**WEEK-1**

(Implemented in Eclipse IDE )

**1)Design Principles And Patterns**

Exercise 1: Implementing the Singleton Pattern:

**Logger.java**

**package** com.singleton.logger;

**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);

}

}  
  
  
**LoggerTest.java**

**package** com.singleton.logger;

**public** **class** LoggerTest {

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

Logger logger1 = Logger.*getInstance*();

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

Logger logger2 = Logger.*getInstance*();

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

// Check if both loggers are same instance

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

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

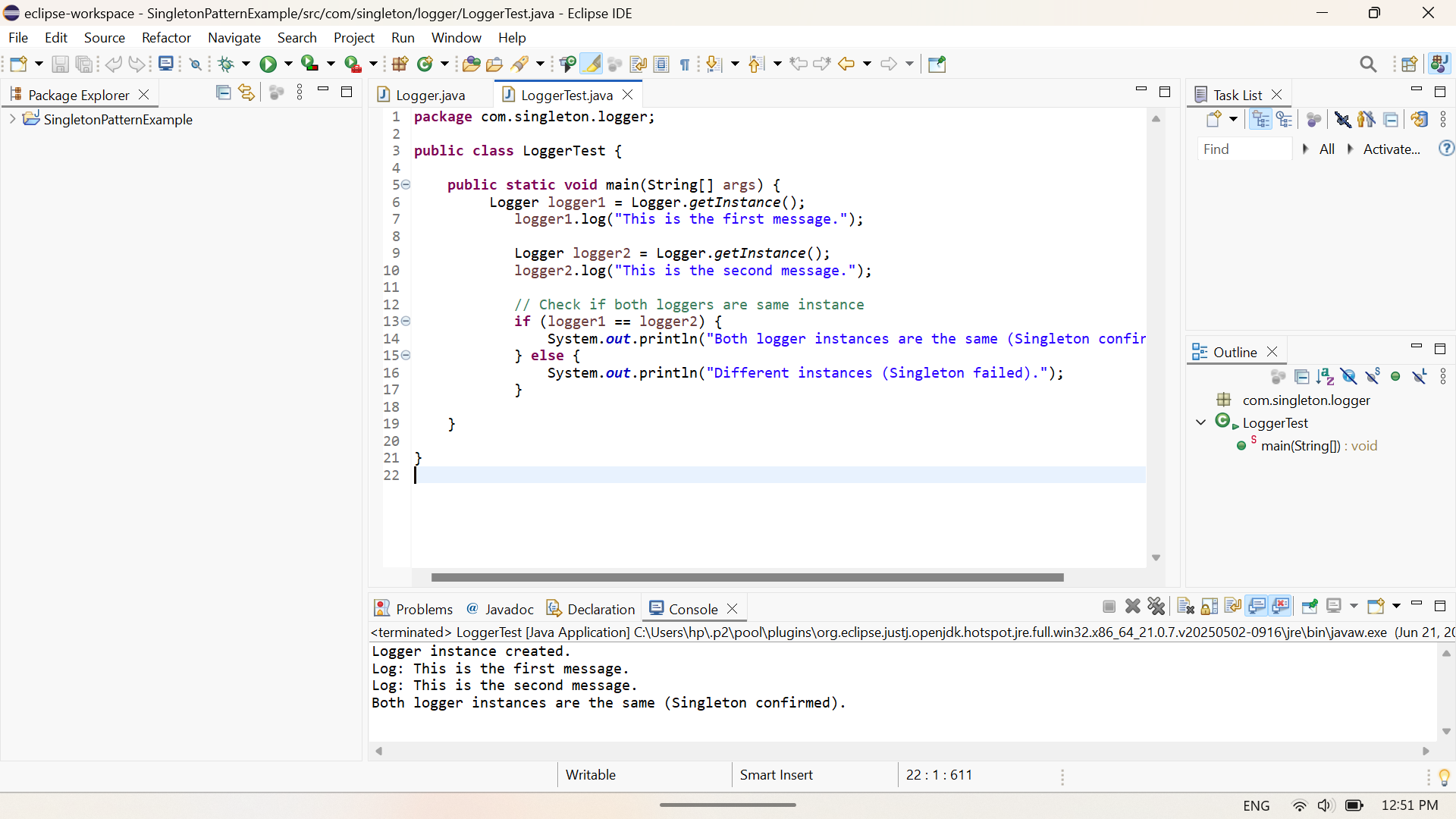
} **else** {

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

}

}

**Output:**

****

Exercise 2: Implementing the Factory Method Pattern:

**Document.java**

**package** com.factory.document;

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

**void** open();

}

**WordDocument.java**

**package** com.factory.document;

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

@Override

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

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

}

}

**PdfDocument.java**

**package** com.factory.document;

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

@Override

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

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

}

}

**ExcelDocument.java**

**package** com.factory.document;

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

@Override

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

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

}

}

**DocumentFactory.java**

**package** com.factory.document;

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

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

}

**WordDocumentFactory.java**

**package** com.factory.document;

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

@Override

**public** Document createDocument() {

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

}

}

**PdfDocumentFactory.java**

**package** com.factory.document;

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

@Override

**public** Document createDocument() {

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

}

}

**ExcelDocumentFactory.java**

**package** com.factory.document;

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

@Override

**public** Document createDocument() {

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

}

}

**DocumentFactoryTest.java**

**package** com.factory.document;

**public** **class** DocumentFactoryTest {

**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();

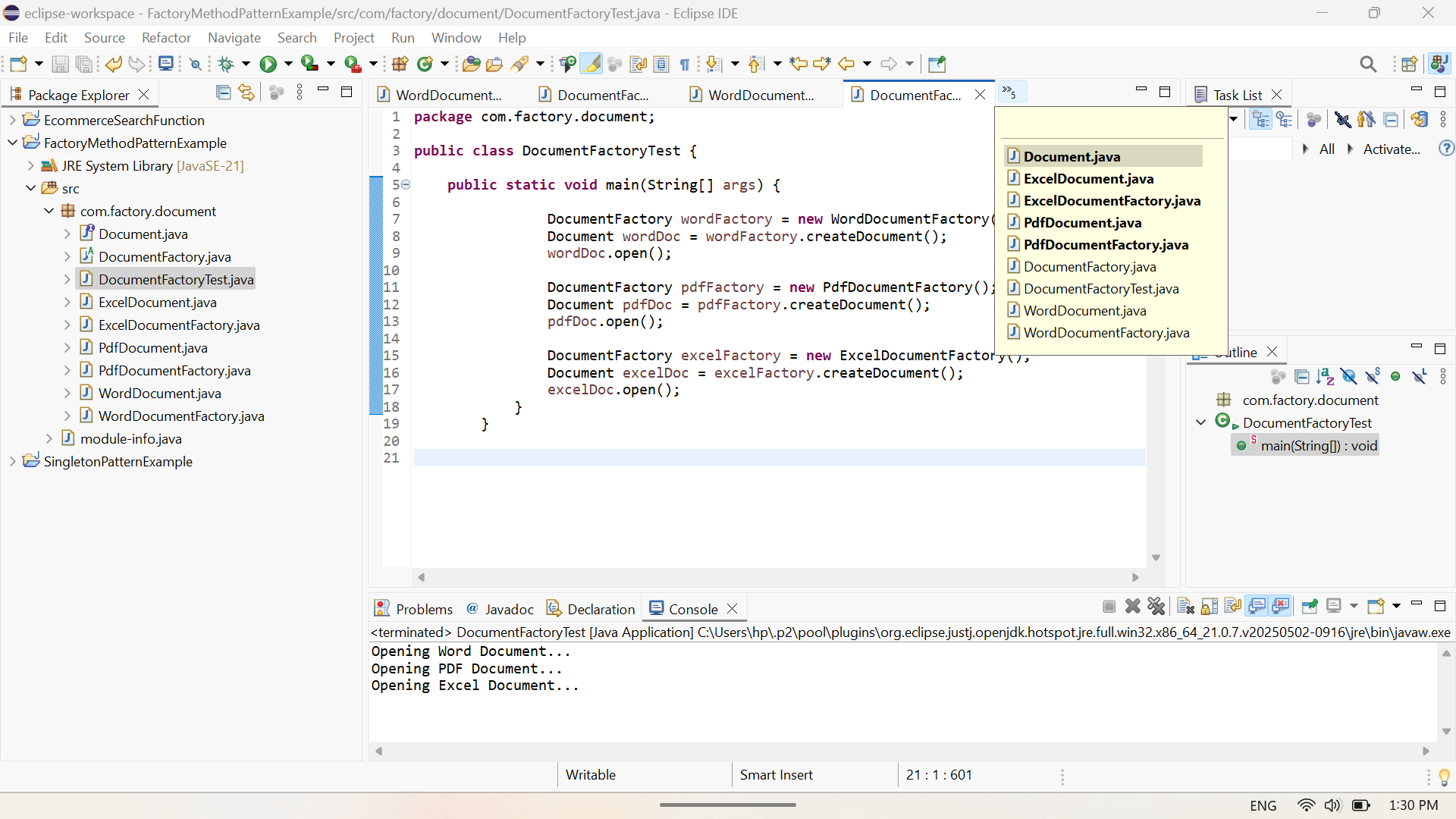
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Output:**

****

**2)Data structures and Algorithms**

Exercise 2: E-Commerce Platform Search Function:

**Product.java**

**package** com.ecommerce.search;

**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 + "}";

}

}

**SearchDemo.java**

**package** com.ecommerce.search;

**import** java.util.\*;

**public** **class** SearchDemo {

// Linear Search

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

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

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

**return** product;

}

}

**return** **null**;

}

// Binary Search (requires sorted array)

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

**int** left = 0, right = products.length - 1;

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

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

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

**return** products[mid];

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

left = mid + 1;

} **else** {

right = mid - 1;

}

}

**return** **null**;}

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

Product[] products = {

**new** Product(101, "Dress", "Fashion"),

**new** Product(102, "Jersey", "Sports"),

**new** Product(103, "Earrings", "Accessories"),

**new** Product(104, "Pen", "Stationery"),

**new** Product(105, "IPhone", "Electronics")

};

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

Product result1 = *linearSearch*(products, 104);

System.***out***.println(result1);

// Sorting for binary search

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

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

Product result2 = *binarySearch*(products, 104);

System.***out***.println(result2);

// Explanation of algorithms

System.***out***.println("🔎 Comparing Linear and Binary Search for E-commerce Platform");

System.***out***.println("Linear Search: Simple, works on unsorted data, O(n) time");

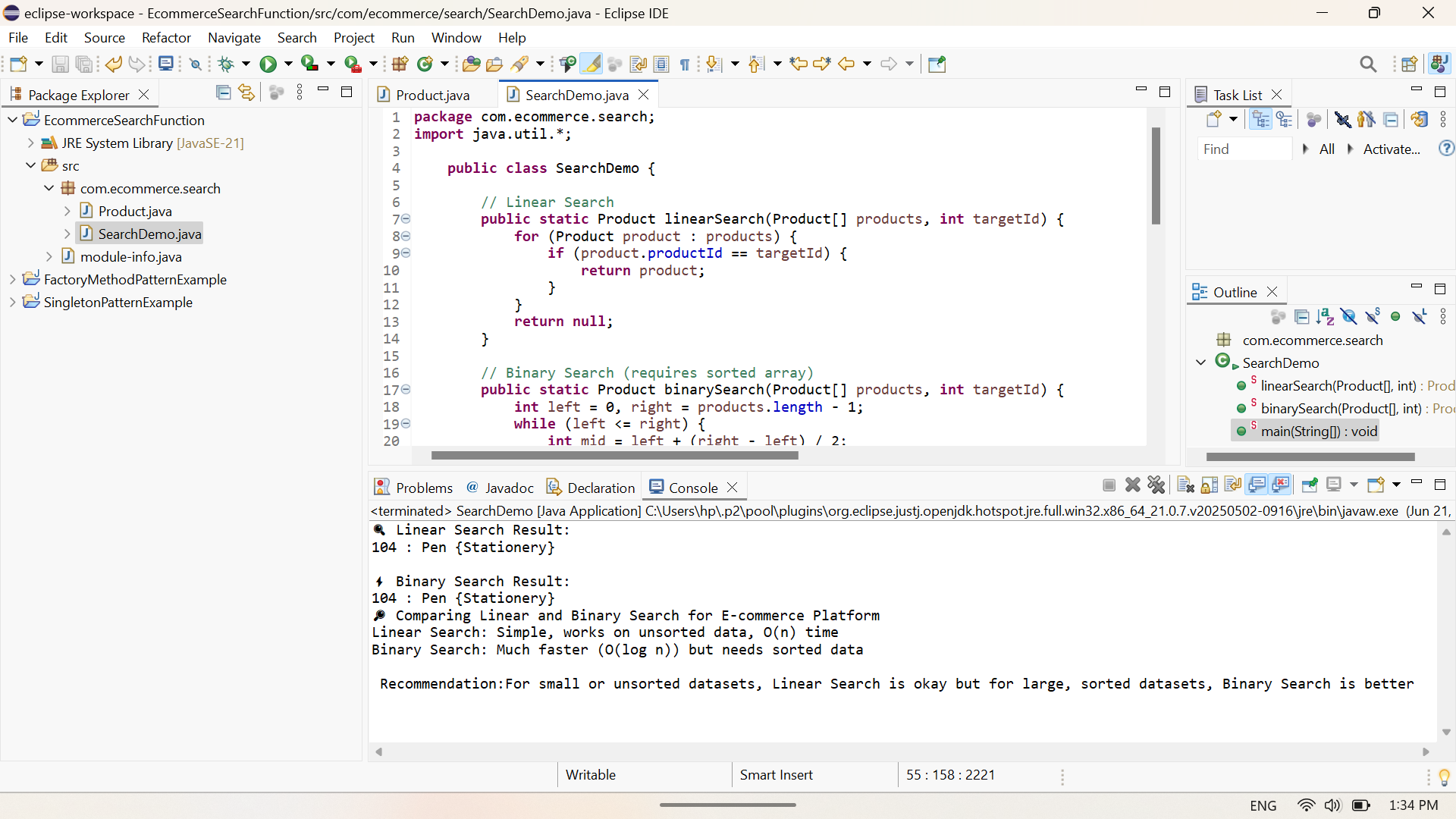
System.***out***.println("Binary Search: Much faster (O(log n)) but needs sorted data");

System.***out***.println("\n Recommendation:For small or unsorted datasets, Linear Search is okay but for large, sorted datasets, Binary Search is better");

}

}

**Output:**

****