**Design Patterns**

**Exercise 1: Implementing the Singleton Pattern**

// Logger.java

**package** SingletonPatternExample;

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

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

**private** Logger() {

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

}

**public** **static** Logger getInstance() {

**if**(*instance*==**null**) {

*instance*=**new** Logger();

}

**return** *instance*;

}

}

// Main.java

**package** SingletonPatternExample;

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

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

Logger logger=Logger.*getInstance*();

Logger logger2 = Logger.*getInstance*();

**if** (logger == logger2) {

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

} **else** {

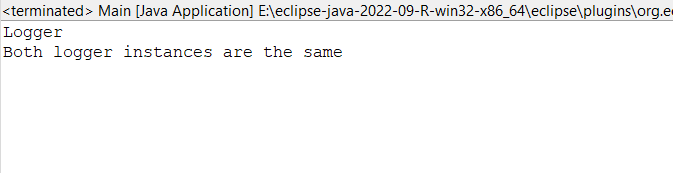
System.***out***.println("Different instances");

}

}

}

**Output:**



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

// Document.java

**package** factory;

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

**void** open();

}

//WordDocument.java

**package** factory;

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

@Override

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

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

}

}

//ExcelDocument.java

**package** factory;

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

@Override

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

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

}

}

//DocumentFactory.java

**package** factory;

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

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

}

//WordDocumentFactory.java

**package** factory;

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

@Override

**public** Document createDocument() {

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

}

}

//PdfDocumentFactory.java

**package** factory;

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

@Override

**public** Document createDocument() {

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

}

}

//ExcelDocumentFactory.java

**package** factory;

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

@Override

**public** Document createDocument() {

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

}

}

//Main.java

**package** factory;

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

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

DocumentFactory wordFactory = **new** WordDocumentFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = **new** PdfDocumentFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

DocumentFactory excelFactory = **new** ExcelDocumentFactory();

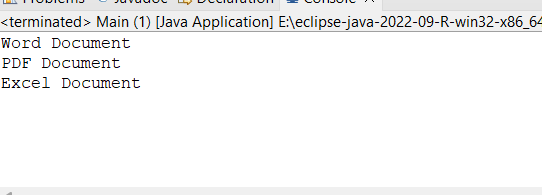
Document excel = excelFactory.createDocument();

excel.open();

}

}

**Output:**



**Data Structures**

**Exercise 2: E-commerce Platform Search Function**

//Product.java

**package** ecommerce;

**public** **class** Product {

**private** **int** productId;

**private** String productName;

**private** String category;

**public** Product(**int** productId, String productName, String category) {

**this**.productId = productId;

**this**.productName = productName;

**this**.category = category;

}

**public** **int** getProductId() {

**return** productId;

}

**public** String getProductName() {

**return** productName;

}

**public** String getCategory() {

**return** category;

}

@Override

**public** String toString() {

**return** "[" +productId+ "] " +productName+ " - " + category;

}

}

//SearchUtility.java

**package** ecommerce;

**public** **class** SearchUtility {

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

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

**if** (product.getProductName().equalsIgnoreCase(productName)) {

**return** product;

}

}

**return** **null**;

}

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

**int** left = 0;

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

**while** (left <= right) {

**int** mid = left + (right - left) / 2;

**int** comparison = products[mid].getProductName().compareToIgnoreCase(productName);

**if** (comparison == 0) {

**return** products[mid];

} **else** **if** (comparison < 0) {

left = mid + 1;

} **else** {

right = mid - 1;

}

}

**return** **null**;

}

}

//Main.java

**package** ecommerce;

**import** java.util.Arrays;

**import** java.util.Comparator;

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

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

Product[] products = {

**new** Product(111, "Laptop", "Electronics"),

**new** Product(112, "EarBuds", "Electronics"),

**new** Product(113, "Smartphone", "Electronics"),

**new** Product(114, "Earrings", "Accessories"),

**new** Product(115, "Ring", "Accessories")

};

Product linearResult = SearchUtility.*linearSearch*(products, "Ring");

System.***out***.println("Linear Search Result: " + (linearResult != **null** ? linearResult : "Not Found"));

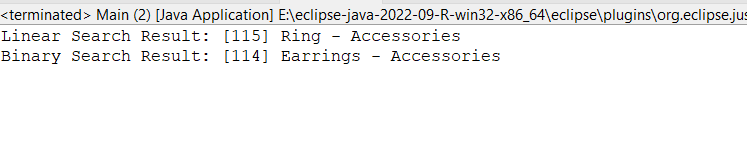
Arrays.*sort*(products, Comparator.*comparing*(Product::getProductName));

Product binaryResult = SearchUtility.*binarySearch*(products, "Earrings");

System.***out***.println("Binary Search Result: " + (binaryResult != **null** ? binaryResult : "Not Found"));

}

}

**Output**

**Exercise 7: Financial Forecasting**

//ForeCast.java

**package** finance;

**public** **class** ForeCast {

**public** **static** **double** forecastValue(**double** currentValue, **double** growthRate,**int** years) {

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

**return** currentValue;

}

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

}

}

//Main.java

**package** finance;

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

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

**double** currentValue = 10000;

**double** growthRate = 0.04;

**int** years = 5;

**double** futureValue = ForeCast.*forecastValue*(currentValue, growthRate, years);

System.***out***.println("Future value after "+years+" years: "+futureValue);

}

}

**Output:**

