SE 471 Software Architecture

Lab 2

Student Name		Student CSUSM ID	Contribution percentage
1	Elaeth Lilagan	200413348	33.33
2	Dalynna Nguyen	200982020	33.33
3	Rafael Refugio	200107185	33.33

100

Grading Rubrics (for instructor only):

Criteria	1. Beginning	2. Developing	3. Proficient	4. Exemplary
Modeling	0-14	15-19	20-24	25-30
Program: functionality ***correctness***	0-9	10-14	15-19	20
Program: functionality ***Behavior Testing***	0-9	10-14	15-19	20
Program: quality -> ***Readability***	0-2	3-5	6-9	10
Program: quality -> ***Modularity***	0-2	3-5	6-9	10
Program: quality -> ***Simplicity***	0-2	3-5	6-9	10
Total Grade (100)				

SE 471 Software Architecture

Problems:

The ABC Company typically uses an object of the SortingUtility class to sort products. A product has at least three attributes: ID, name and price. All are accessible through their corresponding get() method but the ID is fixed once set.

The SortingUtility class implements two private sorting algorithms, bubbleSort and quickSort, each of which takes the list of products and returns an ordered list of products. The SortingUtility class also has a public method List<Product> sort(List<Product> items, int sortingApproach), which simply calls the specified sorting approach (i.e., bubbleSort or quickSort) to return a list of sorted products to its client. Let's now assume that the SortingUtility works and is in use by some client programs.

However, one problem is that the SortingUtility currently does not log the list of sorted products before returning it to the client. Now the ABC Company would like to have an improved sorting service that can log (for this lab, simply printing to the display console) the list of sorted products before returning it to the client. To implement this improved service you can introduce another class but you cannot change **the existing** SortingUtility **class for compatible reason** (except that you may need to relate it to a super class or an interface). Moreover, the returned products from bubbleSort should be logged (printed) with ID followed by name and price, whereas the returned products from the quicksort should be logged (printed) with name first followed by ID and price.

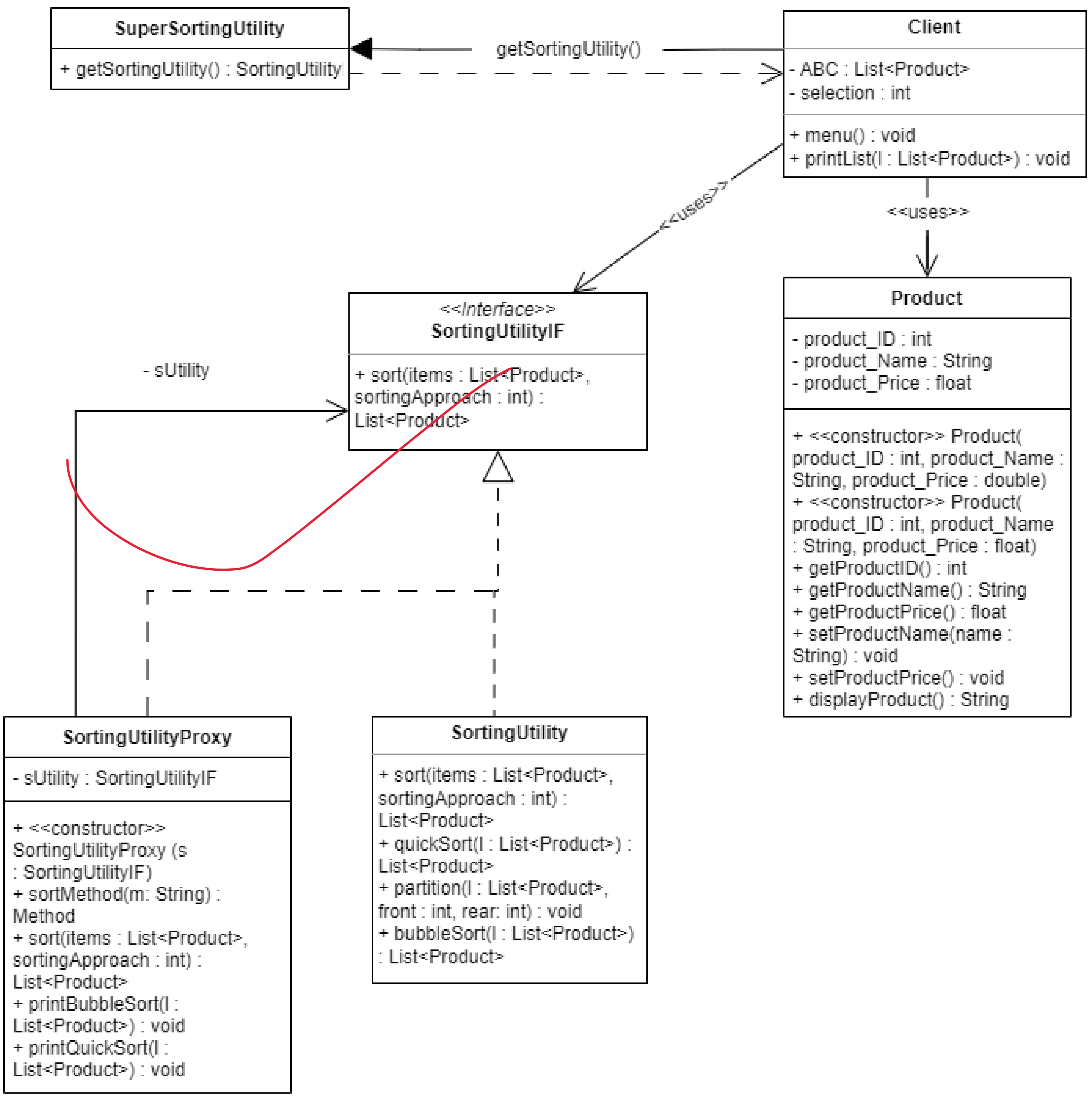
(30 pts) What design pattern can be used? Document your pattern-based design in UML class diagram, ensure attributes, methods, visibility, arguments and relationships are correctly included.

(70 pts) Implement your pattern-based design in Java. Implement two test scenarios: one using quicksort to sort a list of products such as books, bags, and buttons, another using bubblesort to sort the same list of products.

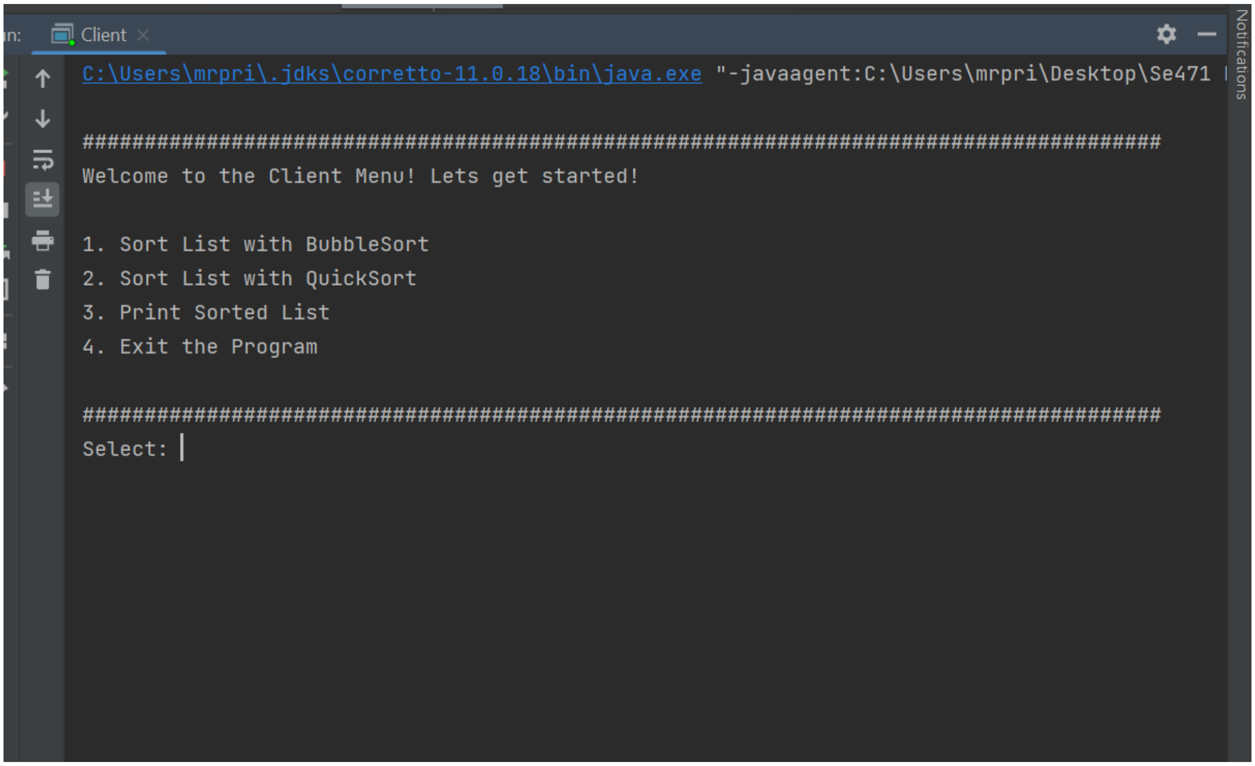
Solution:

- First, remember to zip the src folder of your project and submit the zip file to the ungraded assignment named "Lab2CodeSubmission". One submission from each team.
- Paste a screenshot of a run of your program here.
- Also paste all your source code here.
- Save this report in PDF, then **each student** needs to submit the pdf report to the graded assignment named "Lab2ReportSubmission".

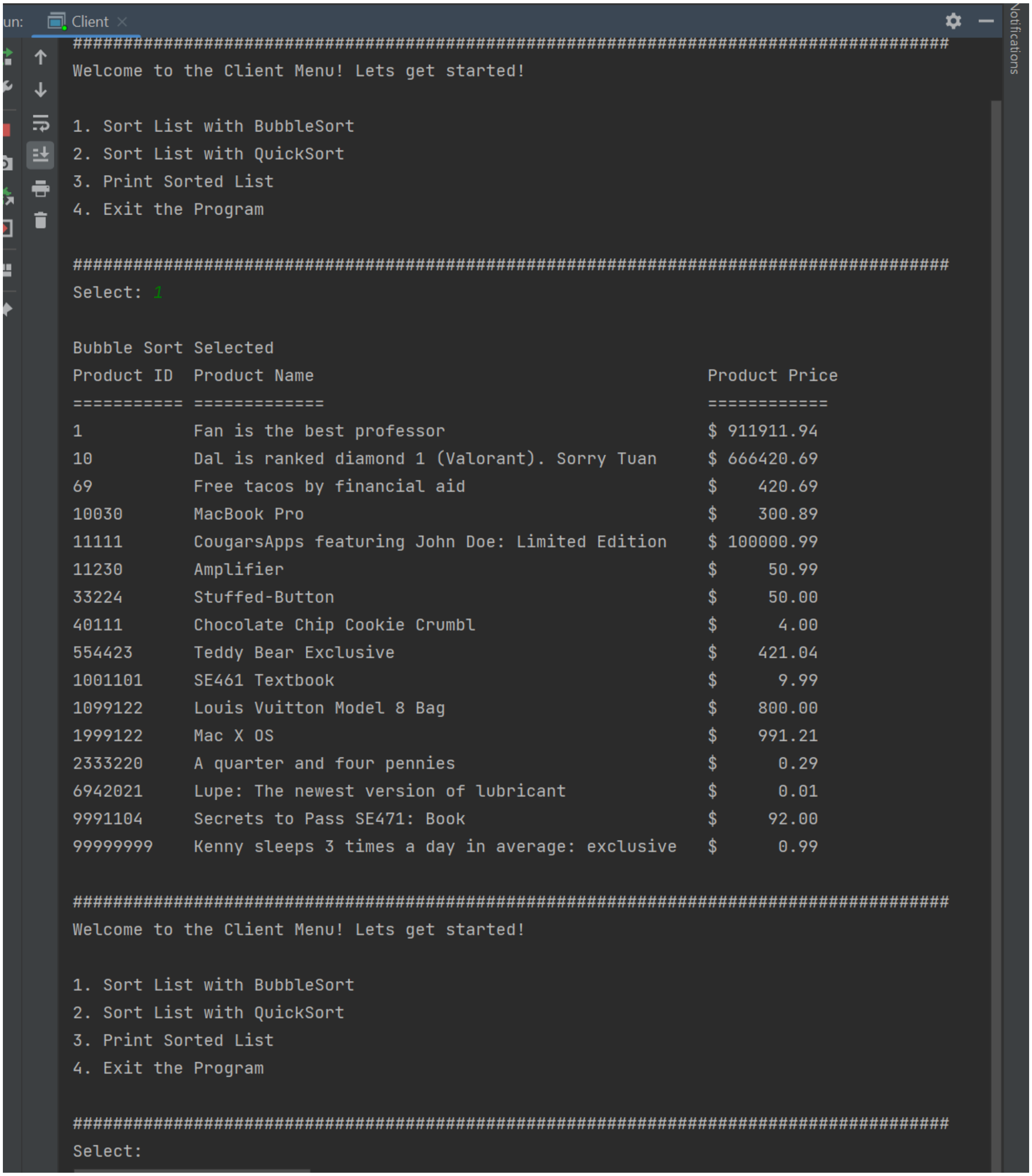
UML CLASS DIAGRAM


SuperSortingUtility
+ getSortingUtility() : SortingUtility
getSortingUtility()
Client
- ABC : List<Product>
- selection : int
+ menu() : void
+ printList(l : List<Product>) : void
<<uses>>
<<uses>>
Product
- product_ID : int
- product_Name : String
- product_Price : float
+ <<constructor>> Product(product_ID : int, product_Name : String, product_Price : double)
+ <<constructor>> Product(product_ID : int, product_Name : String, product_Price : float)
+ getProductID() : int
+ getProductName() : String
+ getProductPrice() : float
+ setProductName(name : String) : void
+ setProductPrice() : void
+ displayProduct() : String
<<Interface>>
SortingUtilityIF
+ sort(items : List<Product>, sortingApproach : int) : List<Product>
- sUtility
SortingUtilityProxy
- sUtility : SortingUtilityIF
+ <<constructor>> SortingUtilityProxy (s : SortingUtilityIF)
+ sortMethod(m: String) : Method
+ sort(items : List<Product>, sortingApproach : int) : List<Product>
+ printBubbleSort(l : List<Product>) : void
+ printQuickSort(l : List<Product>) : void
SortingUtility
+ sort(items : List<Product>, sortingApproach : int) : List<Product>
+ quickSort(l : List<Product>) : List<Product>
+ partition(l : List<Product>, front : int, rear: int) : void
+ bubbleSort(l : List<Product>) : List<Product>

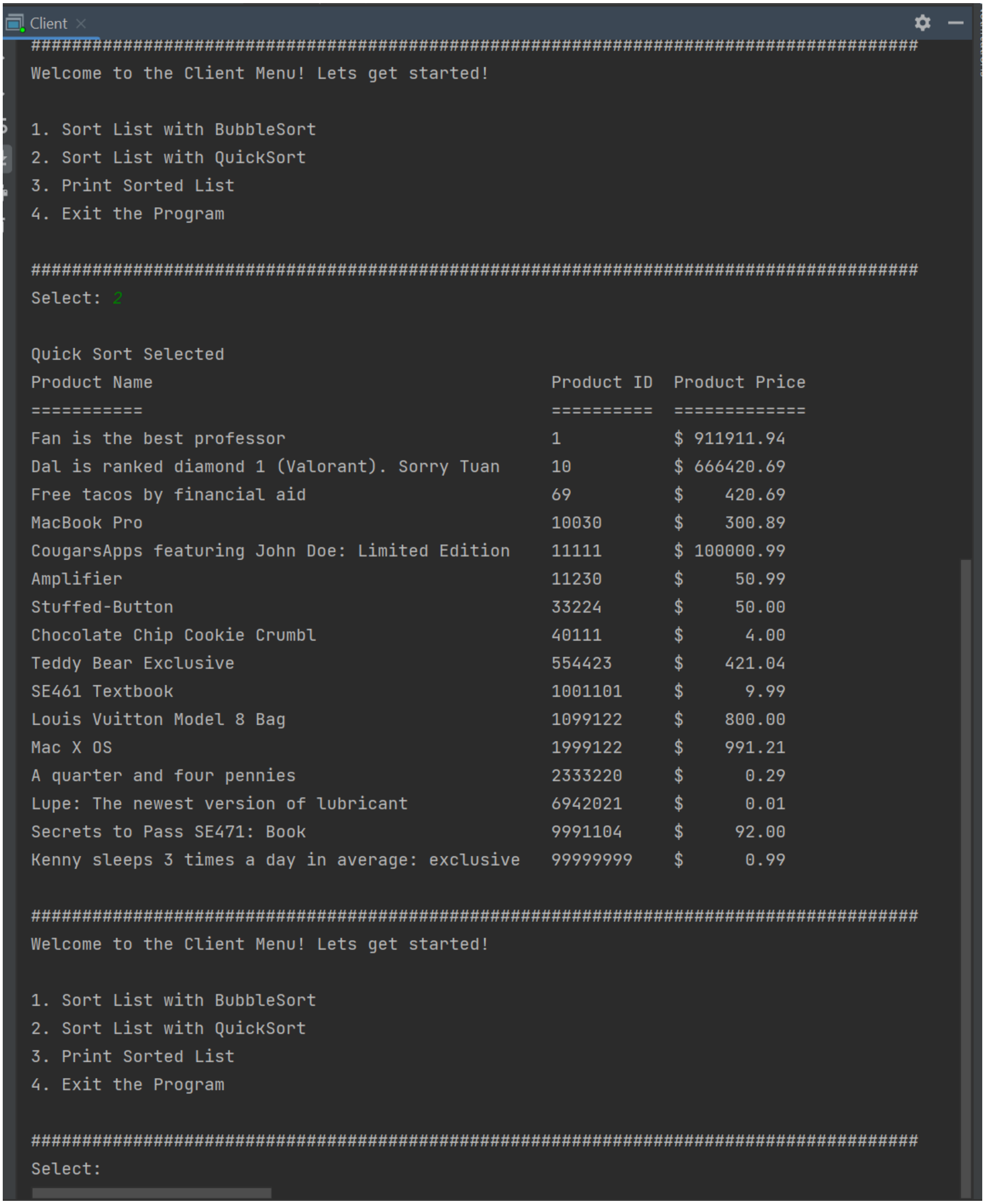
RUN SCREENSHOTS



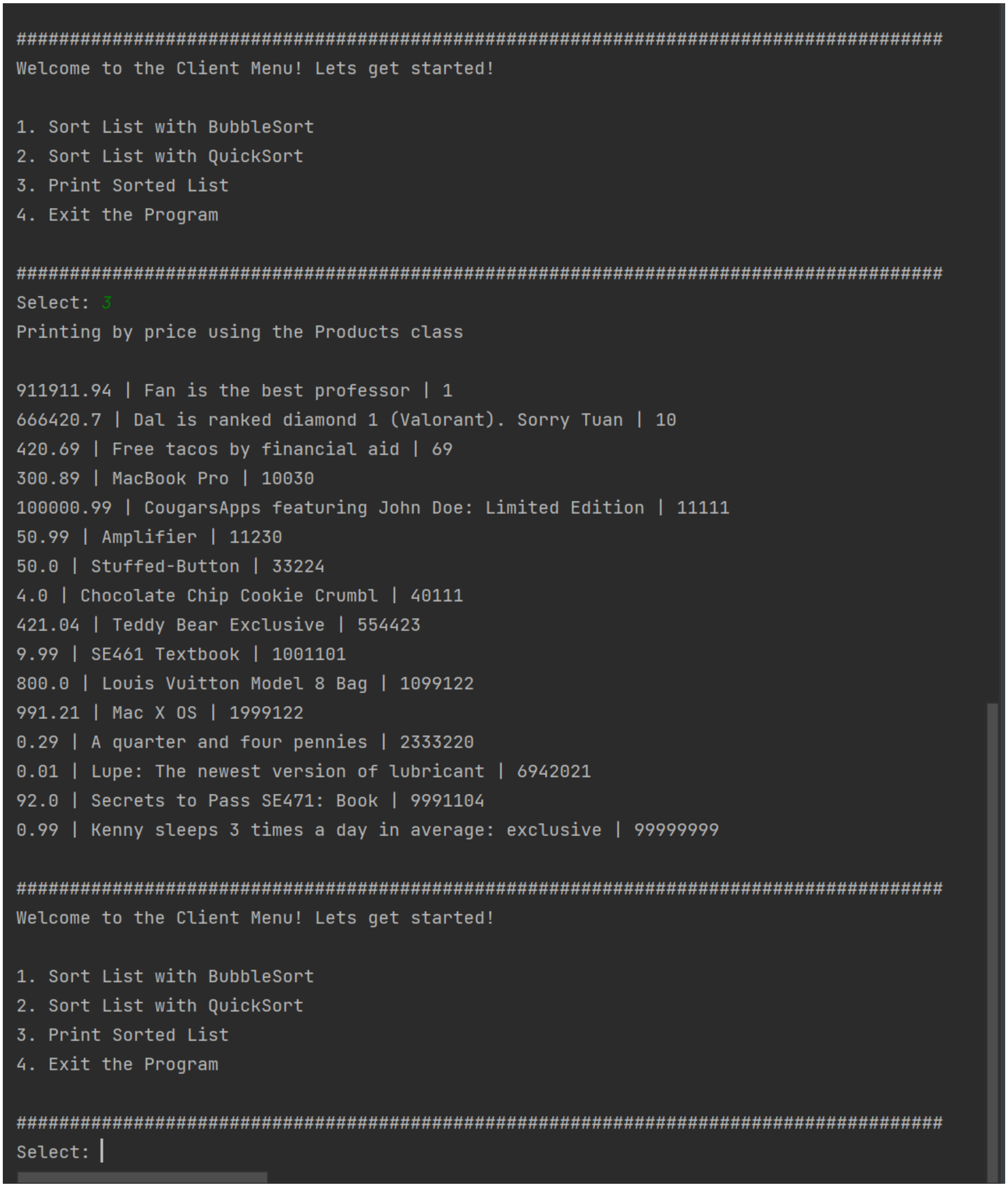
```
C:\Users\mrpri\.jdks\corretto-11.0.18\bin\java.exe "-javaagent:C:\Users\mrpri\Desktop\Se471

#####################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

#####################################################################################
Select: 
```



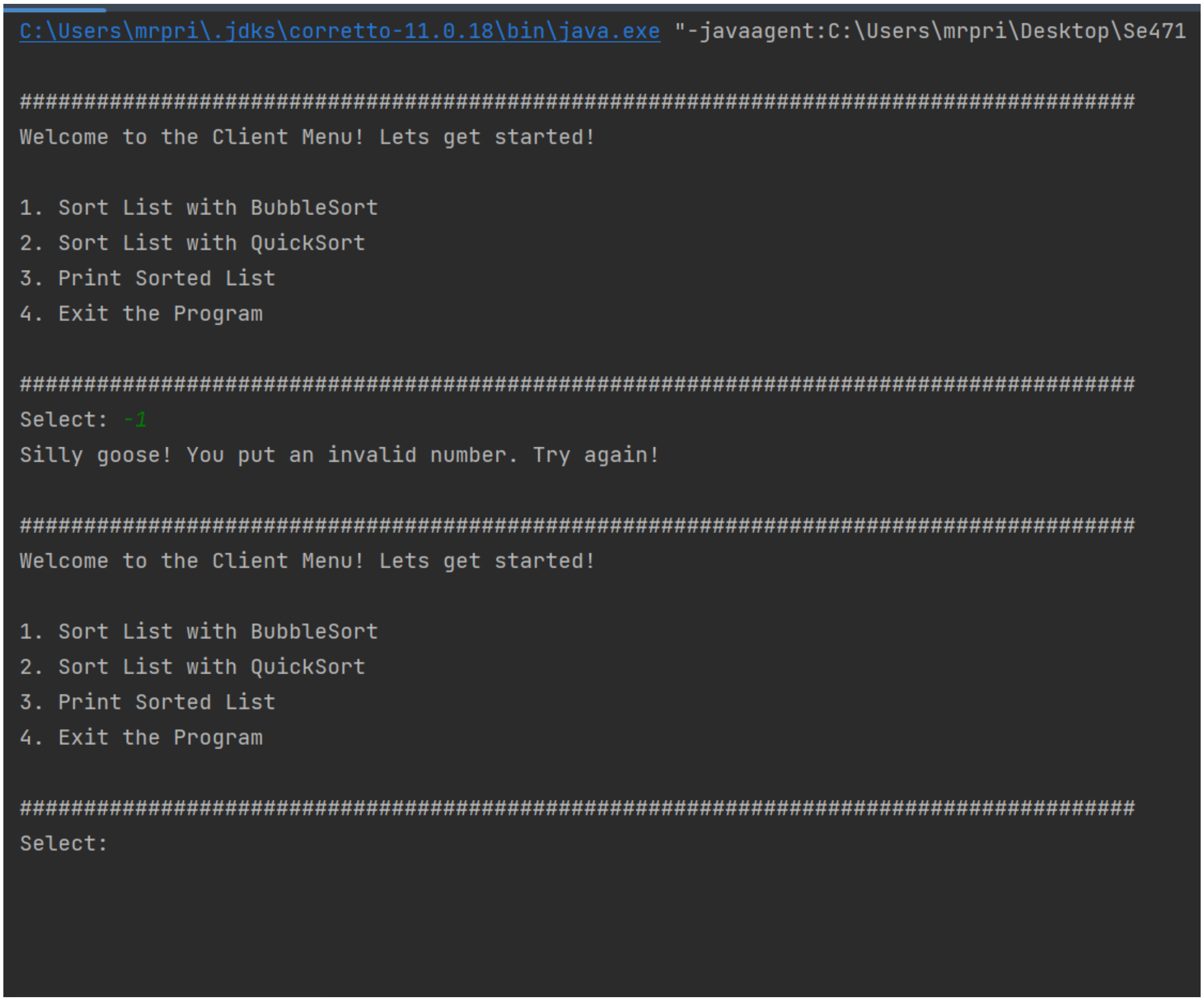
```
########################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

########################################################################################
Select: 1

Bubble Sort Selected
Product ID  Product Name                                       Product Price
=========== =============                                      ============
1           Fan is the best professor                          $ 911911.94
10          Dal is ranked diamond 1 (Valorant). Sorry Tuan     $ 666420.69
69          Free tacos by financial aid                        $    420.69
10030       MacBook Pro                                        $    300.89
11111       CougarsApps featuring John Doe: Limited Edition    $ 100000.99
11230       Amplifier                                          $     50.99
33224       Stuffed-Button                                     $     50.00
40111       Chocolate Chip Cookie Crumbl                       $      4.00
554423      Teddy Bear Exclusive                               $    421.04
1001101     SE461 Textbook                                     $      9.99
1099122     Louis Vuitton Model 8 Bag                          $    800.00
1999122     Mac X OS                                           $    991.21
2333220     A quarter and four pennies                         $      0.29
6942021     Lupe: The newest version of lubricant              $      0.01
9991104     Secrets to Pass SE471: Book                        $     92.00
99999999    Kenny sleeps 3 times a day in average: exclusive   $      0.99

########################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

########################################################################################
Select:
```



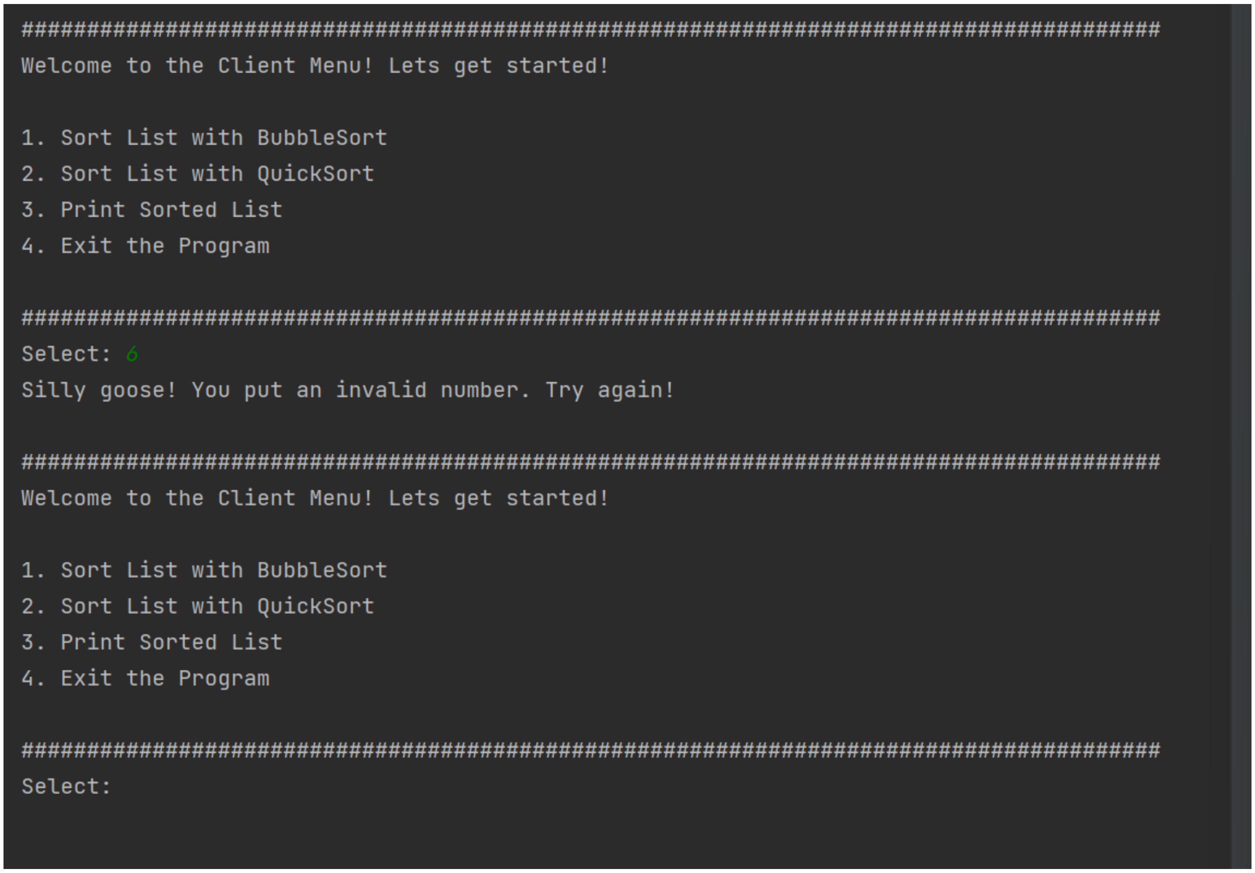
```
#######################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

#######################################################################################
Select: 2

Quick Sort Selected
Product Name                                       Product ID  Product Price
===========                                        ==========  =============
Fan is the best professor                          1           $ 911911.94
Dal is ranked diamond 1 (Valorant). Sorry Tuan     10          $ 666420.69
Free tacos by financial aid                        69          $    420.69
MacBook Pro                                        10030       $    300.89
CougarsApps featuring John Doe: Limited Edition    11111       $ 100000.99
Amplifier                                          11230       $     50.99
Stuffed-Button                                     33224       $     50.00
Chocolate Chip Cookie Crumbl                       40111       $      4.00
Teddy Bear Exclusive                               554423      $    421.04
SE461 Textbook                                     1001101     $      9.99
Louis Vuitton Model 8 Bag                          1099122     $    800.00
Mac X OS                                           1999122     $    991.21
A quarter and four pennies                         2333220     $      0.29
Lupe: The newest version of lubricant              6942021     $      0.01
Secrets to Pass SE471: Book                        9991104     $     92.00
Kenny sleeps 3 times a day in average: exclusive   99999999    $      0.99

#######################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

#######################################################################################
Select:
```



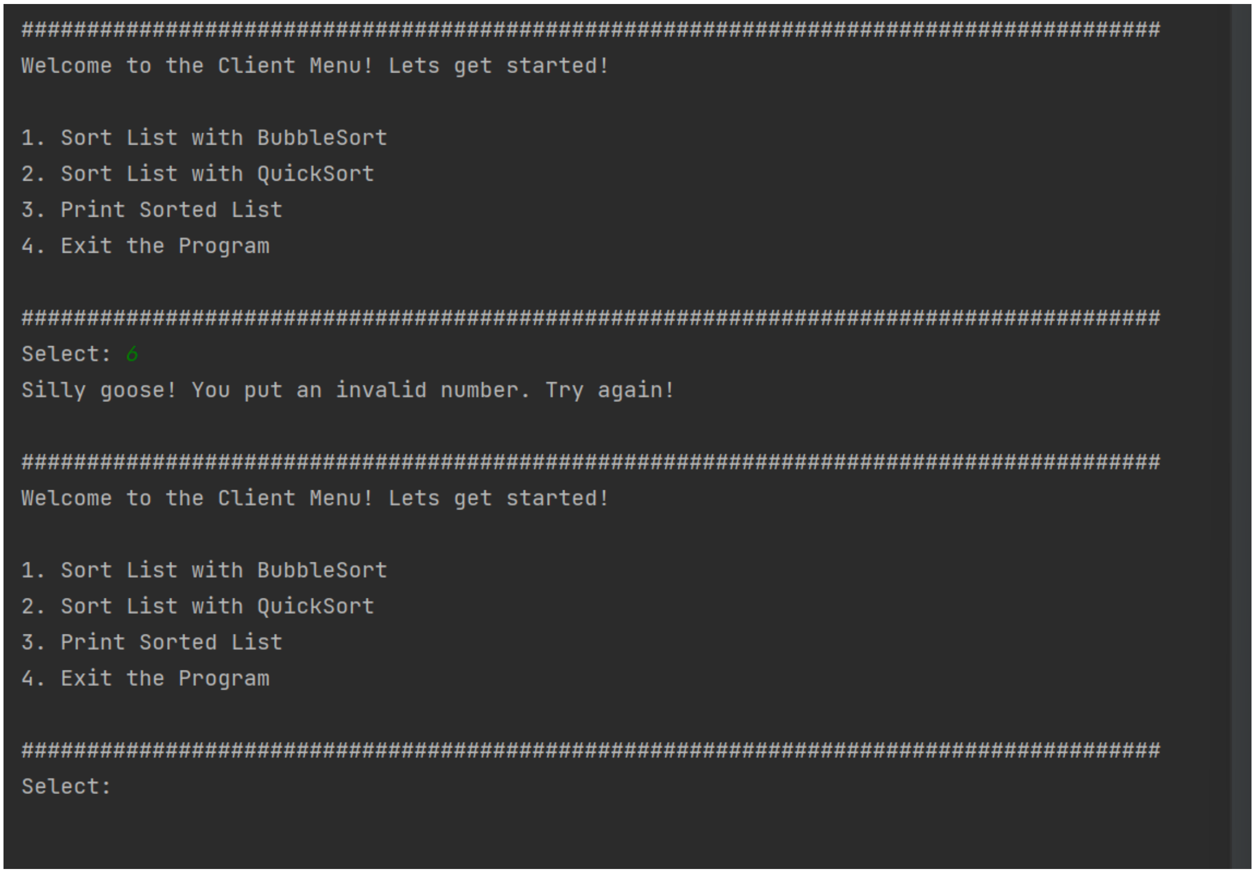
```
########################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

########################################################################################
Select: 3
Printing by price using the Products class

911911.94 | Fan is the best professor | 1
666420.7 | Dal is ranked diamond 1 (Valorant). Sorry Tuan | 10
420.69 | Free tacos by financial aid | 69
300.89 | MacBook Pro | 10030
100000.99 | CougarsApps featuring John Doe: Limited Edition | 11111
50.99 | Amplifier | 11230
50.0 | Stuffed-Button | 33224
4.0 | Chocolate Chip Cookie Crumbl | 40111
421.04 | Teddy Bear Exclusive | 554423
9.99 | SE461 Textbook | 1001101
800.0 | Louis Vuitton Model 8 Bag | 1099122
991.21 | Mac X OS | 1999122
0.29 | A quarter and four pennies | 2333220
0.01 | Lupe: The newest version of lubricant | 6942021
92.0 | Secrets to Pass SE471: Book | 9991104
0.99 | Kenny sleeps 3 times a day in average: exclusive | 99999999

########################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

########################################################################################
Select: 
```



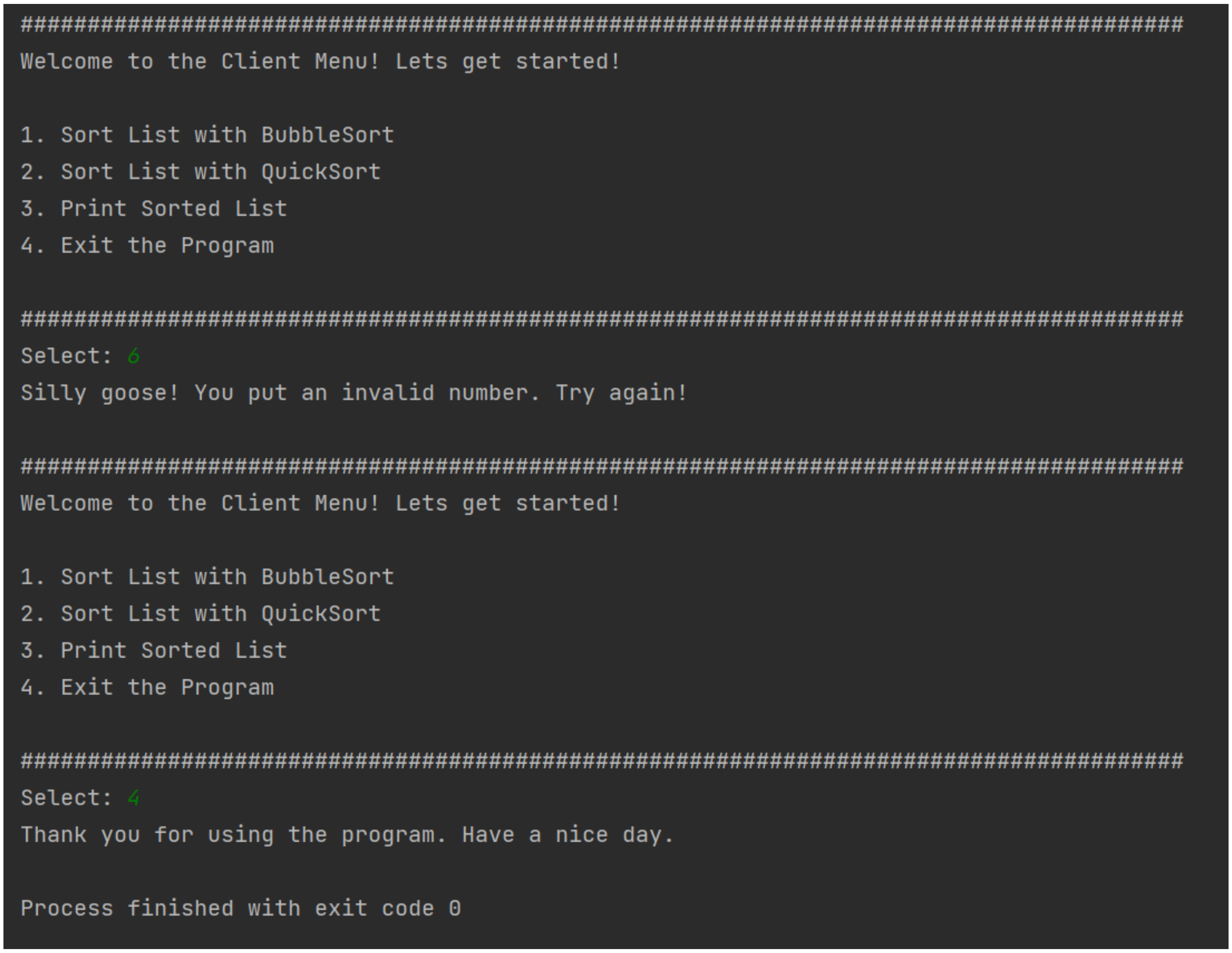
```
C:\Users\mrpri\.jdks\corretto-11.0.18\bin\java.exe "-javaagent:C:\Users\mrpri\Desktop\Se471

#######################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

#######################################################################################
Select: -1
Silly goose! You put an invalid number. Try again!

#######################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

#######################################################################################
Select:
```



```
#######################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

#######################################################################################
Select: 6
Silly goose! You put an invalid number. Try again!

#######################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

#######################################################################################
Select:
```



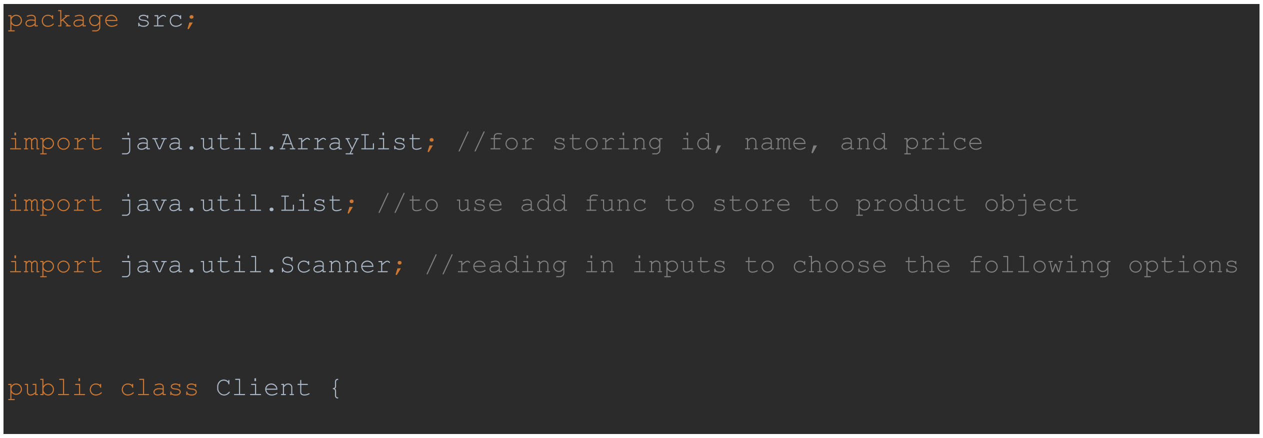
```
#######################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

#######################################################################################
Select: 6
Silly goose! You put an invalid number. Try again!

#######################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

#######################################################################################
Select:
```



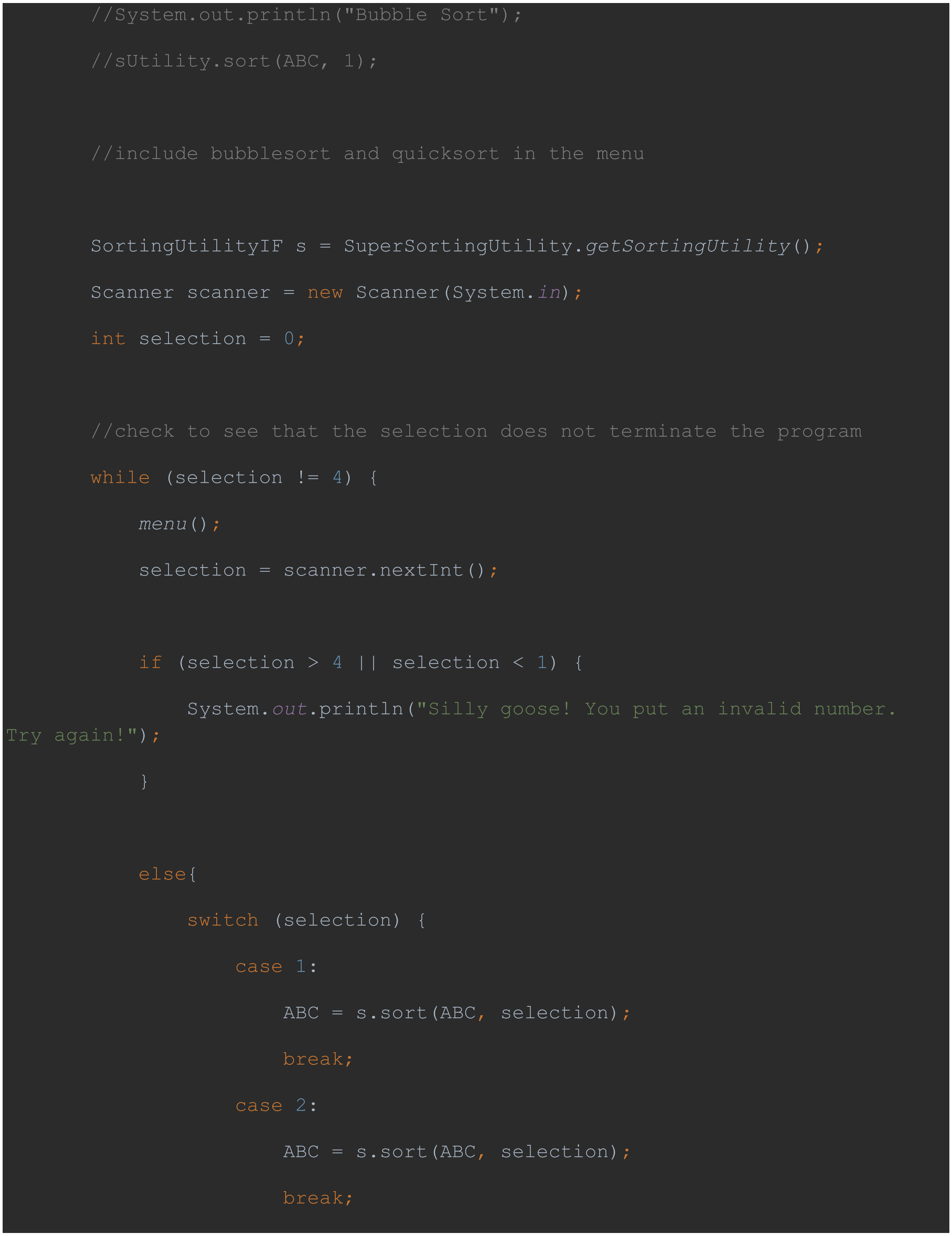
```
########################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

########################################################################################
Select: 6
Silly goose! You put an invalid number. Try again!

########################################################################################
Welcome to the Client Menu! Lets get started!

1. Sort List with BubbleSort
2. Sort List with QuickSort
3. Print Sorted List
4. Exit the Program

########################################################################################
Select: 4
Thank you for using the program. Have a nice day.

Process finished with exit code 0
```

SOURCE CODE:

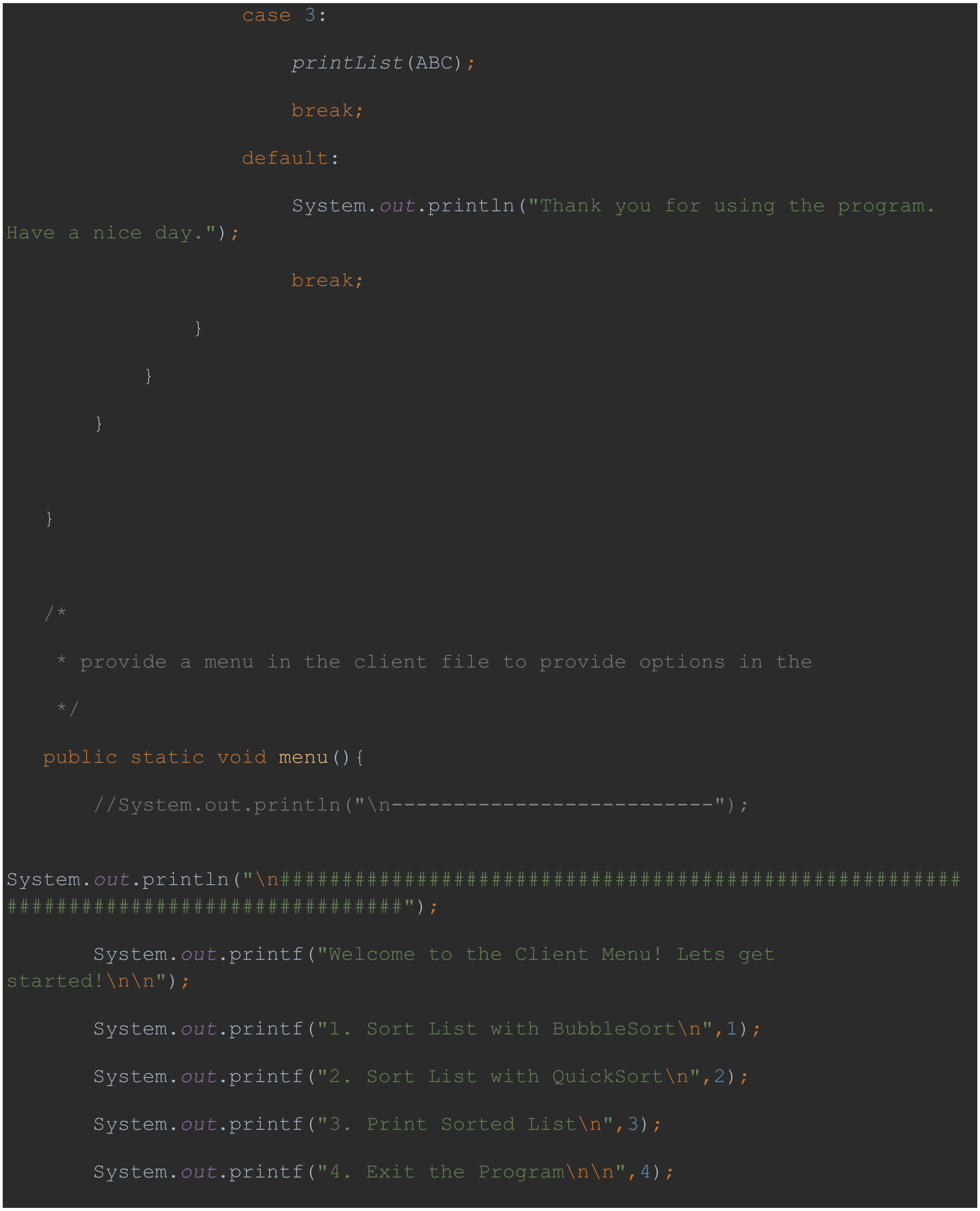
Client



```
package src;


import java.util.ArrayList; //for storing id, name, and price

import java.util.List; //to use add func to store to product object

import java.util.Scanner; //reading in inputs to choose the following options


public class Client {
   public static void main(String[] args) {
       //ABC (ABC company) variable to include attributes within Product
object
       List<Product> ABC = new ArrayList<Product>();
       //SortingUtilityIF sUtility = SuperSortingUtility.getSortingUtility();


       /* Test inputs to use for the program */
       ABC.add(new Product(1001101, "SE461 Textbook", 9.99));
       ABC.add(new Product(9991104, "Secrets to Pass SE471: Book", 92.00));
       ABC.add(new Product(33224, "Stuffed-Button", 50.00));
       ABC.add(new Product(1099122, "Louis Vuitton Model 8 Bag", 800.00));
       ABC.add(new Product(554423, "Teddy Bear Exclusive", 421.04));
       ABC.add(new Product(2333220, "A quarter and four pennies", 0.29));
       ABC.add(new Product(40111, "Chocolate Chip Cookie Crumbl", 4.00));
       ABC.add(new Product(11230, "Amplifier", 50.99));
       ABC.add(new Product(10030, "MacBook Pro", 300.89));
       ABC.add(new Product(1999122, "Mac X OS", 991.21));
       ABC.add(new Product(1, "Fan is the best professor", 911911.91));
       ABC.add(new Product(99999999, "Kenny sleeps 3 times a day in average:
exclusive", 0.99));
       ABC.add(new Product(11111, "CougarsApps featuring John Doe: Limited
Edition", 100000.99));
       ABC.add(new Product(012, "Dal is ranked diamond 1 (Valorant). Sorry
Tuan", 666420.69));
       ABC.add(new Product(69, "Free tacos by financial aid", 420.69));
       ABC.add(new Product(6942021, "Lupe: The newest version of lubricant",
0.01));


       //test case to use without switch statement
```



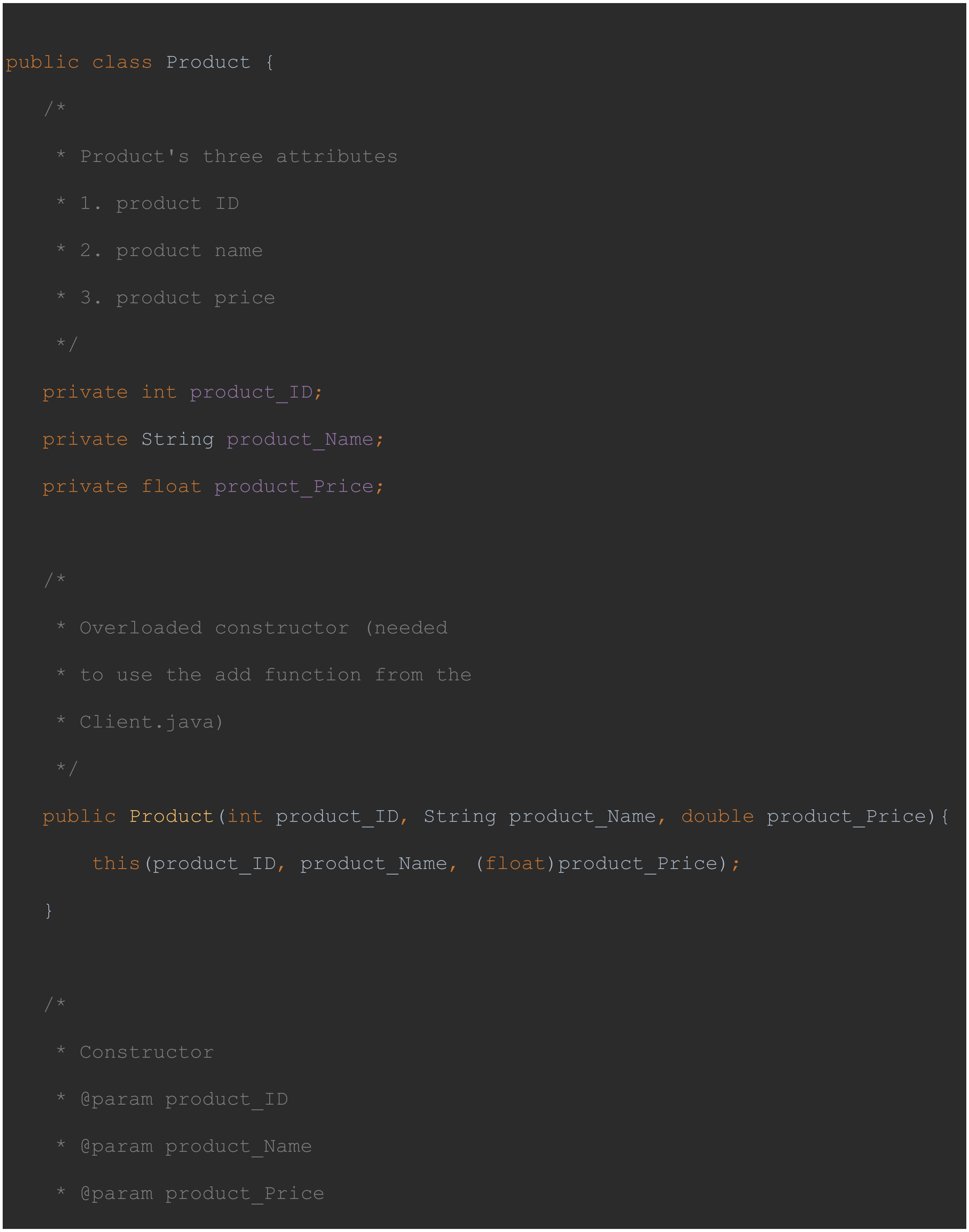
```
        //System.out.println("Bubble Sort");
        //sUtility.sort(ABC, 1);

        //include bubblesort and quicksort in the menu

        SortingUtilityIF s = SuperSortingUtility.getSortingUtility();
        Scanner scanner = new Scanner(System.in);
        int selection = 0;

        //check to see that the selection does not terminate the program
        while (selection != 4) {
            menu();
            selection = scanner.nextInt();

            if (selection > 4 || selection < 1) {
                System.out.println("Silly goose! You put an invalid number. Try again!");
            }

            else{
                switch (selection) {
                    case 1:
                        ABC = s.sort(ABC, selection);
                        break;
                    case 2:
                        ABC = s.sort(ABC, selection);
                        break;
```



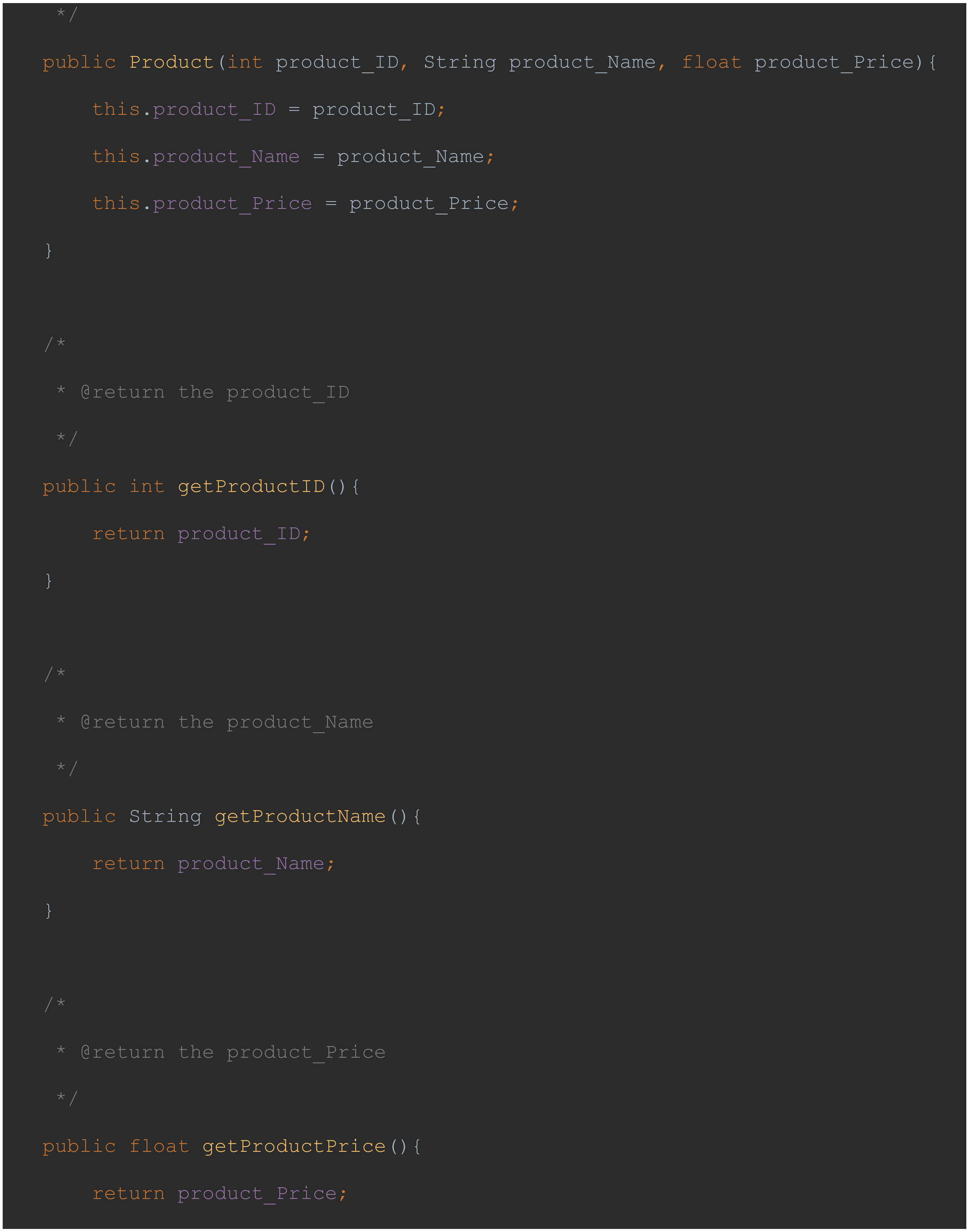
```
                   case 3:
                       printList(ABC);
                       break;
                   default:
                       System.out.println("Thank you for using the program. 
Have a nice day.");
                       break;
               }
           }
       }


   }


   /*
    * provide a menu in the client file to provide options in the
    */
   public static void menu(){
       //System.out.println("\n--------------------------");

System.out.println("\n#######################################################
################################");
       System.out.printf("Welcome to the Client Menu! Lets get 
started!\n\n");
       System.out.printf("1. Sort List with BubbleSort\n",1);
       System.out.printf("2. Sort List with QuickSort\n",2);
       System.out.printf("3. Print Sorted List\n",3);
       System.out.printf("4. Exit the Program\n\n",4);
System.out.println("#########################################################
##############################");

       System.out.print("Select: ");

   }


   /*

    * print function to display the

    * list of products that are

    * collected from the main

    */

   public static void printList(List<Product> l){

       System.out.println("Printing by price using the Products class\n");

       for(Object i : l){

           System.out.println(((Product) i).displayProduct());

       }

   }



}
```

Product

```
package src;
```



```
public class Product {
   /*
    * Product's three attributes
    * 1. product ID
    * 2. product name
    * 3. product price
    */
   private int product_ID;
   private String product_Name;
   private float product_Price;

   /*
    * Overloaded constructor (needed
    * to use the add function from the
    * Client.java)
    */
   public Product(int product_ID, String product_Name, double product_Price){
       this(product_ID, product_Name, (float)product_Price);
   }

   /*
    * Constructor
    * @param product_ID
    * @param product_Name
    * @param product_Price
```

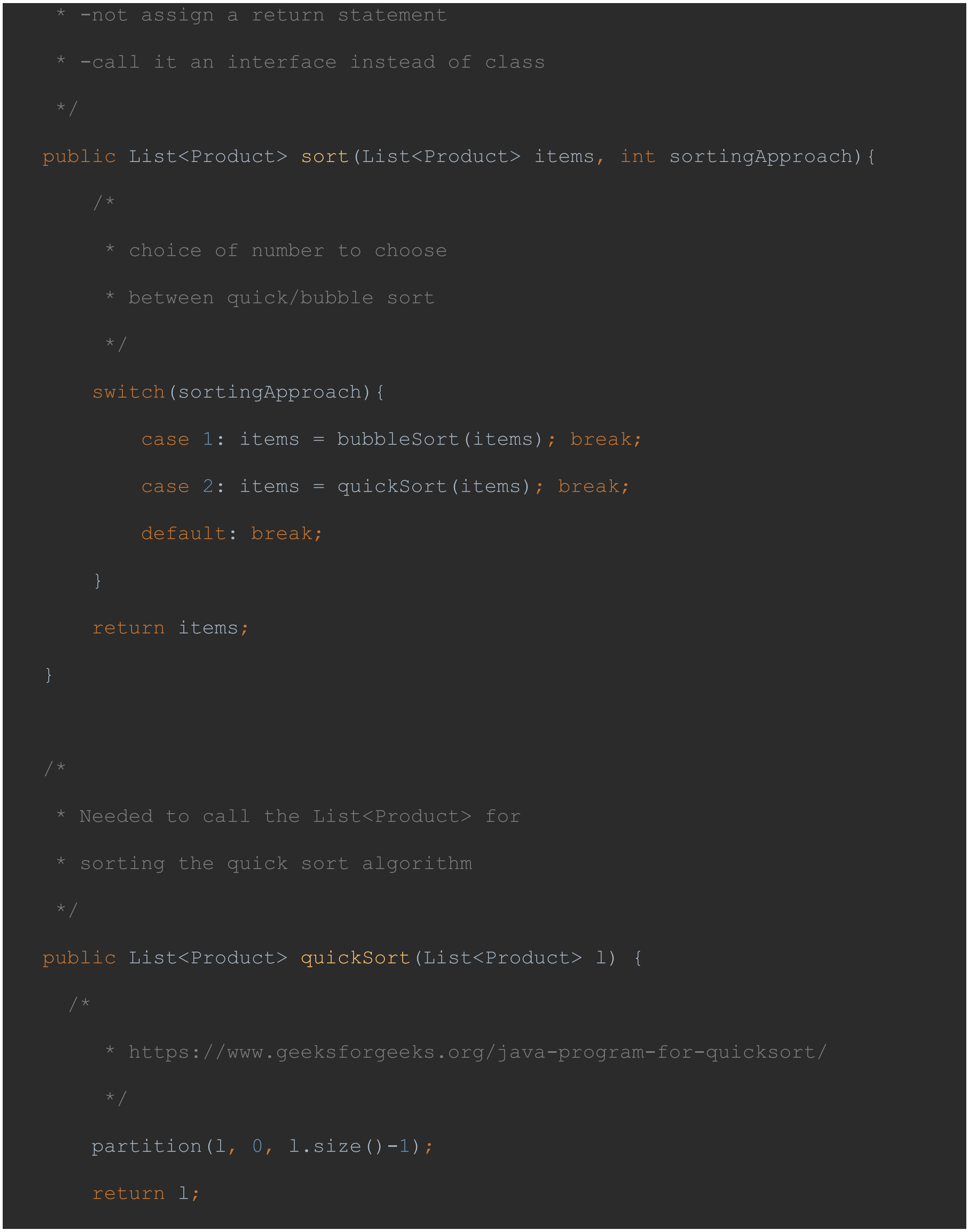


```
 */
public Product(int product_ID, String product_Name, float product_Price){
    this.product_ID = product_ID;
    this.product_Name = product_Name;
    this.product_Price = product_Price;
}

/*
 * @return the product_ID
 */
public int getProductID(){
    return product_ID;
}

/*
 * @return the product_Name
 */
public String getProductName(){
    return product_Name;
}

/*
 * @return the product_Price
 */
public float getProductPrice(){
    return product_Price;
}

/*
 * @param name to set with product_Name
 */
public void setProductName(String name){
    this.product_Name = name;
}

/*
 * @param price to set with product_Price
 */
public void setProductPrice(float price){
    this.product_Price = price;
}

/*
 * display function to output the
 * attributes of the product
 *
 * Use the string class to use function
 * valueOf to convert the following ID
 * and Price into a string
 *
 * @return String of the whole product
 */
```

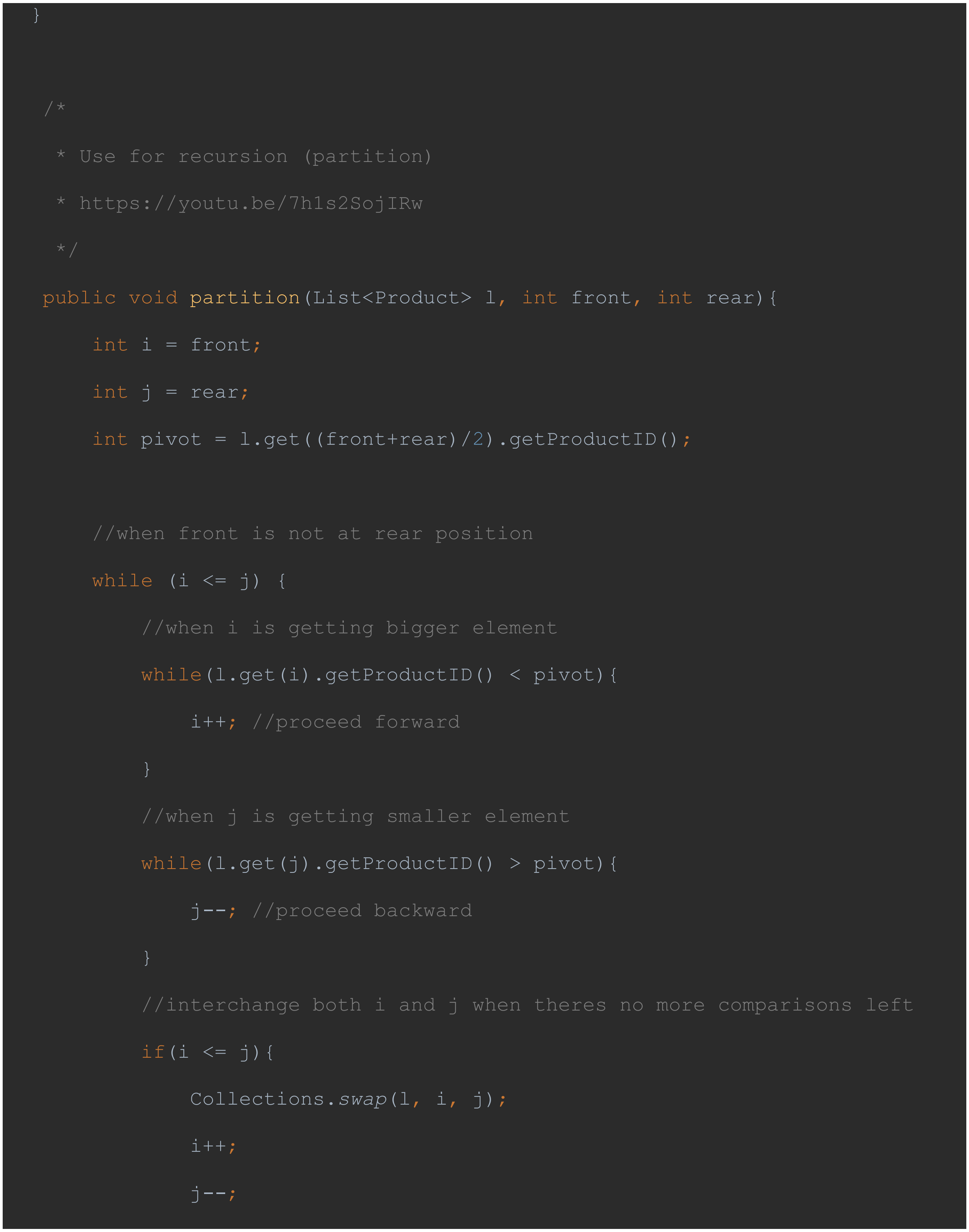
```
    public String displayProduct(){

        return String.valueOf(product_Price) + " | " + product_Name

        + " | " + String.valueOf(product_ID);

        //String.format(); --> trying to set spaces on the outputs

    }

}
```

SortingUtility

```
package src;


import java.util.List;

//import java.util.Collection;

import java.util.Collections;


// Resources gathered

//https://docs.oracle.com/javase/7/docs/api/java/util/List.html -- get()

//https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html --
swap()


public class SortingUtility implements SortingUtilityIF{

    /*

     * calling which sort function

     *

     * challenges/mistakes

     * -forgot to make it a public function
```



```
 * -not assign a return statement
 * -call it an interface instead of class
 */
public List<Product> sort(List<Product> items, int sortingApproach){
    /*
     * choice of number to choose
     * between quick/bubble sort
     */
    switch(sortingApproach){
        case 1: items = bubbleSort(items); break;
        case 2: items = quickSort(items); break;
        default: break;
    }
    return items;
}

/*
 * Needed to call the List<Product> for
 * sorting the quick sort algorithm
 */
public List<Product> quickSort(List<Product> l) {
  /*
     * https://www.geeksforgeeks.org/java-program-for-quicksort/
     */
    partition(l, 0, l.size()-1);
    return l;
```



```
    }


    /*
     * Use for recursion (partition)
     * https://youtu.be/7h1s2SojIRw
     */
    public void partition(List<Product> l, int front, int rear){
        int i = front;
        int j = rear;
        int pivot = l.get((front+rear)/2).getProductID();

        //when front is not at rear position
        while (i <= j) {
            //when i is getting bigger element
            while(l.get(i).getProductID() < pivot){
                i++; //proceed forward
            }
            //when j is getting smaller element
            while(l.get(j).getProductID() > pivot){
                j--; //proceed backward
            }
            //interchange both i and j when theres no more comparisons left
            if(i <= j){
                Collections.swap(l, i, j);
                i++;
                j--;
            }
        }
        //when while condition satisfied, check
        //the updated positions and recursively
        //move to do new comparisons
        if(front < j){
            partition(l, front, j);
        }
        if(i < rear){
            partition(l, i, rear);
        }
    }

    // This is the function that will be used by the proxy.
    /*
     * Products (books, bags, buttons)
     */
    public List<Product> bubbleSort(List<Product> l){
        int max_size = l.size(); // set to the size of list
        /*
         * https://www.geeksforgeeks.org/bubble-sort/
         */
        for(int i = 0; i < max_size - 1; i++){
            for(int j = i+1; j < max_size; j++){
                if(l.get(i).getProductID() > l.get(j).getProductID()){
                //get(i/j) gets the current index of the list
```

```
                    Collections.swap(l, i, j);
                }
            }
        }
        return l;
    }
}
```

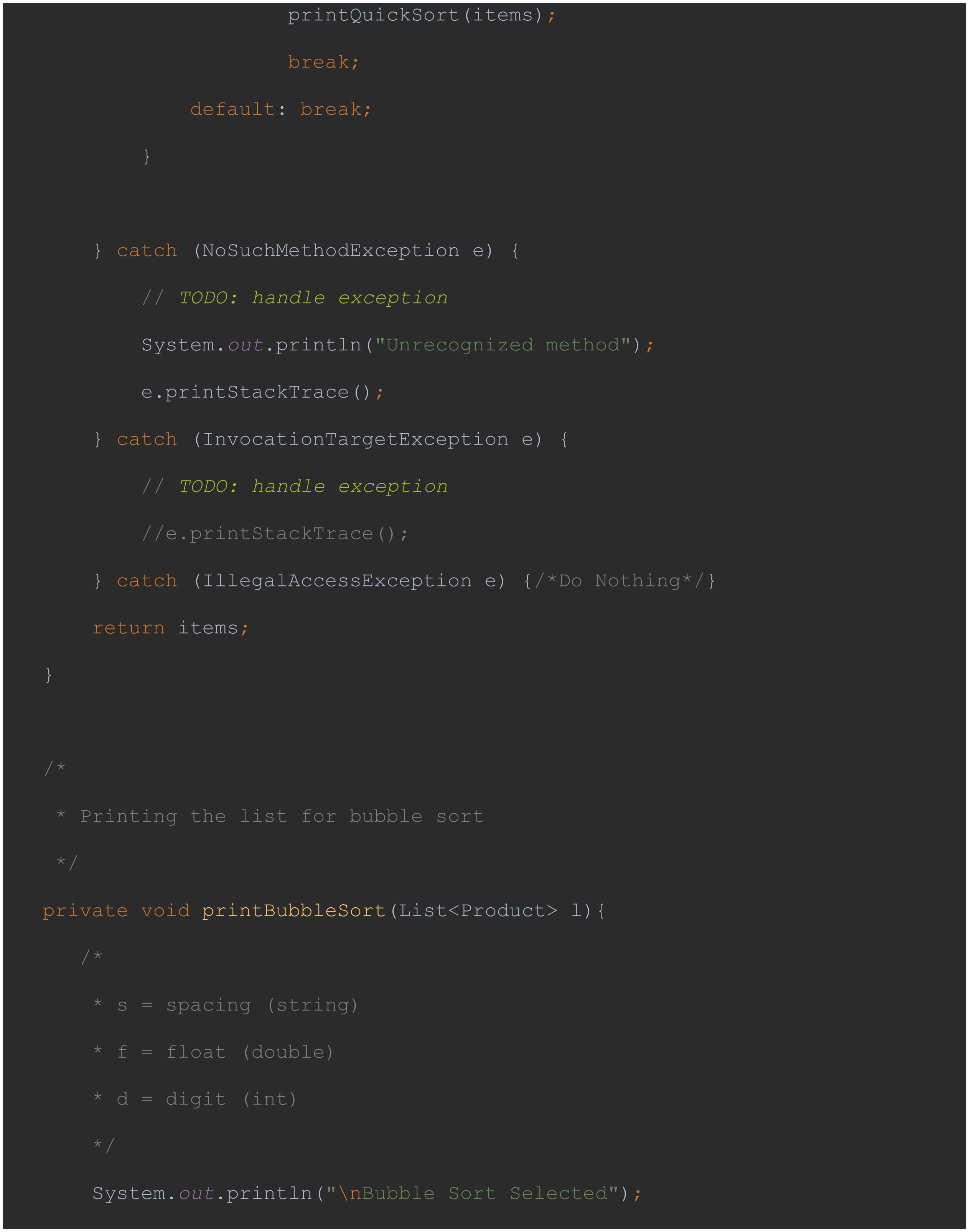
SortingUtilityIF

```
package src;
import java.util.List;
/*
* Interface for sorting purposes
* -NOT A CLASS
*/
public interface SortingUtilityIF {
    public List<Product> sort(List<Product> items, int sortingApproach);
}
```

SortingUtilityProxy

```
package src;


import java.util.List;

import java.lang.reflect.Method;

import java.lang.reflect.InvocationTargetException;


/*

* https://stackoverflow.com/questions/34112276/java-format-string-spacing
(used for outputting statements in quick/bubble sorts)

*/


public class SortingUtilityProxy implements SortingUtilityIF{

   //call the interface object

   private SortingUtilityIF sUtility;


   /*

    * Constructor

    * @param s

    */

   SortingUtilityProxy(SortingUtilityIF s){

       this.sUtility = s;

   }


   /*

    * Getting the sort method

    * @param m to specify method

    * returns the sorted method
```

```
     * throw both exceptions

     */

    private Method sortMethod(String m) throws NoSuchMethodException,
InvocationTargetException{

           Method sortM = sUtility.getClass().getDeclaredMethod(m, List.class);

           sortM.setAccessible(true);

           return sortM;

    }


    // This is where we sort our list of products

    /* (non-Javadoc)

     * @see src.SortingUtilityIF#sort(java.util.List, int)

     */

    /* (non-Javadoc)

     * @see src.SortingUtilityIF#sort(java.util.List, int)

     */

    public List<Product> sort(List<Product> items, int sortingApproach)

    {

           try {

                  switch (sortingApproach) {

                         case 1: /*call bubble sort*/

                                items =
(List<Product>)(this.sortMethod("bubbleSort")).invoke(sUtility, items);

                                printBubbleSort(items);

                                break;

                         case 2: /*call quick sort*/

                                items =
(List<Product>)(this.sortMethod("quickSort")).invoke(sUtility, items);
```



```
                    printQuickSort(items);
                    break;
            default: break;
        }

    } catch (NoSuchMethodException e) {
        // TODO: handle exception
        System.out.println("Unrecognized method");
        e.printStackTrace();
    } catch (InvocationTargetException e) {
        // TODO: handle exception
        //e.printStackTrace();
    } catch (IllegalAccessException e) {/*Do Nothing*/}
    return items;
}

/*
 * Printing the list for bubble sort
 */
private void printBubbleSort(List<Product> l){
   /*
    * s = spacing (string)
    * f = float (double)
    * d = digit (int)
    */
    System.out.println("\nBubble Sort Selected");
```

```
       System.out.printf("%-11s %-50s %-7s\n", "Product ID", "Product Name",
"Product Price");

       System.out.printf("%-11s %-50s %-7s\n", "===========",
"=============", "============");


       for (Product product : l) {

           System.out.printf("%-11d %-50s $%10.2f\n", product.getProductID(),

           product.getProductName(), product.getProductPrice());

       }

   }

   /*

    * Printing the list for quick sort

    */

   private void printQuickSort(List<Product> l){

       /*

       * s = spacing (string)

       * f = float (double)

       */

       System.out.println("\nQuick Sort Selected");

       System.out.printf("%-50s %-11s %-7s\n", "Product Name", "Product ID",
"Product Price");

       System.out.printf("%-50s %-11s %-7s\n", "===========", "==========",
"=============");


       for (Product product : l) {

           System.out.printf("%-50s %-11s $%10.2f\n",
product.getProductName(),

           product.getProductID(), product.getProductPrice());

       }
    }

}
```

SuperSortingUtility

```
package src;

//private SortingUtility


public class SuperSortingUtility {

    public static SortingUtilityIF getSortingUtility(){

        return new SortingUtilityProxy(new SortingUtility());

    }

}
```