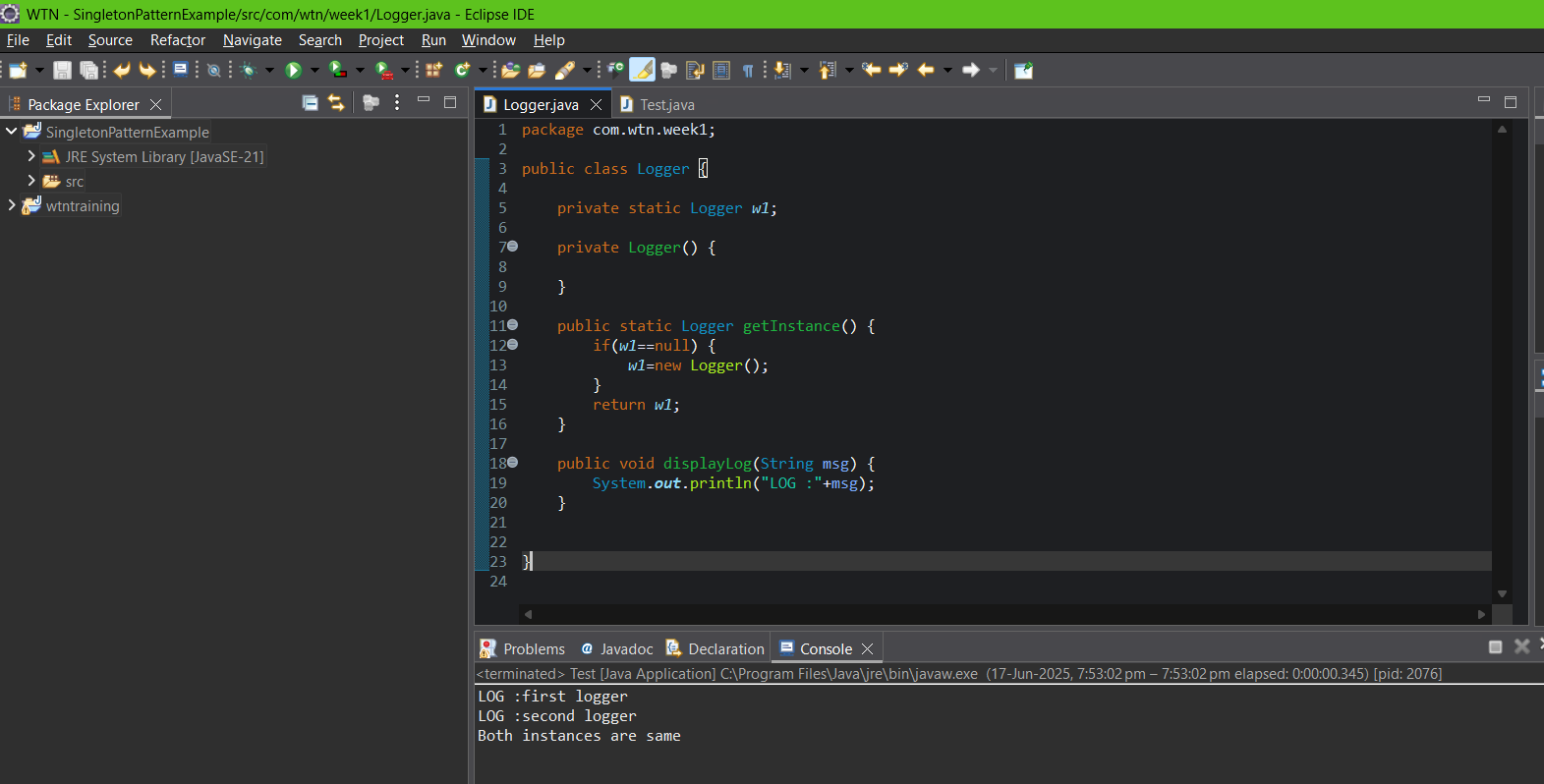
}else

System.out.println("Both are not same");

}

}

**Implementation :**



A computer screen shot of a program

AI-generated content may be incorrect.

**Output :**

A screenshot of a computer program

AI-generated content may be incorrect.

Exercise 2: Implementing the Factory Method Pattern

Scenario:

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

Steps:

1. Create a New Java Project:
   * Create a new Java project named FactoryMethodPatternExample.
2. Define Document Classes:
   * Create interfaces or abstract classes for different document types such as WordDocument, PdfDocument, and ExcelDocument.
3. Create Concrete Document Classes:
   * Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.
4. Implement the Factory Method:
   * Create an abstract class DocumentFactory with a method createDocument().
   * Create concrete factory classes for each document type that extends DocumentFactory and implements the createDocument() method.
5. Test the Factory Method Implementation:
   * Create a test class to demonstrate the creation of different document types using the factory method.

**Solution :**

**Document.java**

package com.week1.cts;

public interface Document {

void open();

}

**ExcelDocument.java**

package com.week1.cts;

public class ExcelDocument implements Document {

*@Override*

public void open() {

System.***out***.println("Excel Document");

}

}

**PdfDocument.java**

package com.week1.cts;

public class PdfDocument implements Document {

*@Override*

public void open() {

System.***out***.println("PDF Document");

}

}

**WordDocument.java**

package com.week1.cts;

public class WordDocument implements Document {

*@Override*

public void open() {

System.***out***.println(" Word Document");

}

}

**DocumentFactory.java**

package com.week1.cts;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**ExcelDocumentFactory .java**

package com.week1.cts;

public class ExcelDocumentFactory extends DocumentFactory {

*@Override*

public Document createDocument() {

return new ExcelDocument();

}

}

**PdfDocumentFactory.java**

package com.week1.cts;

public class PdfDocumentFactory extends DocumentFactory {

*@Override*

public Document createDocument() {

return new PdfDocument();

}

}

**WordDocumentFactory.java**

package com.week1.cts;

public class WordDocumentFactory extends DocumentFactory {

*@Override*

public Document createDocument() {

return new WordDocument();

}

}

**Test.java**

package com.week1.cts;

public class Test {

public static void main(String[] args) {

// **TODO** Auto-generated method stub

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

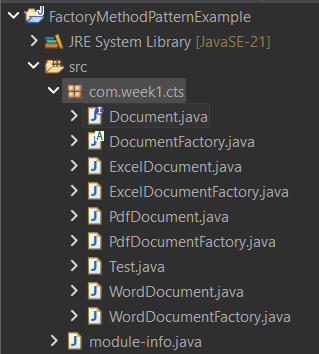
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Implementation :**

 **A screen shot of a computer

AI-generated content may be incorrect.** A screenshot of a computer program

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.** **A screenshot of a computer program

AI-generated content may be incorrect.** A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer program

AI-generated content may be incorrect.

**Output :**

**A screenshot of a computer

AI-generated content may be incorrect.**

**SKILL : Data structures and Algorithms**

Exercise 2: E-commerce Platform Search Function

Scenario:

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

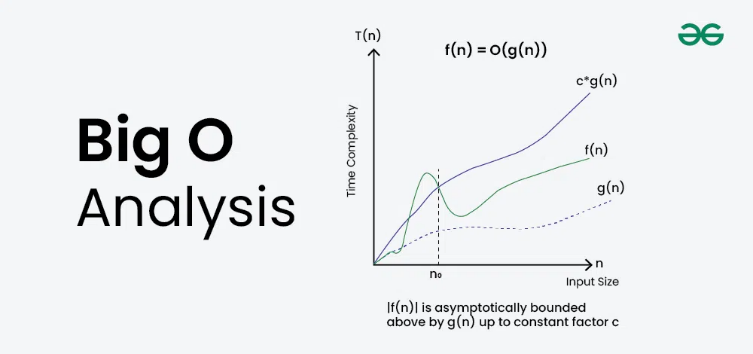
Steps:

1. Understand Asymptotic Notation:
   * Explain Big O notation and how it helps in analyzing algorithms.
   * Describe the best, average, and worst-case scenarios for search operations.
2. Setup:
   * Create a class Product with attributes for searching, such as productId, productName, and category.
3. Implementation:
   * Implement linear search and binary search algorithms.
   * Store products in an array for linear search and a sorted array for binary search.
4. Analysis:
   * Compare the time complexity of linear and binary search algorithms.
   * Discuss which algorithm is more suitable for your platform and why.

**Solution :**

1. Big O notation is used to describe the time complexity of algorithms in computer science; it is mostly used to express the upper bound of an algorithm’s time or space complexity.

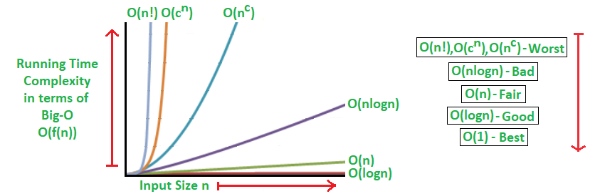
It is denoted as O(f(n)) where f(n) is a function that represents the number of operations an algorithm performs to solve the problem of size n.



It helps in analyzing an algorithm by providing the following details about it :

* + 1. It tells how fast or how much memory an algorithm is taking to complete the execution process, i.e., it measures efficiency.
    2. It ignores machine level factors like processor speed, programming language , hardware, constant time taken, so this makes it universal way to compare the algorithms
    3. It identifies the bottlenecks of an algorithm, i.e., it tells which part of the algorithm is inefficient when we trace the Big O through the algorithm.
    4. It is used to compare the algorithms, and it has a table of notations stating which takes more time and space and which is faster .

O(1) < O(log n) < O(n) < O(n log n) < O(n²) < O(n³) < O(2ⁿ) < O(n!)



**Best Case :**

The best-case scenario occurs when the element you’re searching for is found immediately, usually at the very beginning or in the most optimal position. For example, in linear search, the best case happens when the target is at the first index, making the search finish in constant time — O(1). It represents the most ideal and fastest outcome.

**Average Case:**

The average-case scenario assumes the element could be anywhere in the data structure with equal probability. So, on average, the algorithm might need to check around half the elements in linear search — resulting in O(n) time. This case gives a realistic view of expected performance under normal conditions.

**Worst Case:**

The worst-case scenario happens when the element is either not present at all or is found after scanning through all possible options. In linear search, this means checking every single element until the end — giving a time complexity of O(n). The worst case helps understand the maximum time an algorithm can take.

| **Search Type** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| Linear Search | O(1) | O(n) | O(n) |
| Binary Search | O(1) | O(log n) | O(log n) |
| Hash Table (ideal) | O(1) | O(1) | O(n)\* |
| BST (balanced) | O(log n) | O(log n) | O(n) |

**Code :**

**Product.java**

package com.week1.cts;

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;

}

*@Override*

public String toString() {

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

}

}

**LinearSearch.java**

package com.week1.cts;

public class LinearSearch {

public Product searchById(Product[] products, int targetId) {

for (Product p : products) {

if (p.productId == targetId) {

return p;

}

}

return null;

}

}

**BinarySearch.java**

package com.week1.cts;

import java.util.\*;

public class BinarySearch {

public Product searchById(Product[] products, int targetId) {

int l = 0, r = products.length - 1;

while (l <= r) {

int mid = (l + r) / 2;

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

return products[mid];

}

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

l = mid + 1;

}

else {

r = mid - 1;

}

}

return null;

}

public void sortProducts(Product[] products) {

Arrays.*sort*(products,Comparator.*comparingInt*(p-> p.productId));

}

}

**Test.java**

package com.week1.cts;

public class Test {

public static void main(String[] args) {

Product[] products = {

new Product(5, "Phone", "Electronics"),

new Product(9, "Shoes", "Fashion"),

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

new Product(3, "Book", "Stationery"),

};

LinearSearch l=new LinearSearch();

Product r1 = l.searchById(products, 9);

System.***out***.println("Linear Search Result: " + r1);

BinarySearch b = new BinarySearch();

b.sortProducts(products);

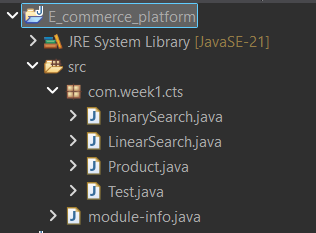
Product r2 = b.searchById(products, 9);

System.***out***.println("Binary Search Result: " + r2);

}

}

**Implementation :**



A screenshot of a computer program

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

**Output:**

A screenshot of a computer program

AI-generated content may be incorrect.\

**Analysis :**

Liner Search takes in **worst case : O(n)** and in **best case : O(1)** time complexity and it is best for small data or unsorted lists, and no sorting is required to perform linear search

Binary Search takes in **worst case : O(log n)** and **in best case : O(1)** time complexity and it is best for large , sorted data here sorting is required to perform the searching operation.

In our case of E-commerce platform search function, it is best to use Binary Search algorithm since its worst case is O(log n) which is lesser when compared to liner search worst case and it is suitable for large data searching to so we need to choose binary search algorithm here.

Exercise 7: Financial Forecasting

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**Steps:**

1. **Understand Recursive Algorithms:**
   * Explain the concept of recursion and how it can simplify certain problems.
2. **Setup:**
   * Create a method to calculate the future value using a recursive approach.
3. **Implementation:**
   * Implement a recursive algorithm to predict future values based on past growth rates.
4. **Analysis:**
   * Discuss the time complexity of your recursive algorithm.
   * Explain how to optimize the recursive solution to avoid excessive computation.

**Solution :**

Recursion is a technique where the function calls itself to solve a smaller part of a bigger problem, it continues to call itself with smaller inputs until it reaches a condition called the base case , which stops the recursion.

It helps to simplify the problems mainly those that can be broken down into smaller , repeating sub problems . It makes the code shorter , cleaner and easier to understand in many cases recursion allows us to solve problems with fewer lines of code and a clear structure.

Eg:

Calculation factorial of a number

Generating Fibonacci series

**Code :**

**FinancialForecast.java**

package com.week1.cts;

public class FinancialForecast {

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

if (years == 0) {

return currentValue;

}

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

}

public static void main(String[] args) {

double startAmount = 10000;

double growthRate = 0.10;

int years = 5;

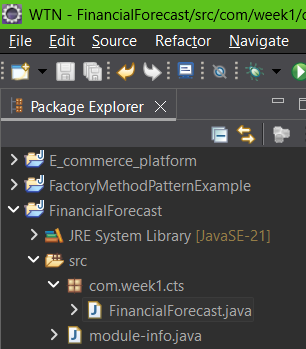
double futureValue = *calculateFutureValue*(startAmount, growthRate, years);

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

}

}

**Implementation :**



A screenshot of a computer program

AI-generated content may be incorrect.

**Output :**

A screen shot of a computer

AI-generated content may be incorrect.

**Analysis :**

Here the method calls itself for every year until it reaches the base case y==0 so , for n years the function makes n recursive calls.

Time complexity : O(n) where n is the number of years

**Optimizing techniques :**

* + We need to use iterations instead of recursion, this will make it easier and more efficient algorithm.
  + We can also use mathematical formula i.e., amount\*Math.pow(1+rate,years) directly instead of using recursion this will make the algorithm to execute in the time complexity of just O(1) which us very efficient and optimized .
  + For recursion it is O(n) it is simple but not optimal
  + For iterative it is O(n) but better memory usage
  + For mathematical approach it is O(1) it is faster and most efficient