**Design Patterns and Principles:**

**Ex.1-SingletonPattern:**

public class Main {  
 public static void main(String[] args) {  
  
 Logger logger1 = Logger.*getInstance*();  
 Logger logger2 = Logger.*getInstance*();  
  
  
 logger1.log("This is the first log message.");  
 logger2.log("This is the second log message.");  
  
  
 if (logger1 == logger2) {  
 System.*out*.println("Both logger1 and logger2 are the same instance.");  
 } else {  
 System.*out*.println("Different logger instances (Something's wrong!)");  
 }  
 }  
}

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);  
 }  
**Output: A screen shot of a computer

AI-generated content may be incorrect.**

**Ex.2-FactoryMethodPattern:**

public interface Document {  
 void open();  
}

public class WordDocument implements Document {  
 @Override  
 public void open() {  
 System.*out*.println("Opening Word document.");  
 }  
}

public class PdfDocument implements Document {  
 @Override  
 public void open() {  
 System.*out*.println("Opening PDF document.");  
 }  
}

public class ExcelDocument implements Document {  
 @Override  
 public void open() {  
 System.*out*.println("Opening Excel document.");  
 }  
}

public abstract class DocumentFactory {  
 public abstract Document createDocument();  
}

public class WordDocumentFactory extends DocumentFactory {  
 @Override  
 public Document createDocument() {  
 return new WordDocument();  
 }  
}

public class PdfDocumentFactory extends DocumentFactory {  
 @Override  
 public Document createDocument() {  
 return new PdfDocument();  
 }  
}

public class ExcelDocumentFactory extends DocumentFactory {  
 @Override  
 public Document createDocument() {  
 return new ExcelDocument();  
 }  
}

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 screen shot of a computer

AI-generated content may be incorrect.**

**Data Structures and Algorithm:**

**Ex.2-E-commerce Platform Search Function:**

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 + ")";  
 }  
}

public class SearchUtil {  
 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) {  
 int left = 0, right = products.length - 1;  
 while (left <= right) {  
 int mid = (left + right) / 2;  
 int cmp = products[mid].productName.compareToIgnoreCase(targetName);  
 if (cmp == 0) {  
 return products[mid];  
 } else if (cmp < 0) {  
 left = mid + 1;  
 } else {  
 right = mid - 1;  
 }  
 }  
 return null;  
 }  
}  
import java.util.Arrays;  
import java.util.Comparator;  
  
public class Main {  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(101, "Laptop", "Electronics"),  
 new Product(102, "Shampoo", "Personal Care"),  
 new Product(103, "Keyboard", "Electronics"),  
 new Product(104, "Notebook", "Stationery")  
 };  
  
  
 System.*out*.println("Linear Search Result:");  
 Product found1 = SearchUtil.*linearSearch*(products, "Shampoo");  
 System.*out*.println(found1 != null ? found1 : "Product not found");  
  
  
 Arrays.*sort*(products, Comparator.*comparing*(p -> p.productName.toLowerCase()));  
  
  
 System.*out*.println("\nBinary Search Result:");  
 Product found2 = SearchUtil.*binarySearch*(products, "Shampoo");  
 System.*out*.println(found2 != null ? found2 : "Product not found");  
 }  
}

**Output:**

**A black screen with white text

AI-generated content may be incorrect.**

**Ex.7-Financial Forecasting:**

public class Forecast {  
 public static double forecastFutureValue(double presentValue, double growthRate, int years) {  
 if (years == 0) {  
 return presentValue;  
 } else {  
 return *forecastFutureValue*(presentValue \* (1 + growthRate), growthRate, years - 1);  
 }  
 }  
 public static double forecastIterative(double presentValue, double growthRate, int years) {  
 for (int i = 0; i < years; i++) {  
 presentValue \*= (1 + growthRate);  
 }  
 return presentValue;  
 }  
 public static void main(String[] args) {  
 double presentValue = 10000.0;  
 double growthRate = 0.05;  
 int years = 5;  
  
 double futureRecursive = *forecastFutureValue*(presentValue, growthRate, years);  
 double futureIterative = *forecastIterative*(presentValue, growthRate, years);  
  
 System.*out*.printf("Recursive Forecast after %d years: ₹%.2f\n", years, futureRecursive);  
 System.*out*.printf("Iterative Forecast after %d years: ₹%.2f\n", years, futureIterative);  
 }  
}

**Output:**

**A black screen with many small colored lines

AI-generated content may be incorrect.**