Object-oriented programming and design

Lab 2

Shopping Cart Application

Prerequisites, Goals, and Outcomes

Prerequisites: Before you begin this exercise, you need mastery of the following:

- *Java Language*
  - Knowledge of Java control structures
- *Java API*
  - Knowledge of package `java.io`
    - How to get input from the keyboard
    - How to output results, prompts, and errors
  - Knowledge of wrapper class `Integer`: how to read an integer value as input
  - Knowledge of class `StringTokenizer`: how to parse a line with multiple values
- *Exception Objects*
  - Knowledge of exception handling
  - Knowledge of `NumberFormatException`
- *Programming Practice*
  - How to produce Java applications
  - Knowledge of Sun's code conventions
  - Knowledge of Javadoc: how to document classes, methods, and variables

Goals: Reinforce your ability to use I/O classes, tokenizers, wrapper classes, and exceptions

Outcomes: You will demonstrate mastery in the following:

- Produce an application that reads input from the keyboard and uses:
  - Wrapper classes to read numerical data
  - `StringTokenizer` to parse a line with multiple values
  - Exceptions to handle malformed data
  - Control structures to control the reading of data

Background

This assignment asks you to complete an application that adds products to an electronic shopping cart. The application uses three classes: `Product`, `ShoppingCart`, and`ShoppingCartApplication`. Part of the work has been done for you and is provided in the student archive. You will implement the method in `ShoppingCartApplication` that reads product information from the keyboard and creates a `Product` object.

Description

The application presents the user with a menu of options and prompts the user for a choice:

- Choice 0 terminates the program.
- Choice 1 adds a product to the shopping cart.
- Choice 2 displays the information of all the products stored in the shopping cart.
- Choice 3 displays the total cost of all the products in the shopping cart.

To add a product, the user enters a line with the following format:

name_quntity_price

Where:

- *name* is the name of the product.
- *quantity* is the quantity of the product.
- *price* is the price of the product.

The fields are delimited by an underscore (_). If the user's input is invalid, the application displays an error message.

The following is a screen shot of the application after some products have been added.

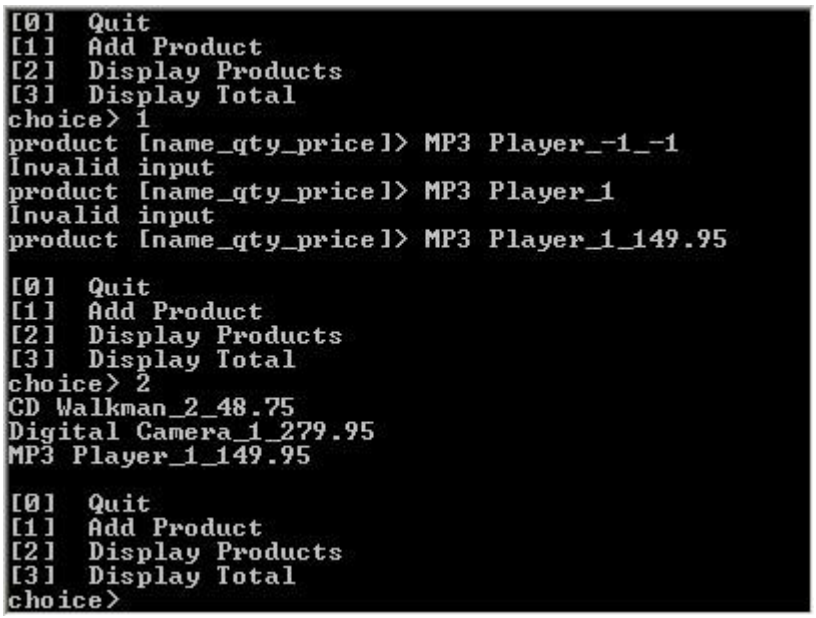

Figure 1 *Execution of ShoppingCartApplication*

The application uses classes `Product` and `ShoppingCart`. `ShoppingCart` maintains a collection of products. Complete implementations of both are provided in the student archive *`student-files.zip`*. Review their documentation and become familiar with it.

- *`Product`*. Documentation for class `Product`
- *`ShoppingCart`*. Documentation for class `ShoppingCart`

A partial implement of `ShoppingCartApplication` is provided in the student archive *`student-files.zip`*. It contains some variables declarations and three methods that need no modification. You should complete method `readProduct`, the method that reads product information from the keyboard and returns a `Product` object.

Files

The following files are needed to complete this assignment:

- *`exe-shopping-cart.jar`*. Download this file. It is the sample executable.
- *`student-files.zip`*. Download this file. This archive contains the following:
  - *`Product.class`*. A complete implementation
  - *`ShoppingCart.class`*. A complete implementation
  - *`Product.html`*. Documentation
  - *`ShoppingCart.html`*. Documentation
  - *`ShoppingCartApplication.java`* — Use this template to complete your implementation.

Tasks

To complete this assignment, you will finish the implementation of class `ShoppingCartApplication`. The following steps will guide you through this assignment. Document using Javadoc and follow Sun's code conventions. Work incrementally and test each increment. Save often.

1. **Extract** the student archive by issuing the following command at the command prompt:

   ```
   C:\>unzip student-files.zip
   ```

2. **Run** the sample executable by issuing the following command at the command prompt:

```
C:\>java -jar exe-shopping-cart.jar
```

Observe how the program responds to the following types of input:

- Input with a quantity that is not a valid integer: `MP3 Player_1.0_150.0`
- Input with a price that is not a valid double: `MP3 Player_1_A`.
- Input that contains negative numbers: `MP3 Player_-1_150.0`, or `MP3 Player_1_-150.0`.
- Input that contains more than three values: `MP3 Player_1_150.0_1`.
- Input that contains fewer than three values: `MP3 Player_1`.
- Valid input: `MP3 Player_1_150.0`, and `CD Walkman_2_48.75`.

2. **Then**, complete method `readProduct`:

- *`private Product readProduct() throws IOException`*. This method prompts the user for input, reads product information from the keyboard, and creates an instance of class `Product`. The product information should consist of three values, all entered on the same line, and delimited by an underscore (_). The first value should be a non-empty string that represents the name of the product. The second value should be a positive integer that represents the quantity of product. The third value should be a positive double that represents the price. Use java.util.StringTokenizer to extract the three values from the input.

`readProduct` validates the user's input:

- If the user enters more than three values, an error message is displayed.
- If the user enters fewer than three values, an error message is displayed.
- If the user enters a quantity that is not a valid integer, java.lang.NumberFormatException is caught and output.
- If the user enters a price that is not a valid double, java.lang.NumberFormatException is caught and output.
- If the user enters a quantity that is negative or zero, an error message is displayed.
- If the user enters a price that is negative, an error message is displayed.

The error messages displayed by your implementation should match the error messages displayed by the sample executable.

If the input is invalid, `readProduct` re-prompts the user for new input. Otherwise, it creates a new `Product` object using the specified name, quantity, and price and returns a reference to the new object to the calling method.

Submission

Upon completion, submit **only** the following:

1. *`ShoppingCartApplication.java`*