Java Design Patterns and Algorithms - Comprehensive Document

# 1. Implementing the Singleton Pattern

The Singleton Pattern ensures that a class has only one instance throughout the application's lifecycle and provides a global access point to it. This is particularly useful for classes like loggers, which should have only one instance to maintain consistency in logging.

## Steps:

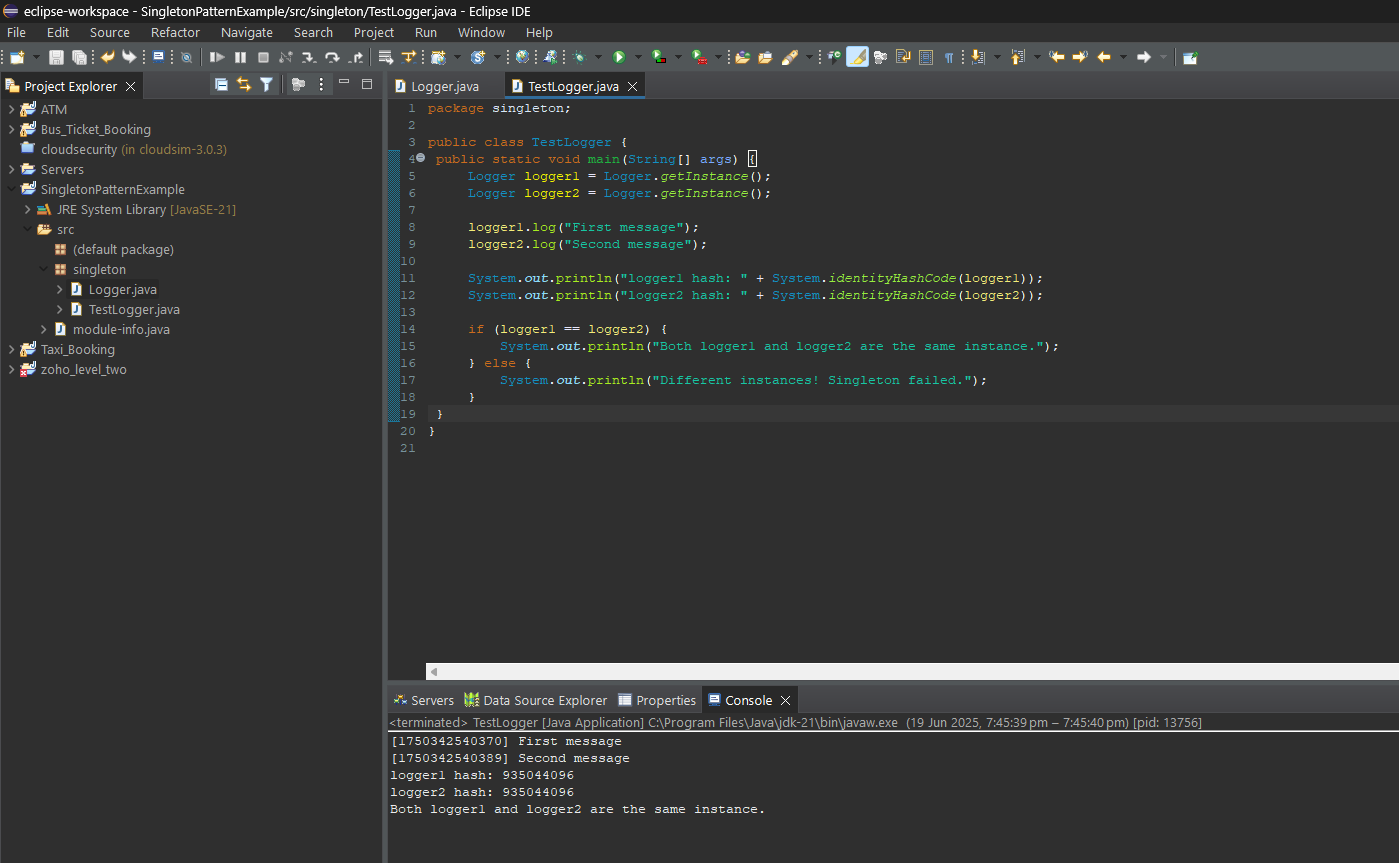
1. Create a new Java project named SingletonPatternExample.  
2. Define a Singleton class named Logger with a private static instance of itself.  
3. Ensure the constructor of Logger is private.  
4. Provide a public static method to return the instance.  
5. Test the singleton implementation by checking if two instances are the same.

## Java Code:

// 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;  
 }  
  
 public void log(String message) {  
 System.out.println("Log: " + message);  
 }  
}  
  
// Main.java  
public class Main {  
 public static void main(String[] args) {  
 Logger logger1 = Logger.getInstance();  
 logger1.log("First log message.");  
  
 Logger logger2 = Logger.getInstance();  
 logger2.log("Second log message.");  
  
 if (logger1 == logger2) {  
 System.out.println("Both logger instances are the same.");  
 }  
 }  
}

## Output:

Logger instance created.  
Log: First log message.  
Log: Second log message.  
Both logger instances are the same.



## Explanation:

The getInstance() method ensures that only one object is created. Even though logger1 and logger2 are created separately, they both point to the same object. This ensures consistent logging throughout the application.

# 2. Implementing the Factory Method Pattern

The Factory Method Pattern helps in creating objects without specifying the exact class of the object that will be created. It provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created.

## Steps:

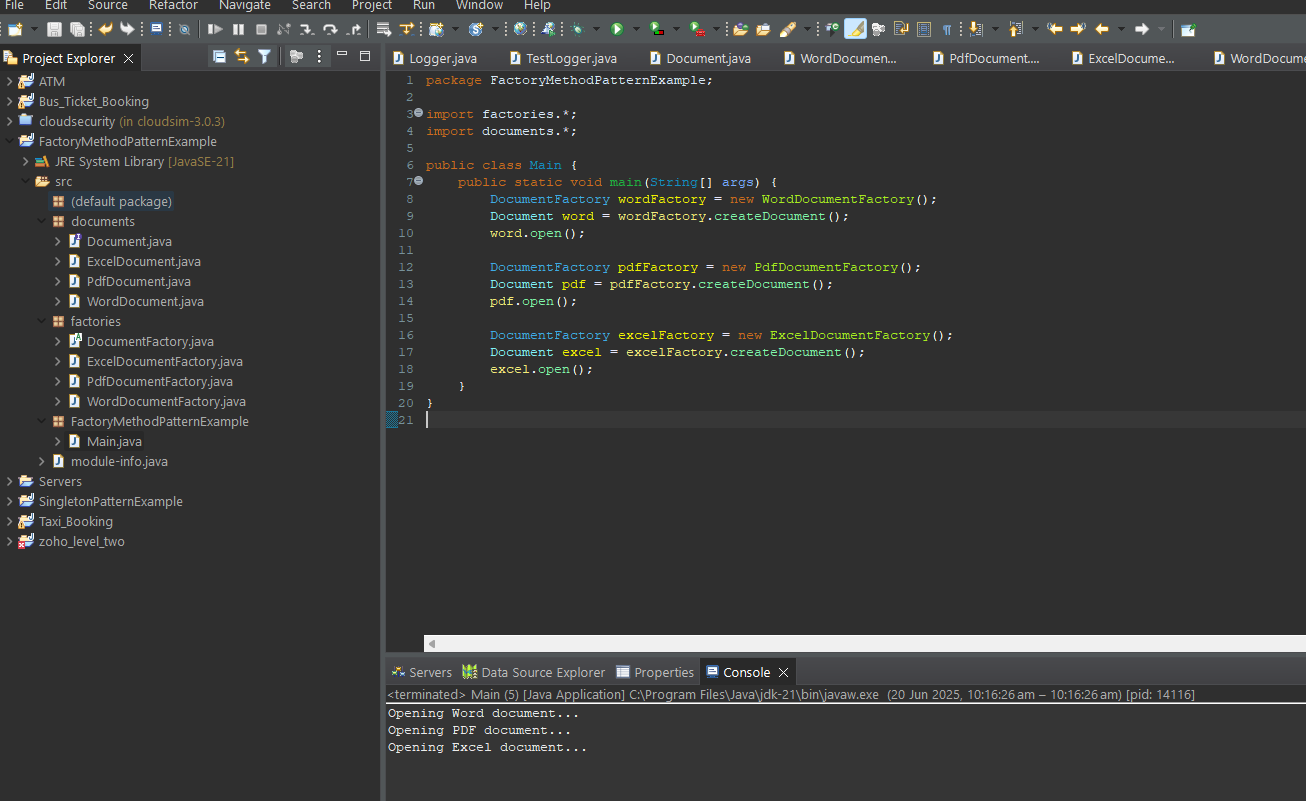
1. Create a new Java project named FactoryMethodPatternExample.  
2. Define an interface or abstract class named Document.  
3. Create concrete classes WordDocument, PdfDocument, ExcelDocument implementing Document.  
4. Define an abstract class DocumentFactory with a createDocument() method.  
5. Create concrete factories for each document type.  
6. Use the factories in the main method to create and open different documents.

## Java Code:

// Document.java  
public interface Document {  
 void open();  
}  
  
// WordDocument.java, PdfDocument.java, ExcelDocument.java  
public class WordDocument implements Document {  
 public void open() {  
 System.out.println("Opening Word document...");  
 }  
}  
public class PdfDocument implements Document {  
 public void open() {  
 System.out.println("Opening PDF document...");  
 }  
}  
public class ExcelDocument implements Document {  
 public void open() {  
 System.out.println("Opening Excel document...");  
 }  
}  
  
// DocumentFactory.java and subclasses  
public abstract class DocumentFactory {  
 public abstract Document createDocument();  
}  
public class WordDocumentFactory extends DocumentFactory {  
 public Document createDocument() {  
 return new WordDocument();  
 }  
}  
public class PdfDocumentFactory extends DocumentFactory {  
 public Document createDocument() {  
 return new PdfDocument();  
 }  
}  
public class ExcelDocumentFactory extends DocumentFactory {  
 public Document createDocument() {  
 return new ExcelDocument();  
 }  
}  
  
// Main.java  
public class Main {  
 public static void main(String[] args) {  
 DocumentFactory factory;  
  
 factory = new WordDocumentFactory();  
 factory.createDocument().open();  
  
 factory = new PdfDocumentFactory();  
 factory.createDocument().open();  
  
 factory = new ExcelDocumentFactory();  
 factory.createDocument().open();  
 }  
}

## Output:

Opening Word document...  
Opening PDF document...  
Opening Excel document...



## Explanation:

Each factory is responsible for creating its own type of document. The client code uses the factory without knowing the exact class name, thus adhering to the Factory Method Pattern.

# 3. E-commerce Platform Search Function

To implement search functionality in an e-commerce platform, we must consider performance. Two commonly used search methods are linear search and binary search.

## Steps:

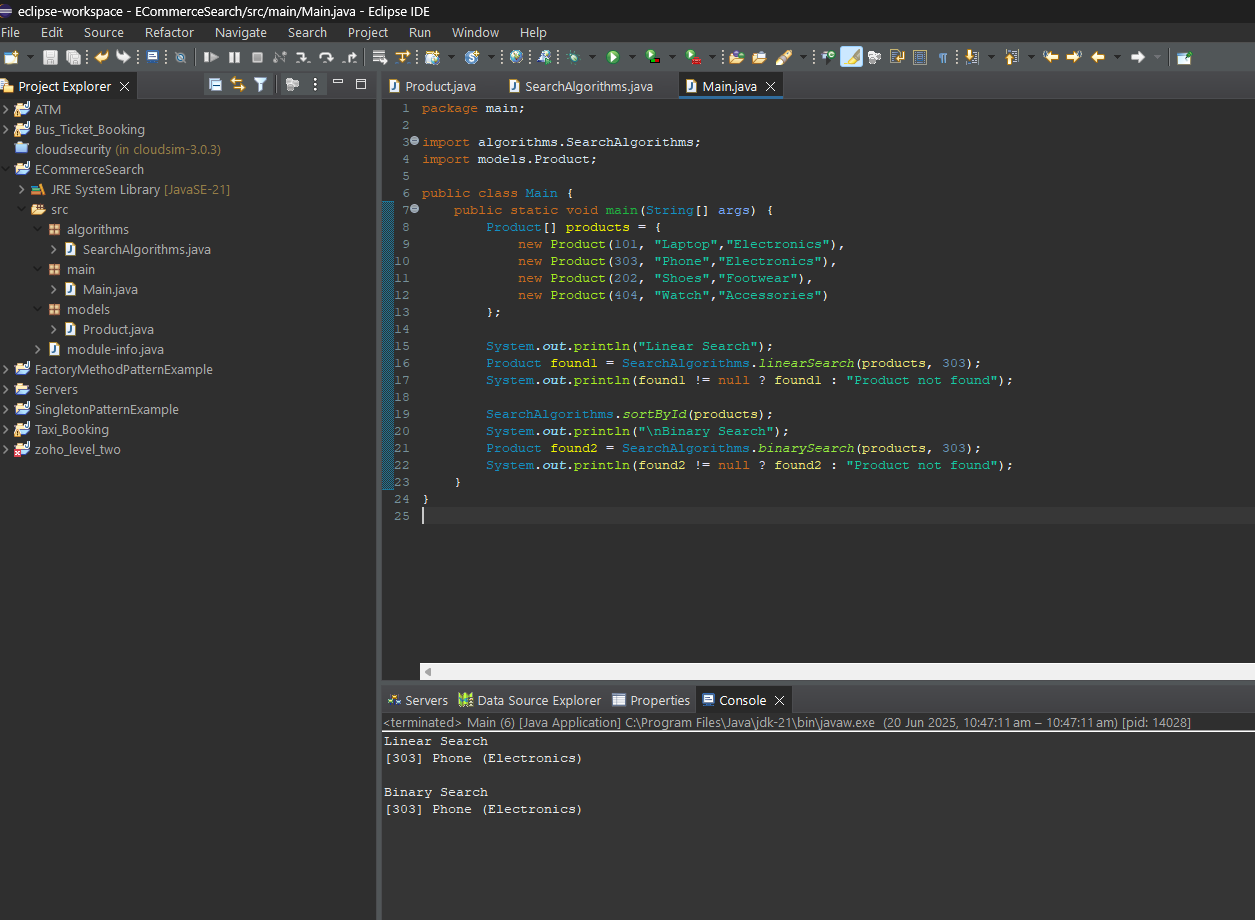
1. Understand Big O notation for analyzing algorithms.  
2. Create a Product class with relevant attributes.  
3. Implement linear and binary search functions.  
4. Analyze time complexity and determine the best approach.

## Java Code:

// Product.java  
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 + ")";  
 }  
}  
  
// Search.java  
public class Search {  
 public static Product linearSearch(Product[] products, String targetName) {  
 for (Product product : products) {  
 if (product.productName.equalsIgnoreCase(targetName)) {  
 return product;  
 }  
 }  
 return null;  
 }  
  
 public static Product binarySearch(Product[] products, String targetName) {  
 Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));  
 int left = 0, right = products.length - 1;  
 while (left <= right) {  
 int mid = (left + right) / 2;  
 int comp = products[mid].productName.compareToIgnoreCase(targetName);  
 if (comp == 0) return products[mid];  
 else if (comp < 0) left = mid + 1;  
 else right = mid - 1;  
 }  
 return null;  
 }  
}  
  
// Main.java  
public class Main {  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(101, "Laptop", "Electronics"),  
 new Product(102, "Shirt", "Clothing"),  
 new Product(103, "Book", "Education"),  
 new Product(104, "Headphones", "Electronics")  
 };  
  
 Product foundLinear = Search.linearSearch(products, "Book");  
 System.out.println("Linear Search Result: " + (foundLinear != null ? foundLinear : "Not Found"));  
  
 Product foundBinary = Search.binarySearch(products, "Book");  
 System.out.println("Binary Search Result: " + (foundBinary != null ? foundBinary : "Not Found"));  
 }  
}

## Output:

Linear Search Result: [103] Book (Education)  
Binary Search Result: [103] Book (Education)



## Explanation:

Linear search checks each element, making it O(n). Binary search, which requires sorted data, reduces time complexity to O(log n). For large datasets, binary search is faster and preferable if the data can be sorted.

# 4. Financial Forecasting Using Recursion

Recursion allows a function to call itself to solve a problem by breaking it into smaller sub-problems. This can be applied to predict financial growth by repeatedly applying the growth rate.

## Steps:

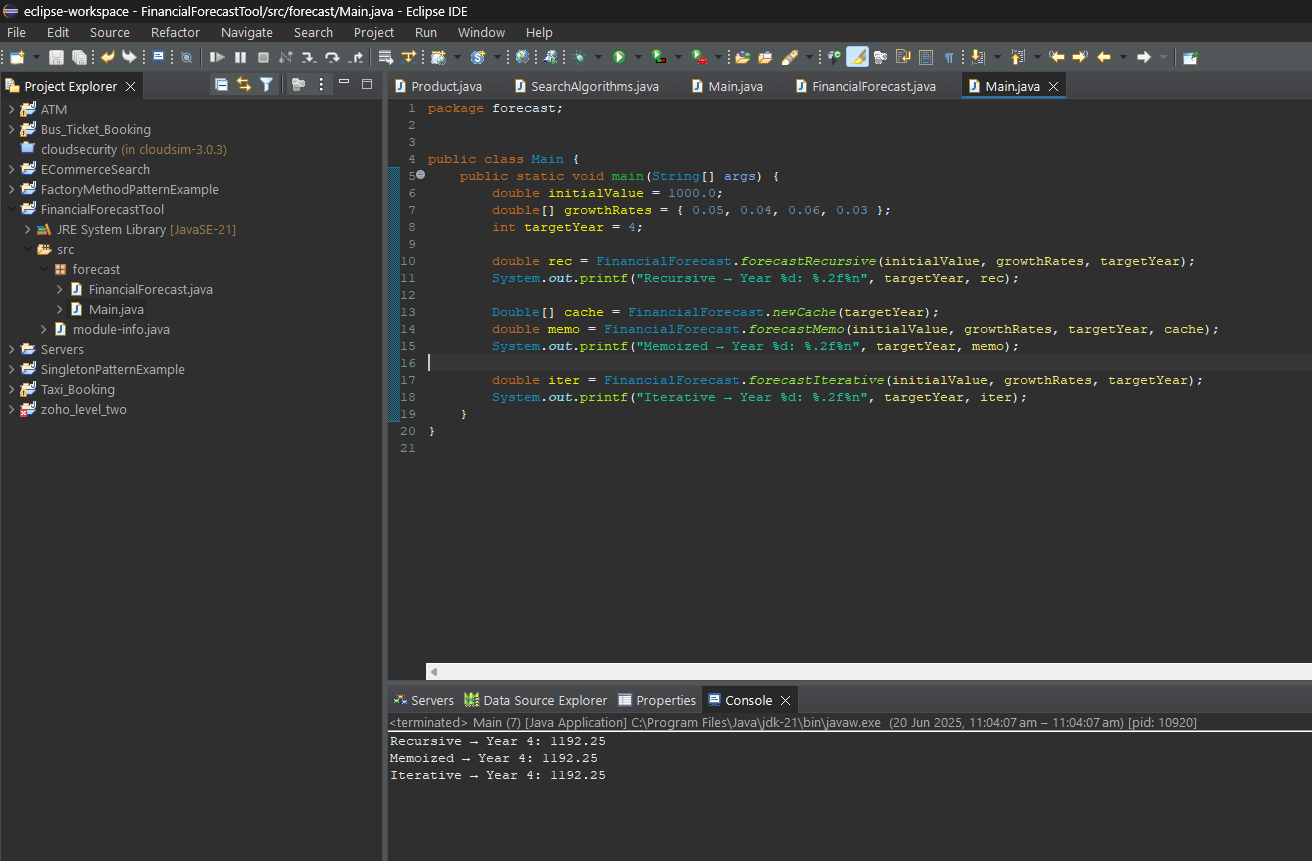
1. Understand how recursion simplifies repetitive calculations.  
2. Create a method to calculate future value using recursion.  
3. Implement the algorithm to calculate future value from present value and growth rate.  
4. Analyze and optimize the recursive algorithm.

## Java Code:

// FinancialForecast.java  
public class FinancialForecast {  
  
 public static double calculateFutureValue(double presentValue, double growthRate, int years) {  
 if (years == 0) {  
 return presentValue;  
 }  
 return calculateFutureValue(presentValue, growthRate, years - 1) \* (1 + growthRate);  
 }  
  
 public static void main(String[] args) {  
 double presentValue = 10000.0;  
 double growthRate = 0.05;  
 int years = 5;  
  
 double futureValue = calculateFutureValue(presentValue, growthRate, years);  
 System.out.printf("Predicted future value after %d years: %.2f\n", years, futureValue);  
 }  
}

## Output:

Predicted future value after 5 years: 12762.82



## Explanation:

The recursive method multiplies the present value by (1 + growth rate) every year. While recursion makes the logic clear, it can be inefficient for large inputs. Using memoization or iterative methods can optimize performance.