Implementing the Singleton Pattern

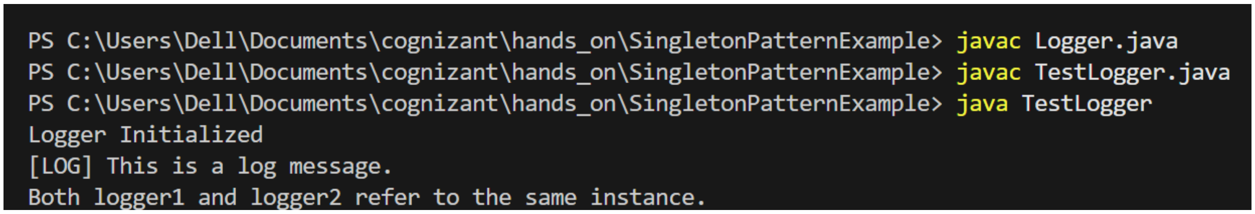
Logger.java

```java
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);
    }
}
```

TestLogger.java

```java
public class TestLogger {
    public static void main(String[] args) {
        Logger logger1 = Logger.getInstance();
        Logger logger2 = Logger.getInstance();

        logger1.log("This is a log message.");

        if (logger1 == logger2) {
            System.out.println("Both logger1 and logger2 refer to the same 
instance.");
        } else {
            System.out.println("Different instances exist (Singleton failed).");
        }
    }
}
```

Output:



```
PS C:\Users\Dell\Documents\cognizant\hands_on\SingletonPatternExample> javac Logger.java
PS C:\Users\Dell\Documents\cognizant\hands_on\SingletonPatternExample> javac TestLogger.java
PS C:\Users\Dell\Documents\cognizant\hands_on\SingletonPatternExample> java TestLogger
Logger Initialized
[LOG] This is a log message.
Both logger1 and logger2 refer to the same instance.
```

Implementing the Factory Method Pattern

Document.java

```java
public interface Document {
    void open();
}
```

WordDocument.java

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

PdfDocument.java

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

ExcelDocument.java

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

DocumentFactory.java

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

WordDocumentFactory.java

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

PdfDocumentFactory.java

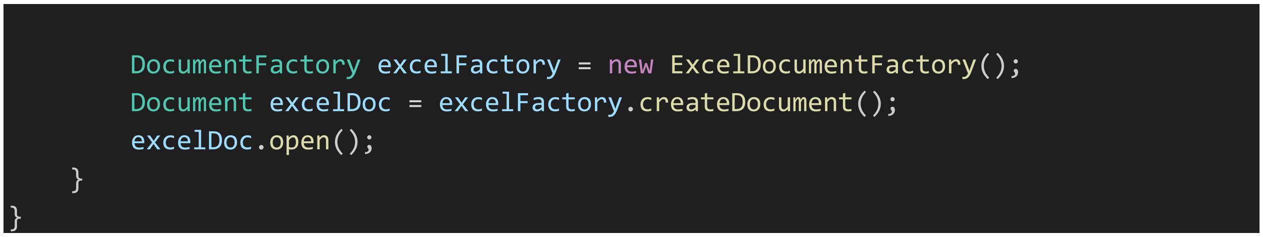
```java
public class PdfDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new PdfDocument();
    }
}
```

ExcelDocumentFactory.java

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

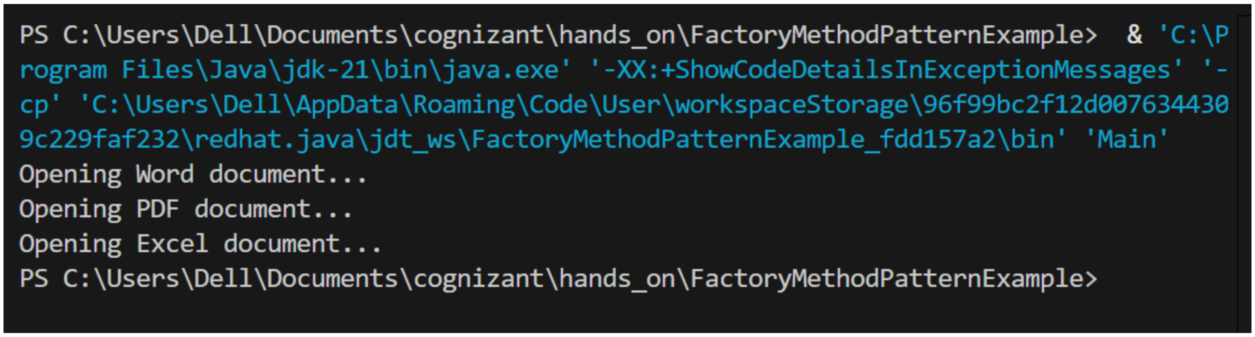
Main.java

```java
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:



```
PS C:\Users\Dell\Documents\cognizant\hands_on\FactoryMethodPatternExample>  & 'C:\P
rogram Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-
cp' 'C:\Users\Dell\AppData\Roaming\Code\User\workspaceStorage\96f99bc2f12d007634430
9c229faf232\redhat.java\jdt_ws\FactoryMethodPatternExample_fdd157a2\bin' 'Main'
Opening Word document...
Opening PDF document...
Opening Excel document...
PS C:\Users\Dell\Documents\cognizant\hands_on\FactoryMethodPatternExample>
```

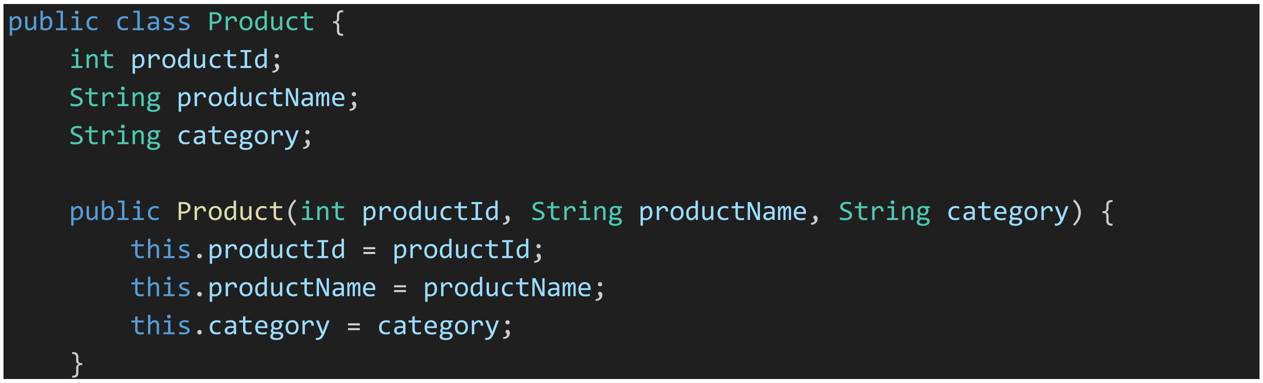
E-commerce Platform Search Function:

Big O notation describes the upper bound of an algorithm's running time.

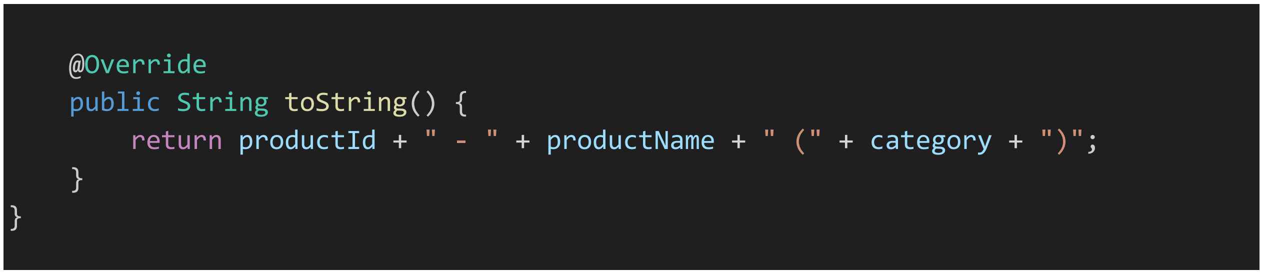
It helps us analyze how the performance of an algorithm scales with input size n.

Algorithm	Best Case	Average Case	Worst Case
Linear Search	O(1)	O(n)	O(n)
Binary Search	O(1)	O(log n)	O(log n)

Product.java

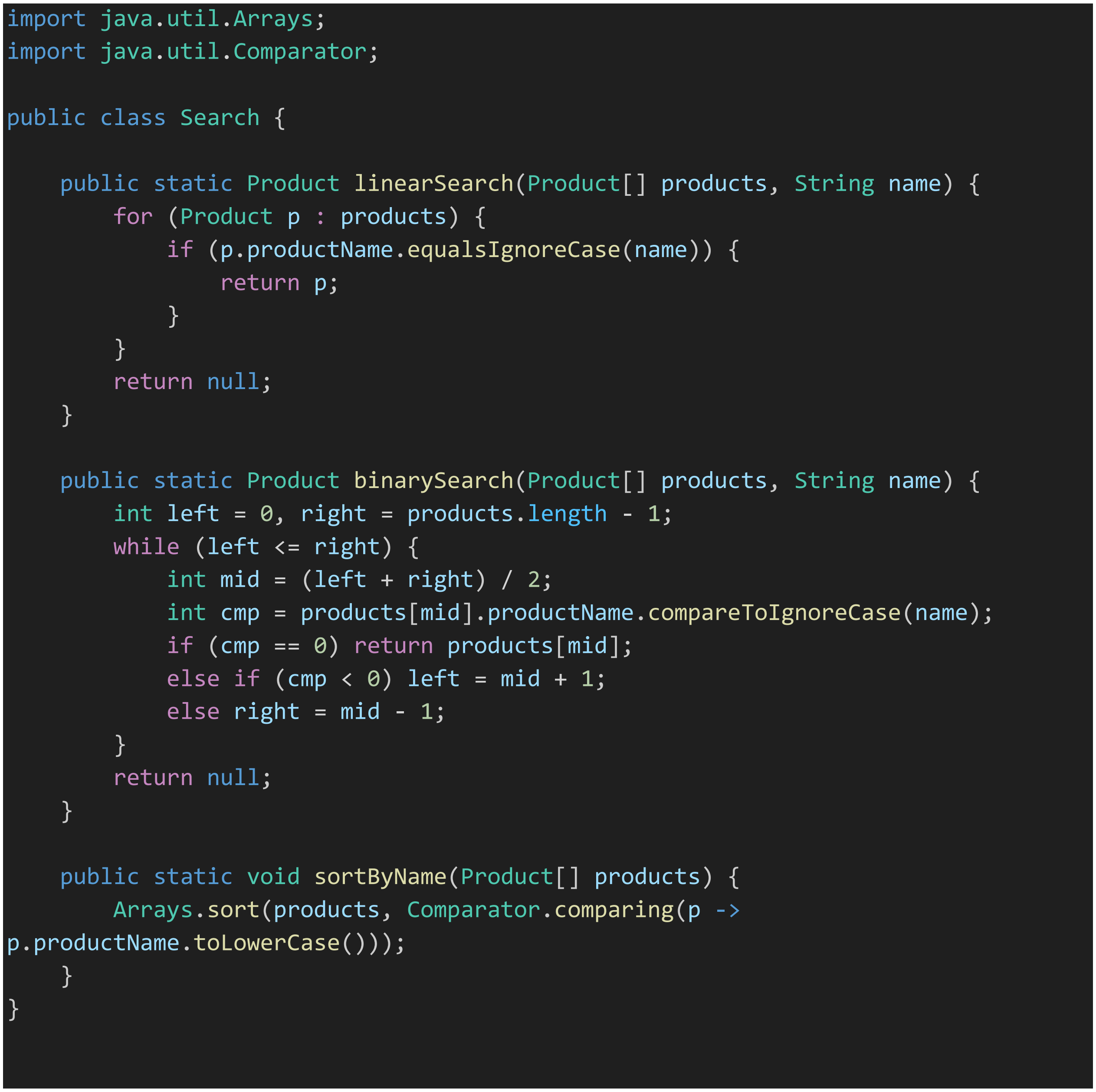


```
public class Product {
    int productId;
    String productName;
    String category;

    public Product(int productId, String productName, String category) {
        this.productId = productId;
        this.productName = productName;
        this.category = category;
    }
```



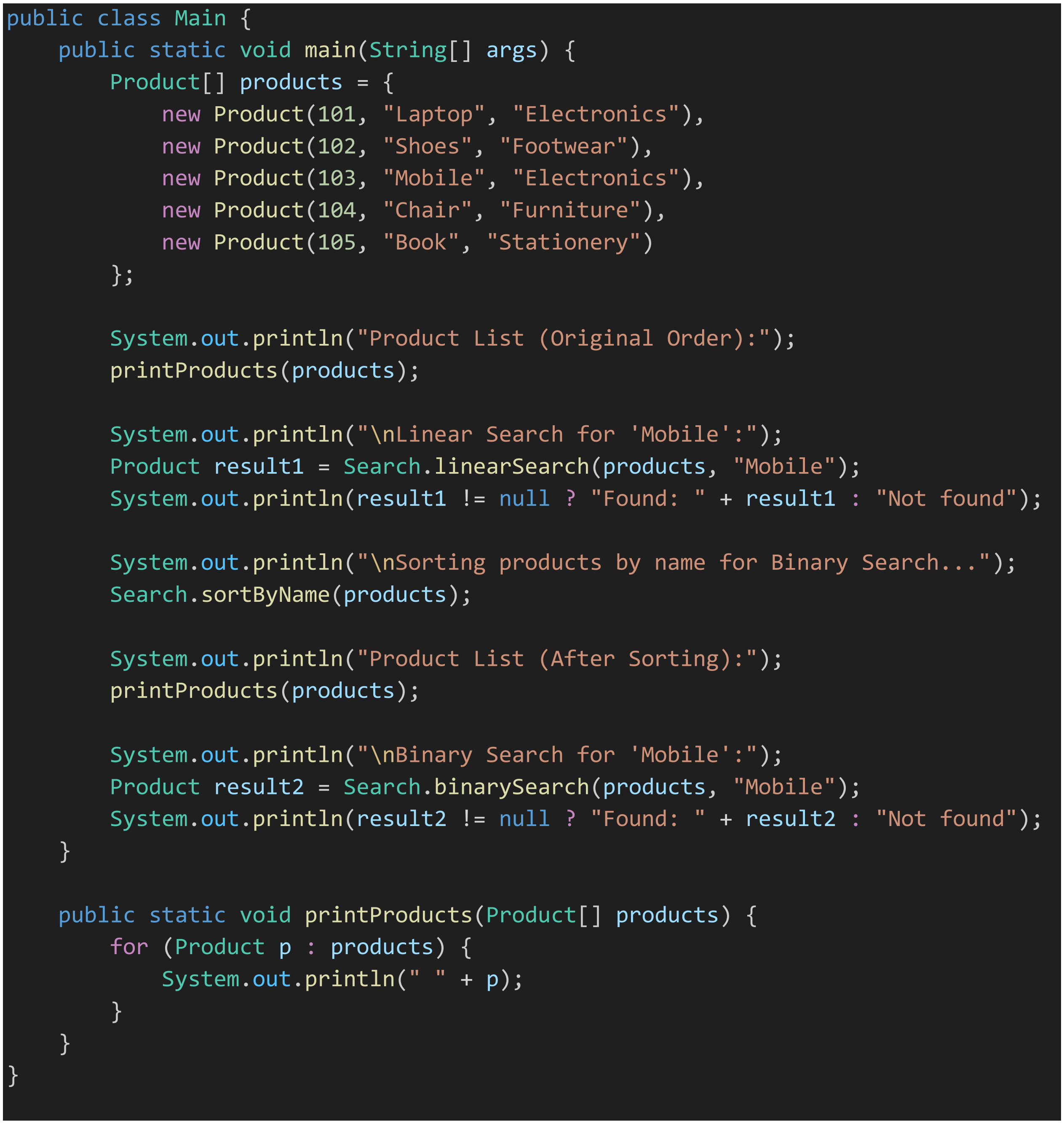
```java
    @Override
    public String toString() {
        return productId + " - " + productName + " (" + category + ")";
    }
}
```

Search.java

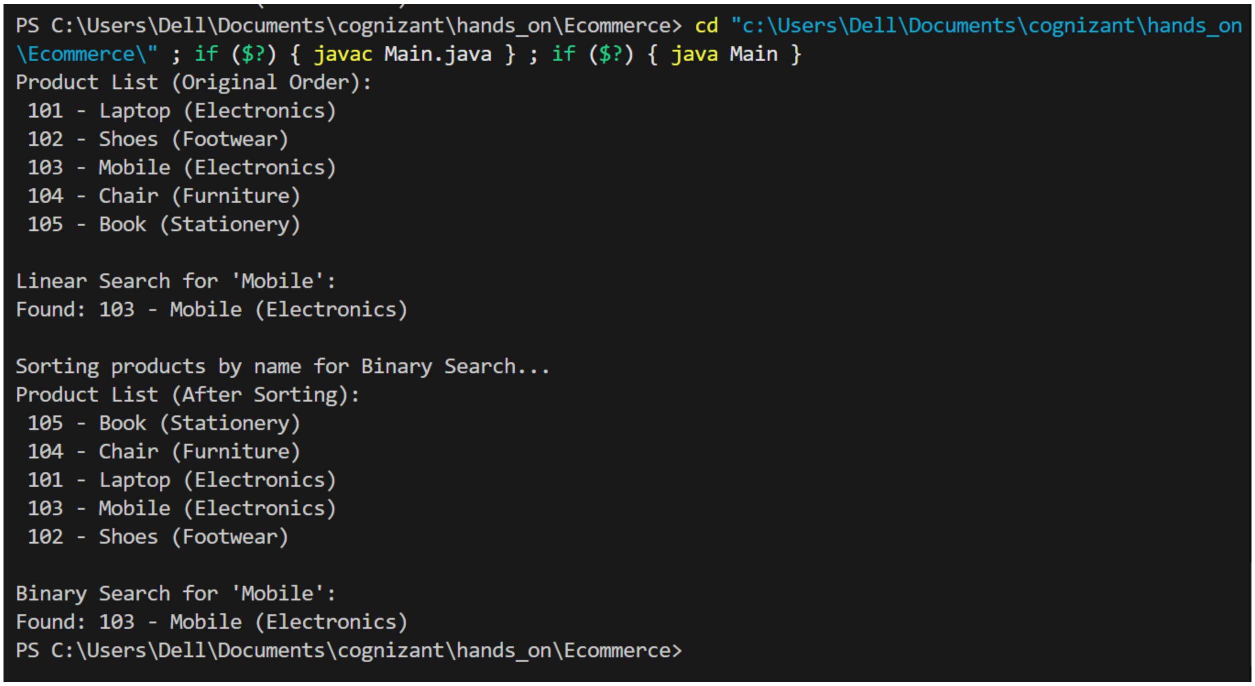


```java
import java.util.Arrays;
import java.util.Comparator;

public class Search {

    public static Product linearSearch(Product[] products, String name) {
        for (Product p : products) {
            if (p.productName.equalsIgnoreCase(name)) {
                return p;
            }
        }
        return null;
    }

    public static Product binarySearch(Product[] products, String name) {
        int left = 0, right = products.length - 1;
        while (left <= right) {
            int mid = (left + right) / 2;
            int cmp = products[mid].productName.compareToIgnoreCase(name);
            if (cmp == 0) return products[mid];
            else if (cmp < 0) left = mid + 1;
            else right = mid - 1;
        }
        return null;
    }

    public static void sortByName(Product[] products) {
        Arrays.sort(products, Comparator.comparing(p ->
p.productName.toLowerCase()));
    }
}
```

Main.java



```java
public class Main {
    public static void main(String[] args) {
        Product[] products = {
            new Product(101, "Laptop", "Electronics"),
            new Product(102, "Shoes", "Footwear"),
            new Product(103, "Mobile", "Electronics"),
            new Product(104, "Chair", "Furniture"),
            new Product(105, "Book", "Stationery")
        };

        System.out.println("Product List (Original Order):");
        printProducts(products);

        System.out.println("\nLinear Search for 'Mobile':");
        Product result1 = Search.linearSearch(products, "Mobile");
        System.out.println(result1 != null ? "Found: " + result1 : "Not found");

        System.out.println("\nSorting products by name for Binary Search...");
        Search.sortByName(products);

        System.out.println("Product List (After Sorting):");
        printProducts(products);

        System.out.println("\nBinary Search for 'Mobile':");
        Product result2 = Search.binarySearch(products, "Mobile");
        System.out.println(result2 != null ? "Found: " + result2 : "Not found");
    }

    public static void printProducts(Product[] products) {
        for (Product p : products) {
            System.out.println(" " + p);
        }
    }
}
```



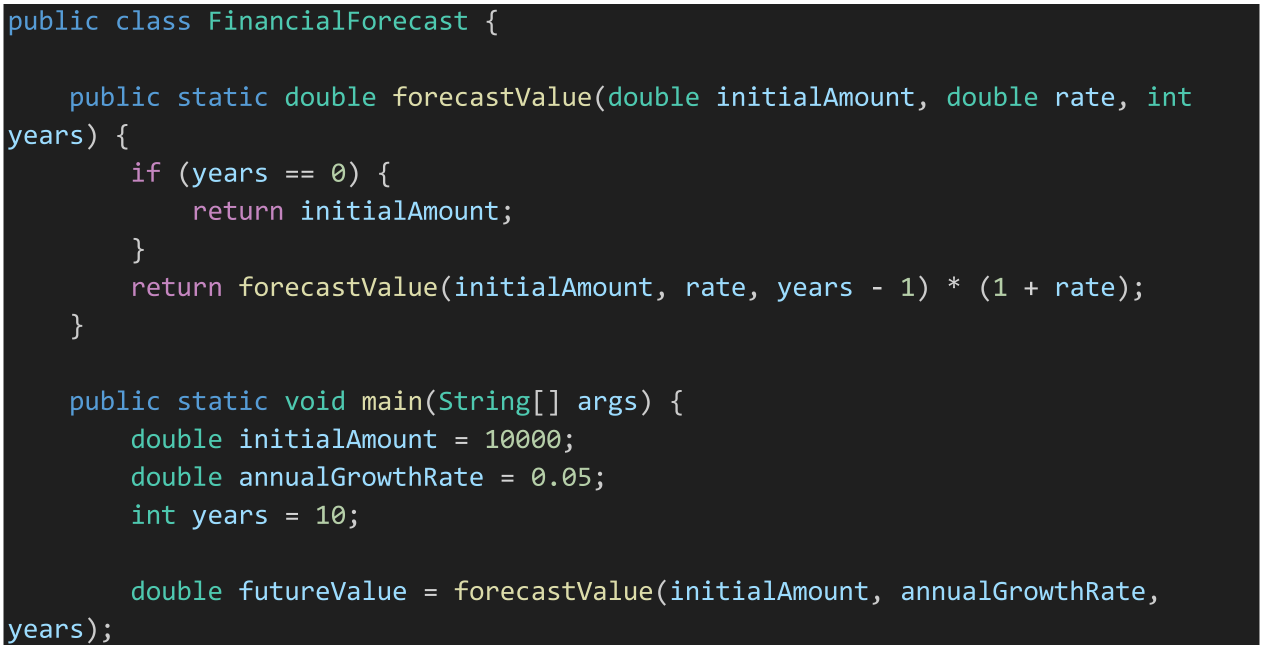
```
PS C:\Users\Dell\Documents\cognizant\hands_on\Ecommerce> cd "c:\Users\Dell\Documents\cognizant\hands_on
\Ecommerce\" ; if ($?) { javac Main.java } ; if ($?) { java Main }
Product List (Original Order):
 101 - Laptop (Electronics)
 102 - Shoes (Footwear)
 103 - Mobile (Electronics)
 104 - Chair (Furniture)
 105 - Book (Stationery)

Linear Search for 'Mobile':
Found: 103 - Mobile (Electronics)

Sorting products by name for Binary Search...
Product List (After Sorting):
 105 - Book (Stationery)
 104 - Chair (Furniture)
 101 - Laptop (Electronics)
 103 - Mobile (Electronics)
 102 - Shoes (Footwear)

Binary Search for 'Mobile':
Found: 103 - Mobile (Electronics)
PS C:\Users\Dell\Documents\cognizant\hands_on\Ecommerce>
```

Financial Forecasting:

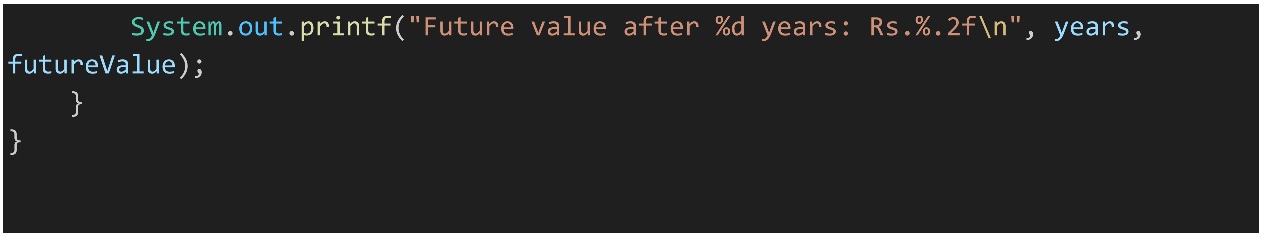
Recursion is when a method calls itself to solve a smaller instance of the same problem.

It simplifies complex problems by breaking them down into base cases and recursive cases.

FinancialForecast.java

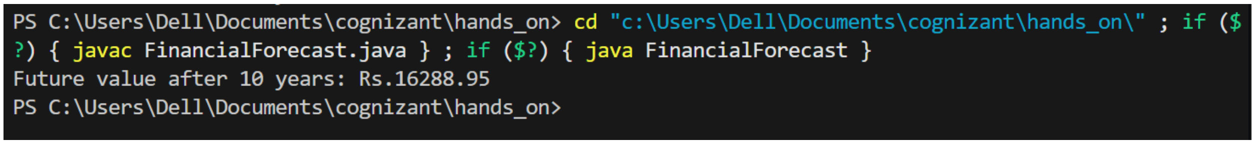


```
public class FinancialForecast {

    public static double forecastValue(double initialAmount, double rate, int
years) {
        if (years == 0) {
            return initialAmount;
        }
        return forecastValue(initialAmount, rate, years - 1) * (1 + rate);
    }

    public static void main(String[] args) {
        double initialAmount = 10000;
        double annualGrowthRate = 0.05;
        int years = 10;

        double futureValue = forecastValue(initialAmount, annualGrowthRate,
years);
```



```
        System.out.printf("Future value after %d years: Rs.%.2f\n", years,
futureValue);
    }
}
```

Output:



```
PS C:\Users\Dell\Documents\cognizant\hands_on> cd "c:\Users\Dell\Documents\cognizant\hands_on\" ; if ($
?) { javac FinancialForecast.java } ; if ($?) { java FinancialForecast }
Future value after 10 years: Rs.16288.95
PS C:\Users\Dell\Documents\cognizant\hands_on>
```

Time Complexity: O(n)

Optimization: Use iterative approach