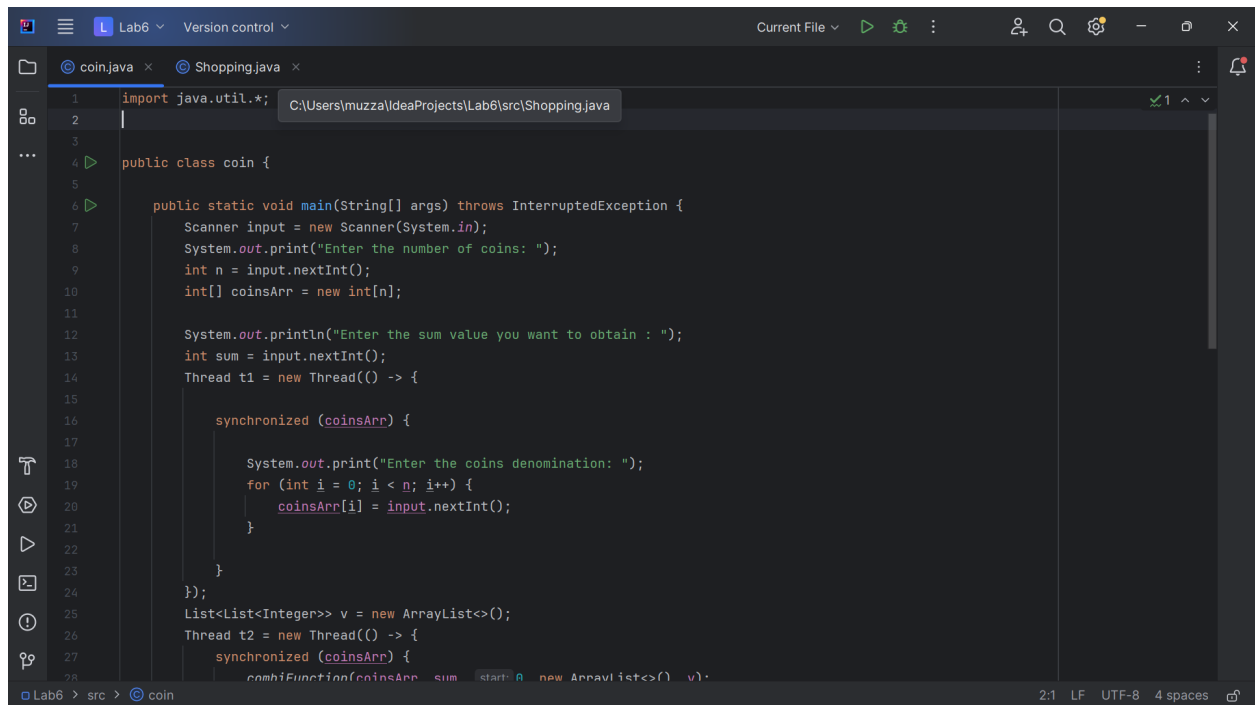
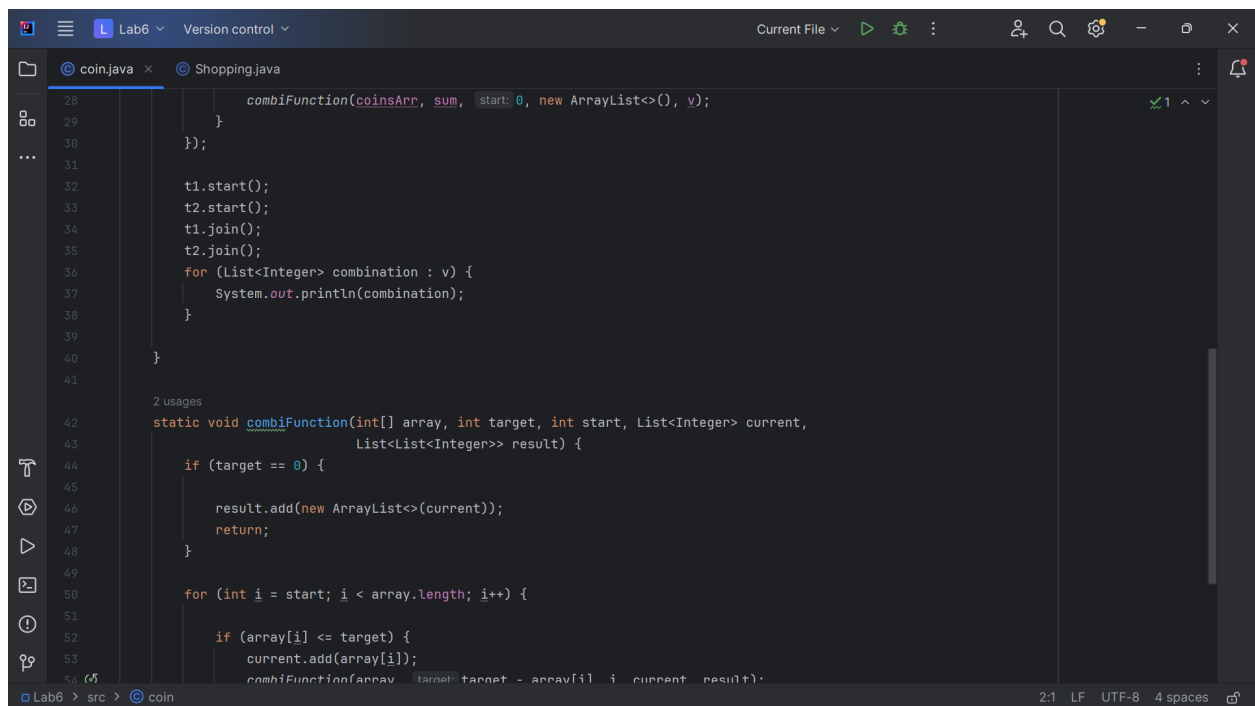


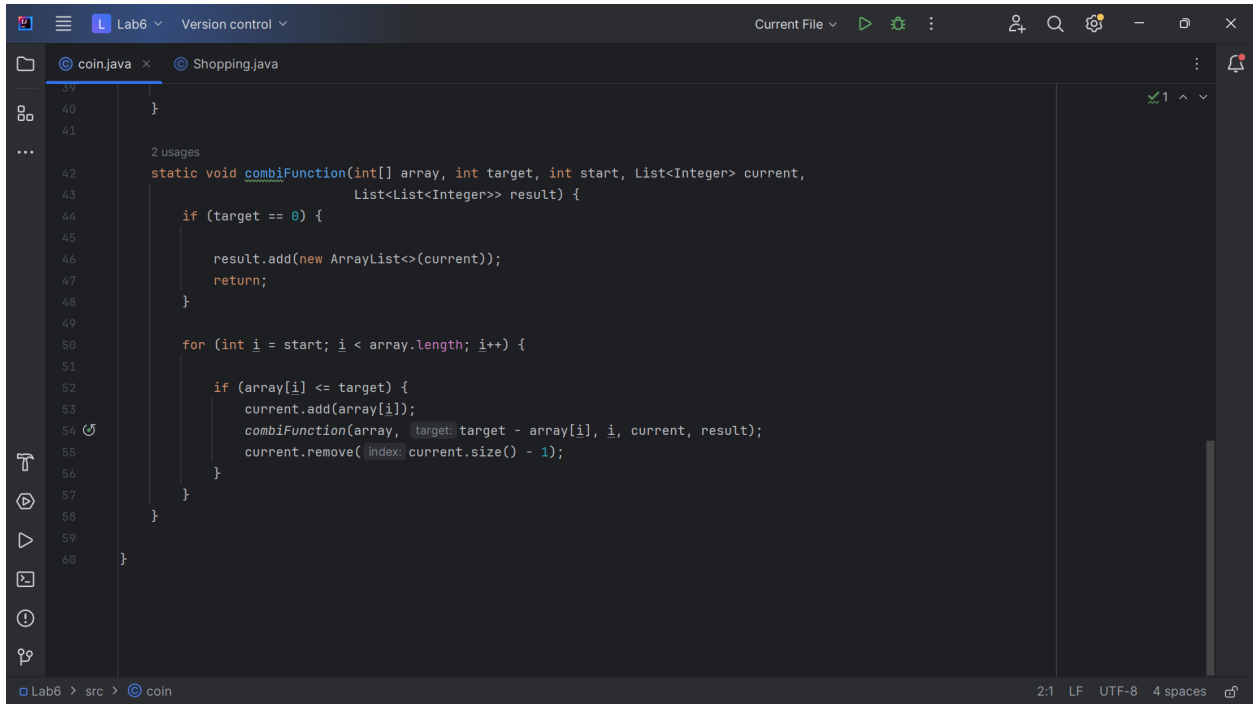
Program 1:



```
1 import java.util.*;
2
3
4 public class coin {
5
6     public static void main(String[] args) throws InterruptedException {
7         Scanner input = new Scanner(System.in);
8         System.out.print("Enter the number of coins: ");
9         int n = input.nextInt();
10        int[] coinsArr = new int[n];
11
12        System.out.println("Enter the sum value you want to obtain : ");
13        int sum = input.nextInt();
14        Thread t1 = new Thread(() -> {
15
16            synchronized (coinsArr) {
17
18                System.out.print("Enter the coins denomination: ");
19                for (int i = 0; i < n; i++) {
20                    coinsArr[i] = input.nextInt();
21                }
22            }
23        });
24        List<List<Integer>> v = new ArrayList<>();
25        Thread t2 = new Thread(() -> {
26            synchronized (coinsArr) {
27                combiFunction(coinsArr, sum, start: 0, new ArrayList<>(), v);
28            }
29        });
30        t1.start();
31        t2.start();
32        t1.join();
33        t2.join();
34        for (List<Integer> combination : v) {
35            System.out.println(combination);
36        }
37    }
38}
```



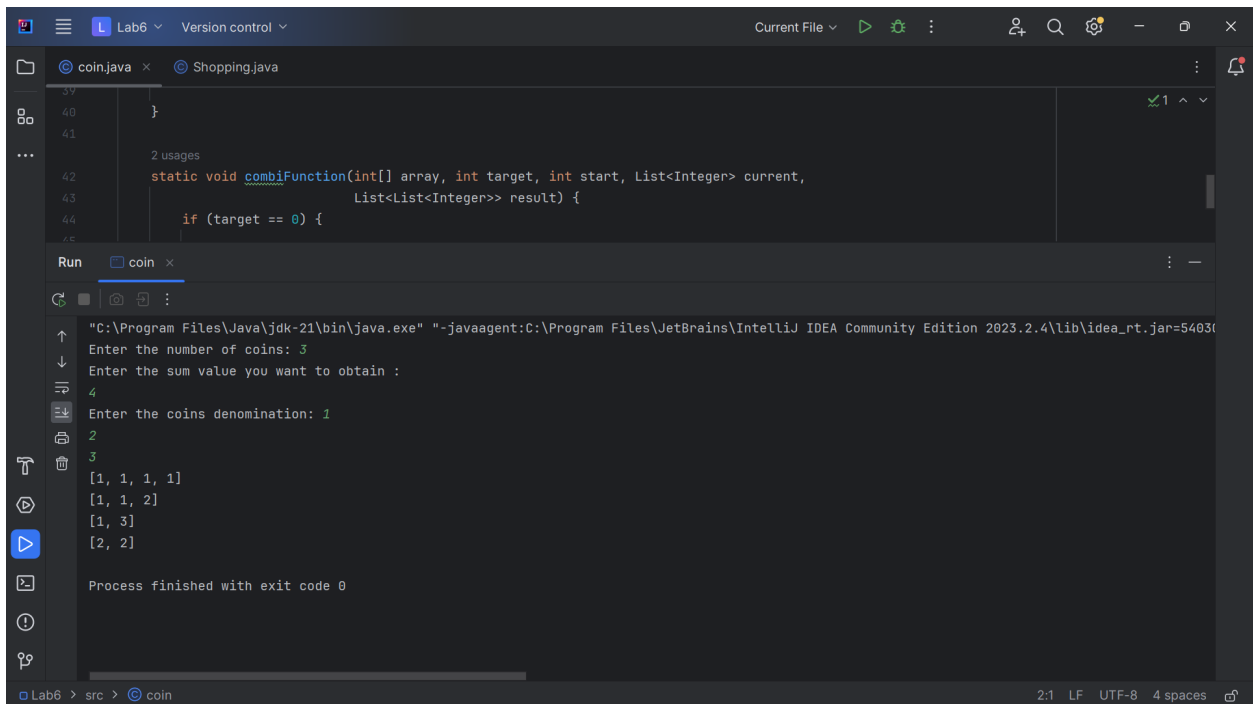
```
28        combiFunction(coinsArr, sum, start: 0, new ArrayList<>(), v);
29    }
30}
31
32t1.start();
33t2.start();
34t1.join();
35t2.join();
36for (List<Integer> combination : v) {
37    System.out.println(combination);
38}
39
40}
41
422 usages
43static void combiFunction(int[] array, int target, int start, List<Integer> current,
44    List<List<Integer>> result) {
45    if (target == 0) {
46        result.add(new ArrayList<>(current));
47        return;
48    }
49    for (int i = start; i < array.length; i++) {
50        if (array[i] <= target) {
51            current.add(array[i]);
52            combiFunction(array, target - array[i], i, current, result);
53        }
54    }
55}
```



```
39
40 }
41
42 2 usages
43 static void combiFunction(int[] array, int target, int start, List<Integer> current,
44                          List<List<Integer>> result) {
45     if (target == 0) {
46         result.add(new ArrayList<>(current));
47         return;
48     }
49
50     for (int i = start; i < array.length; i++) {
51
52         if (array[i] <= target) {
53             current.add(array[i]);
54             combiFunction(array, target - array[i], i, current, result);
55             current.remove(index: current.size() - 1);
56         }
57     }
58 }
59
60 }
```

Lab6 > src > coin 2:1 LF UTF-8 4 spaces

Output:

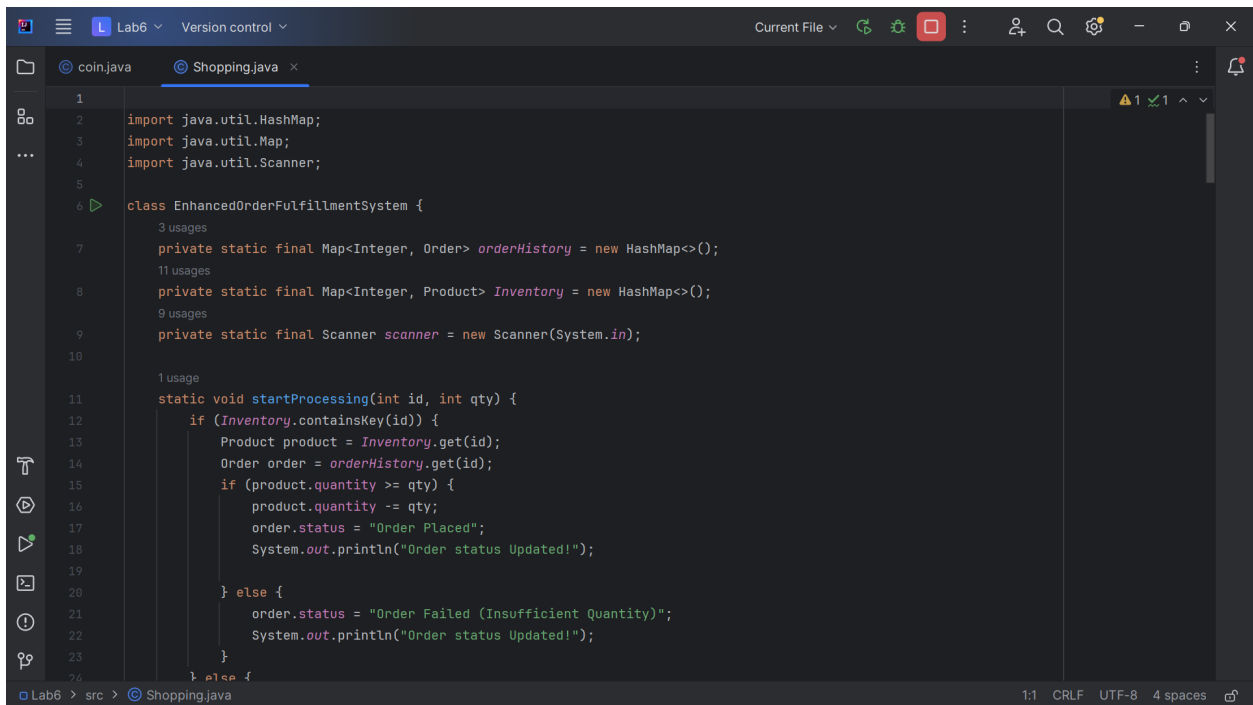


```
Run coin x
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.4\lib\idea_rt.jar=5403(
Enter the number of coins: 3
Enter the sum value you want to obtain :
4
Enter the coins denomination: 1
2
3
[1, 1, 1, 1]
[1, 1, 2]
[1, 3]
[2, 2]

Process finished with exit code 0

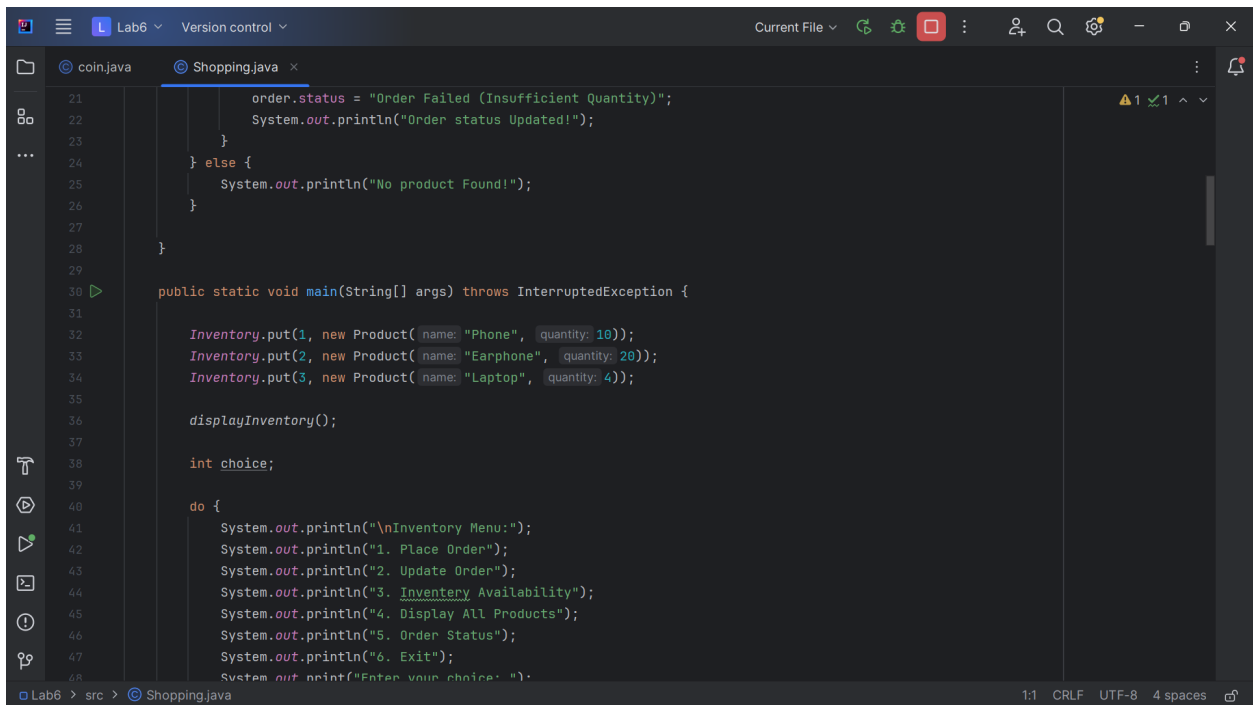
Lab6 > src > coin 2:1 LF UTF-8 4 spaces
```

Program 2:



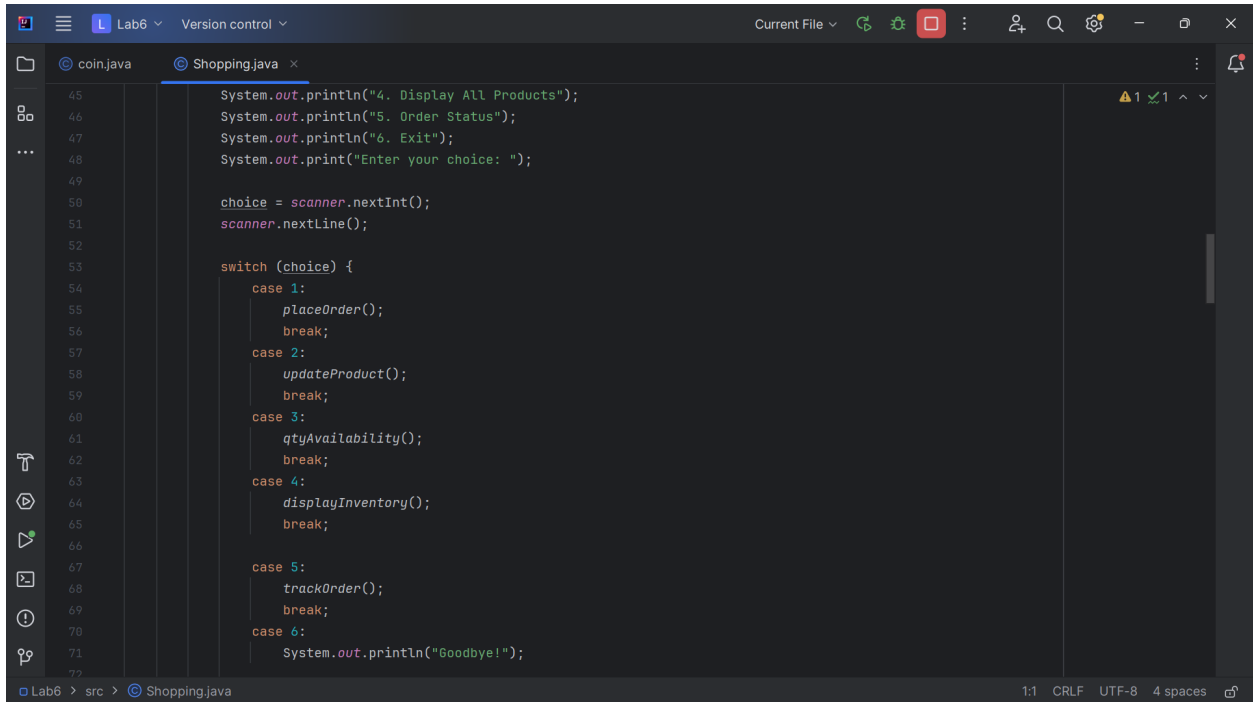
This screenshot shows the first part of the `Shopping.java` file in an IDE. The code includes imports for `HashMap`, `Map`, and `Scanner`. It defines a class `EnhancedOrderFulfillmentSystem` with static fields for `orderHistory`, `Inventory`, and `scanner`. A static method `startProcessing` is defined, which checks if a product exists in the inventory and updates its status and quantity.

```
1
2 import java.util.HashMap;
3 import java.util.Map;
4 import java.util.Scanner;
5
6 class EnhancedOrderFulfillmentSystem {
7     3 usages
8     private static final Map<Integer, Order> orderHistory = new HashMap<>();
9     11 usages
10    private static final Map<Integer, Product> Inventory = new HashMap<>();
11    9 usages
12    private static final Scanner scanner = new Scanner(System.in);
13
14    1 usage
15    static void startProcessing(int id, int qty) {
16        if (Inventory.containsKey(id)) {
17            Product product = Inventory.get(id);
18            Order order = orderHistory.get(id);
19            if (product.quantity >= qty) {
20                product.quantity -= qty;
21                order.status = "Order Placed";
22                System.out.println("Order status Updated!");
23            } else {
24                order.status = "Order Failed (Insufficient Quantity)";
25                System.out.println("Order status Updated!");
26            }
27        } else {
28            System.out.println("No product Found!");
29        }
30    }
31 }
```



This screenshot shows the second part of the `Shopping.java` file. It continues the `startProcessing` method and includes a `main` method that initializes the inventory with products like Phone, Earphone, and Laptop. It also includes a `displayInventory` method and a menu-driven loop for user interaction.

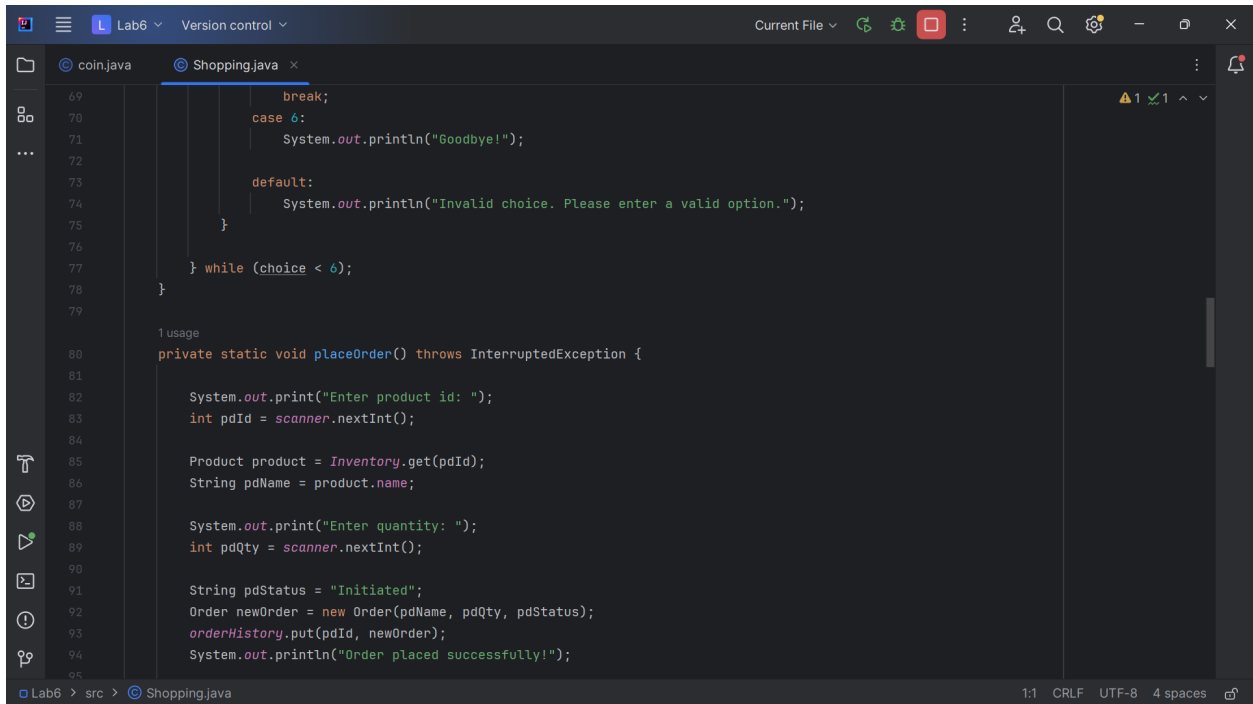
```
21
22     order.status = "Order Failed (Insufficient Quantity)";
23     System.out.println("Order status Updated!");
24 }
25 } else {
26     System.out.println("No product Found!");
27 }
28 }
29
30 public static void main(String[] args) throws InterruptedException {
31
32     Inventory.put(1, new Product( name: "Phone", quantity: 10));
33     Inventory.put(2, new Product( name: "Earphone", quantity: 20));
34     Inventory.put(3, new Product( name: "Laptop", quantity: 4));
35
36     displayInventory();
37
38     int choice;
39
40     do {
41         System.out.println("\nInventory Menu:");
42         System.out.println("1. Place Order");
43         System.out.println("2. Update Order");
44         System.out.println("3. Inventory Availability");
45         System.out.println("4. Display All Products");
46         System.out.println("5. Order Status");
47         System.out.println("6. Exit");
48         System.out.print("Enter your choice: ");
49     } while (choice != 6);
50 }
```



This screenshot shows the Shopping.java file in an IDE. The code includes a switch statement that handles six different choices. Choice 1 calls placeOrder(), choice 2 calls updateProduct(), choice 3 calls qtyAvailability(), choice 4 calls displayInventory(), choice 5 calls trackOrder(), and choice 6 prints a goodbye message. The code is as follows:

```
45      System.out.println("4. Display All Products");
46      System.out.println("5. Order Status");
47      System.out.println("6. Exit");
48      System.out.print("Enter your choice: ");
49
50      choice = scanner.nextInt();
51      scanner.nextLine();
52
53      switch (choice) {
54          case 1:
55              placeOrder();
56              break;
57          case 2:
58              updateProduct();
59              break;
60          case 3:
61              qtyAvailability();
62              break;
63          case 4:
64              displayInventory();
65              break;
66
67          case 5:
68              trackOrder();
69              break;
70          case 6:
71              System.out.println("Goodbye!");
72      }
```

The IDE interface shows the file path as Lab6 > src > Shopping.java and the status bar indicates 1:1 CRLF UTF-8 4 spaces.



This screenshot shows the continuation of the Shopping.java file. It includes a while loop that repeats the choice handling process as long as the choice is less than 6. Below the loop is the implementation of the placeOrder() method, which prompts the user for a product ID and quantity, retrieves the product from the inventory, creates a new order, and prints a success message. The code is as follows:

```
69      break;
70      case 6:
71          System.out.println("Goodbye!");
72
73      default:
74          System.out.println("Invalid choice. Please enter a valid option.");
75      }
76
77      } while (choice < 6);
78  }
79
80  1 usage
81  private static void placeOrder() throws InterruptedException {
82
83      System.out.print("Enter product id: ");
84      int pdId = scanner.nextInt();
85
86      Product product = Inventory.get(pdId);
87      String pdName = product.name;
88
89      System.out.print("Enter quantity: ");
90      int pdQty = scanner.nextInt();
91
92      String pdStatus = "Initiated";
93      Order newOrder = new Order(pdName, pdQty, pdStatus);
94      orderHistory.put(pdId, newOrder);
95      System.out.println("Order placed successfully!");
96  }
```

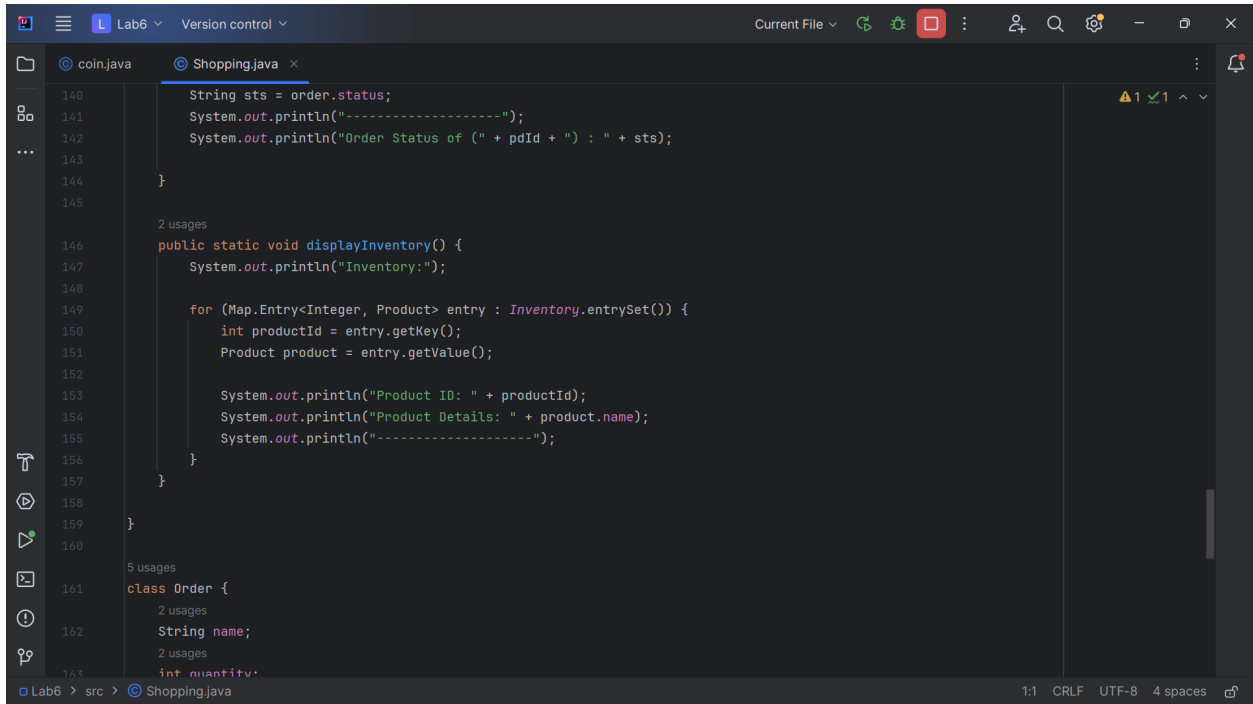
The IDE interface shows the file path as Lab6 > src > Shopping.java and the status bar indicates 1:1 CRLF UTF-8 4 spaces.

```
92      Order newOrder = new Order(pdName, pdQty, pdStatus);
93      orderHistory.put(pdId, newOrder);
94      System.out.println("Order placed successfully!");
95
96      Thread t1 = new Thread() -> {
97      {
98          startProcessing(pdId, pdQty);
99      }
100    };
101    t1.start();
102    t1.join();
103  }
104
105  1 usage
106  private static void updateProduct() {
107      System.out.print("Enter product id to update: ");
108      int pdId = scanner.nextInt();
109
110      if (Inventory.containsKey(pdId)) {
111          System.out.print("Enter the new name: ");
112          String pdName = scanner.nextLine();
113          System.out.print("Enter the new quantity: ");
114          int pdQty = scanner.nextInt();
115          Product product = Inventory.get(pdId);
116          product.name = pdName;
117          product.quantity = pdQty;
118          Inventory.put(pdId, product);
119          System.out.println("Product updated successfully!");
120      }
121      else {
122          System.out.println("Product not found!");
123      }
124  }
```

Lab6 > src > Shopping.java 1:1 CRLF UTF-8 4 spaces

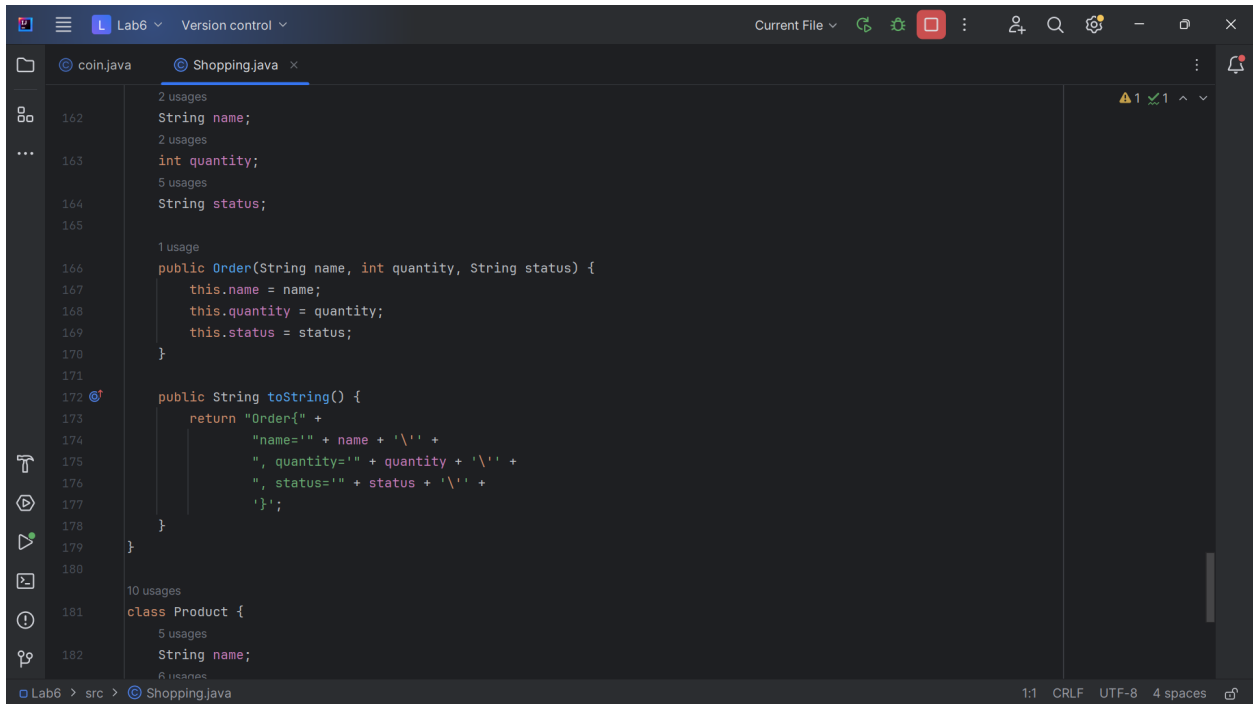
```
118      System.out.println("Product updated successfully!");
119  } else {
120      System.out.println("Product not found!");
121  }
122  }
123
124  1 usage
125  private static void qtyAvailability() {
126      System.out.print("Enter product id to retrieve: ");
127      int pdId = scanner.nextInt();
128
129      Product product = Inventory.get(pdId);
130      int qty = product.quantity;
131      System.out.println("-----");
132      System.out.println("Available Quantity: " + qty);
133  }
134
135  1 usage
136  private static void trackOrder() {
137      System.out.print("Enter product id to retrieve: ");
138      int pdId = scanner.nextInt();
139
140      Order order = orderHistory.get(pdId);
141      String sts = order.status;
142      System.out.println("-----");
143      System.out.println("Order Status of (" + pdId + ") : " + sts);
144  }
```

Lab6 > src > Shopping.java 1:1 CRLF UTF-8 4 spaces



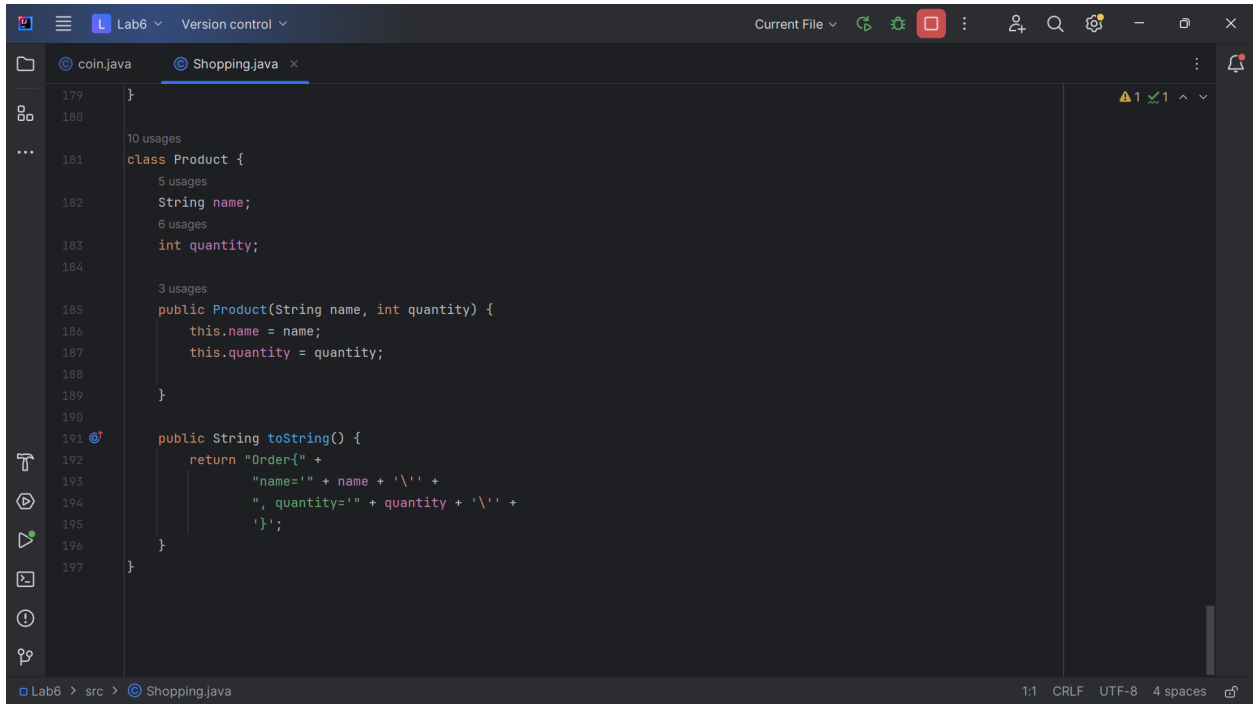
```
140     String sts = order.status;
141     System.out.println("-----");
142     System.out.println("Order Status of (" + pdId + ") : " + sts);
143
144 }
145
146 2 usages
147 public static void displayInventory() {
148     System.out.println("Inventory:");
149
150     for (Map.Entry<Integer, Product> entry : Inventory.entrySet()) {
151         int productId = entry.getKey();
152         Product product = entry.getValue();
153
154         System.out.println("Product ID: " + productId);
155         System.out.println("Product Details: " + product.name);
156         System.out.println("-----");
157     }
158 }
159
160 5 usages
161 class Order {
162     2 usages
163     String name;
164     2 usages
165     int quantity;
166 }
```

Lab6 > src > Shopping.java 1:1 CRLF UTF-8 4 spaces



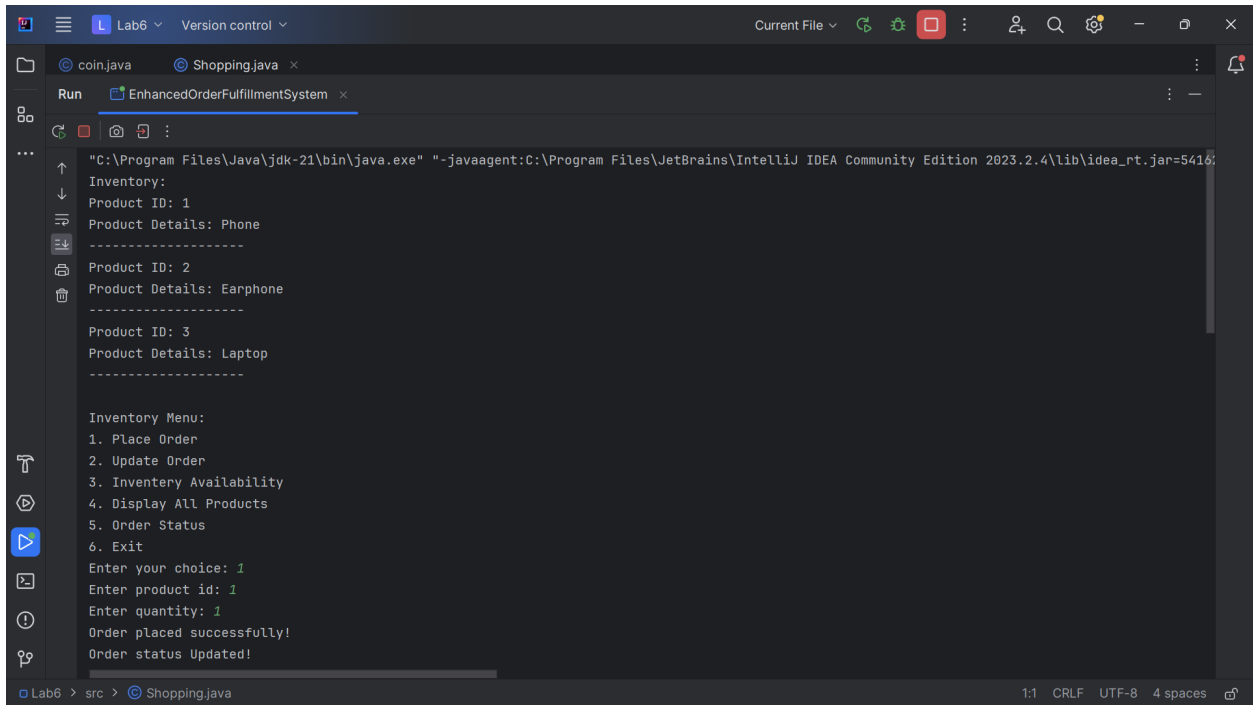
```
162 2 usages
163 String name;
164 2 usages
165 int quantity;
166 5 usages
167 String status;
168
169 1 usage
170 public Order(String name, int quantity, String status) {
171     this.name = name;
172     this.quantity = quantity;
173     this.status = status;
174 }
175
176 public String toString() {
177     return "Order{" +
178         "name='" + name + '\'' +
179         ", quantity='" + quantity + '\'' +
180         ", status='" + status + '\'' +
181         '}';
182 }
183
184 10 usages
185 class Product {
186     5 usages
187     String name;
188 }
```

Lab6 > src > Shopping.java 1:1 CRLF UTF-8 4 spaces



```
179 }
180
181 10 usages
182 class Product {
183     5 usages
184     String name;
185     6 usages
186     int quantity;
187
188     3 usages
189     public Product(String name, int quantity) {
190         this.name = name;
191         this.quantity = quantity;
192     }
193
194     public String toString() {
195         return "Order{" +
196             "name=" + name + '\'' +
197             ", quantity=" + quantity + '\'' +
198             '}';
199     }
200 }
```

Output:



```
Run EnhancedOrderFulfillmentSystem
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.4\lib\idea_rt.jar=5416:..."
Inventory:
Product ID: 1
Product Details: Phone
-----
Product ID: 2
Product Details: Earphone
-----
Product ID: 3
Product Details: Laptop
-----

Inventory Menu:
1. Place Order
2. Update Order
3. Inventory Availability
4. Display All Products
5. Order Status
6. Exit
Enter your choice: 1
Enter product id: 1
Enter quantity: 1
Order placed successfully!
Order status Updated!
```