Exercise 2: E-commerce Platform Search Function

Code:

Product.java

```
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 + ")";
  }

}
```

SearchTest.java:

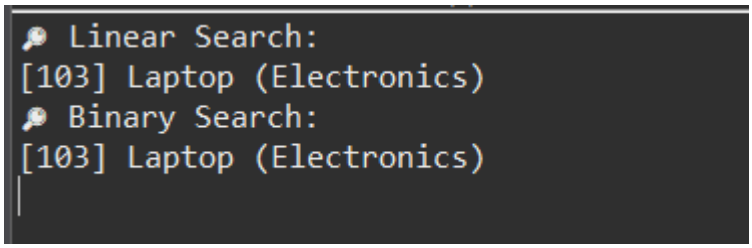
```
public class SearchTest {

 public static void main(String[] args) {
  Product[] products = {
        new Product(101, "Shoes", "Footwear"),
        new Product(102, "T-shirt", "Clothing"),
        new Product(103, "Laptop", "Electronics"),
        new Product(104, "Mobile", "Electronics"),
        new Product(105, "Socks", "Footwear")
      };

      String searchTarget = "Laptop";
      System.out.println(" 🔎 Linear Search:");
      Product result1 = SearchUtility.linearSearch(products, searchTarget);
```

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

        System.out.println("🔎 Binary Search:");
        Product result2 = SearchUtility.binarySearch(products, searchTarget);
        System.out.println(result2 != null ? result2 : "Product not found.");

    }

}
```

SearchUtility.java

```
import java.util.*;
public class SearchUtility {
    public static Product linearSearch(Product[] products, String name) {
        for (Product product : products) {
            if (product.getProductName().equalsIgnoreCase(name)) {
                return product;
            }
        }
        return null;
    }

    public static Product binarySearch(Product[] products, String name) {
        Arrays.sort(products, Comparator.comparing(Product::getProductName));
        int low = 0, high = products.length - 1;

        while (low <= high) {
            int mid = (low + high) / 2;
            int comparison = name.compareToIgnoreCase(products[mid].getProductName());

            if (comparison == 0) {
                return products[mid];
            } else if (comparison < 0) {
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }
        return null;
    }

}
```

OUTPUT:



Exercise 7: Financial Forecasting

Code:

FinancialForecaster.java

```
public class FinancialForecaster {
   public static double forecastRecursive(double presentValue, double growthRate, int years) {
      if (years == 0) return presentValue;
      return (1 + growthRate) * forecastRecursive(presentValue, growthRate, years - 1);
   }

   // Optimized version using memoization
   public static double forecastMemo(double presentValue, double growthRate, int years, Double[]
memo) {
      if (years == 0) return presentValue;
      if (memo[years] != null) return memo[years];

      memo[years] = (1 + growthRate) * forecastMemo(presentValue, growthRate, years - 1, memo);
      return memo[years];
   }

}
```

ForecastTest.java

```
public class ForecastTest {

   public static void main(String[] args) {
      double presentValue = 10000.0;
      double growthRate = 0.05;
      int years = 5;

      double forecast = FinancialForecaster.forecastRecursive(presentValue, growthRate, years);
      System.out.println("Recursive Forecast after " + years + " years: ₹" + forecast);

      Double[] memo = new Double[years + 1];
      double forecastMemoized = FinancialForecaster.forecastMemo(presentValue, growthRate, years,
memo);
      System.out.println("Optimized Forecast with Memoization: ₹" + forecastMemoized);

   }

}
```

OUTPUT: