**WEEK 1**

**Design Patterns, Data Structures & Algorithms**

1. **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.

**Logger.java: -**

**package** week1;

**public** **class** Logger {

**private** **static** Logger *instance*;

**private** Logger() {

System.***out***.println("New 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: -**

**package** week1;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Logger logger1 = Logger.*getInstance*();

logger1.log("This is the first log message.");

Logger logger2 = Logger.*getInstance*();

logger2.log("This is the second log message.");

**if** (logger1 == logger2) {

System.***out***.println("Both logger instances are the same. Singleton works!");

} **else** {

System.***out***.println("Different instances. Singleton failed!");

}

}

}

**Output: -**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **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.

**Document.java: -**

**package** factoryMethod;

**public** **interface** Document {

**void** open();

}

**DocumentFactory.java: -**

**package** factoryMethod;

**public** **abstract** **class** DocumentFactory {

**public** **abstract** Document createDocument();

}

**ExcelDocument.java: -**

**package** factoryMethod;

**public** **class** ExcelDocument **implements** Document {

**public** **void** open() {

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

}

}

**ExcelDocumentFactory.java: -**

**package** factoryMethod;

**public** **class** ExcelDocumentFactory **extends** DocumentFactory {

**public** Document createDocument() {

**return** **new** ExcelDocument();

}

}

**PdfDocument.java: -**

**package** factoryMethod;

**public** **class** PdfDocument **implements** Document {

**public** **void** open() {

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

}

}

**PdfDocumentFactory.java: -**

**package** factoryMethod;

**public** **class** PdfDocumentFactory **extends** DocumentFactory {

**public** Document createDocument() {

**return** **new** PdfDocument();

}

}

**WordDocument.java: -**

**package** factoryMethod;

**public** **class** WordDocument **implements** Document {

**public** **void** open() {

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

}

}

**WordDocumentFactory.java: -**

**package** factoryMethod;

**public** **class** WordDocumentFactory **extends** DocumentFactory {

**public** Document createDocument() {

**return** **new** WordDocument();

}

}

**Main.java: -**

**package** factoryMethod;

**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: -**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **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.

**Product.java: -**

**package** ecommerceSearch;

**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;

}

}

**ProductSearch.java: -**

**package** ecommerceSearch;

**import** java.util.Arrays;

**import** java.util.Comparator;

**public** **class** ProductSearch {

**public** **static** Product linearSearch(Product[] products, **int** targetId) {

**for** (Product product : products) {

**if** (product.productId == targetId) {

**return** product;

}

}

**return** **null**;

}

**public** **static** Product binarySearch(Product[] products, **int** targetId) {

**int** low = 0;

**int** high = products.length - 1;

**while** (low <= high) {

**int** mid = (low + high) / 2;

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

**return** products[mid];

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

low = mid + 1;

} **else** {

high = mid - 1;

}

}

**return** **null**;

}

**public** **static** **void** sortProductsById(Product[] products) {

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

}

}

**Main.java: -**

**package** ecommerceSearch;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Product[] products = {

**new** Product(103, "Shoes", "Footwear"),

**new** Product(101, "Laptop", "Electronics"),

**new** Product(105, "Keyboard", "Accessories"),

**new** Product(102, "T-shirt", "Apparel"),

**new** Product(104, "Phone", "Electronics")

};

System.***out***.println("🔍 Linear Search:");

Product result1 = ProductSearch.*linearSearch*(products, 105);

System.***out***.println(result1 != **null** ? result1 : "Product not found");

System.***out***.println("\n🔍 Binary Search:");

ProductSearch.*sortProductsById*(products); // Ensure sorted

Product result2 = ProductSearch.*binarySearch*(products, 105);

System.***out***.println(result2 != **null** ? result2 : "Product not found");

}

}

**Output: -**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Financial Forecasting**

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

**Forecast.java: -**

**package** financialForecast;

**import** java.util.Scanner;

**public** **class** Forecast {

**public** **static** **double** predictFutureValue(**double** initialValue, **double** growthRate, **int** years) {

**if** (years == 0) {

**return** initialValue;

}

**double** previousValue = *predictFutureValue*(initialValue, growthRate, years - 1);

**return** previousValue \* (1 + growthRate / 100);

}

**public** **static** **void** main(String[] args) {

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

System.***out***.print("Enter Initial Investment: ");

**double** initialInvestment = sc.nextDouble();

System.***out***.print("Enter Annual Growth: ");

**double** annualGrowthRate = sc.nextDouble();

System.***out***.print("Enter Forecast Years: ");

**int** forecastYears = sc.nextInt();

**double** futureValue = *predictFutureValue*(initialInvestment, annualGrowthRate, forecastYears);

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

}

}

**Output: -** **A screenshot of a computer

AI-generated content may be incorrect.**