P03 Exceptional Shopping Cart

Overview

In this assignment, we are going to implement an enhanced version of the Shopping Cart (P01). You are going to add the following features to our famous program:

- Throw exceptions to report bugs or misuse of a set of methods that we have already implemented in P01.
- Load cart summary to the exceptional shopping cart from a file.
- Save the cart summary of the cart in a file too, with respect to a specific format.
- Develop unit tests to check the correctness of the implementation of the different operations supported by our exceptional shopping cart.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P3 pre-assignment quiz is accessible through Canvas before having access to this specification by **11:59PM on Sunday 02/13/2022**. The pre-assignment quiz contains hints which can help you in the development of this assignment.
20 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident of this, you should run additional tests of your own.
15 points	**Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways.
10 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission. We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope.

Learning Objectives

The goals of this assignment include:

- Learn how to improve the robustness of a program so it can survive unusual circumstances and cope with erroneous input without crashing.
- Develop your understanding of the difference between checked and unchecked exceptions, and get practice both throwing and catching exceptions of each kind.
- Get more practice writing tests, specifically tests that detect whether exceptions are thrown under the prescribed circumstances or not.

Additional Assignment Requirements and Notes

(Please read carefully!)

- Pair programming is **NOT ALLOWED** for this assignment. You MUST complete and submit your p01 INDIVIDUALLY.
- **NO import statements** other than the following are allowed in your `ExceptionalShoppingCart` class.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.NoSuchElementException;
import java.util.zip.DataFormatException;
import java.util.Scanner;
```

- You CAN further import java.util.Arrays in your `ExceptionalShoppingCartTester` class.
- Only your submitted `ExceptionalShoppingCartTester` class contains a **main** method.
- You are NOT allowed to add any constant or variable not defined or provided in this write-up outside of any method.
- You CAN define local variables that you may need to implement the methods defined in this program.
- You CAN define **private static** helper methods to help implement the different public static methods defined in this write-up.

- You MUST NOT add any **public methods** to your `ExceptionalShoppingCart` class other than those defined in this write-up.
- All your test methods should be defined and implemented in your `ExceptionalShoppingCartTester`.
- All your test methods must be **public static**. Also, they must take **zero arguments**, **return a boolean**, and must be defined and implemented in your `ExceptionalShoppingCartTester` class.
- Your tester methods must NOT throw any exception. They should return either with true or false. Make sure to catch any expected or unexpected exceptions, and return false if your tester detects any bug.
- Your `ExceptionalShoppingCartTester` class must implement at least the test methods defined in this write-up with exactly the same signatures. Feel free to add additional test methods and/or consider additional test scenarios to further convince yourself of the correctness of your implementation.
- You are also responsible for maintaining secure back-ups of your progress as you work. The OneDrive and GoogleDrive accounts associated with your UW NetID are often convenient and secure places to store such backups.
- Make sure to submit your code (work in progress) of this assignment on Gradescope both early and often. This will 1) give you time before the deadline to fix any defects that are detected by the tests, 2) provide you with an additional backup of your work, and 3) help track your progress through the implementation of the assignment. These tests are designed to detect and provide feedback about only very specific kinds of defects. **It is your responsibility to implement additional testing to verify that the rest of your code is functioning in accordance with this write-up.**
- You can submit your work in progress multiple times on gradescope. Your submission may include methods not implemented or with partial implementation or with a default return statement. But avoid submitting a code which does not compile. A submission which contains compile errors won't pass any of the automated tests on gradescope.
- Feel free to **reuse** any of the provided source code in this write-up verbatim in your own submission.
- All implemented methods (either public or private) including the main and tester methods MUST have their own javadoc-style method headers, with accordance to the CS300 Course Style Guide.
- All your classes MUST have a javadoc-style class header.
- You MUST adhere to the Academic Conduct Expectations and Advice

1 Getting Started

Start by creating a new Java Project in eclipse called P03 Exceptional Shopping Cart, for instance. You MUST ensure that your new project uses Java 11, by setting the "Use an execution environment JRE:" drop down setting to "JavaSE-11" within the new Java Project dialog box. DO NOT create any module within to your java project. Then, create two Java classes/source files within that project's src folder (both inside the **default** source package) called `ExceptionalShoppingCart` and `ExceptionalShoppingCartTester`.

- ONLY the `ExceptionalShoppingCartTester` class should include a main method.
- DO NOT generate or add a main method to the `ExceptionalShoppingCart` class.
- Next, add the following tester method to your `ExceptionalShoppingCartTester` class.

```
public static boolean runAllTests(){}
```

  Your `runAllTests()` must call all the tester methods that you are going to implement in your tester class, return true if all tests pass, and false if any of your tests fails.

2 Complete the implementation of ExceptionalShoppingCart class

In this assignment, we are going to provide you with a starting code for the `ExceptionalShoppingCart` class that you are going to complete. Start by reading this code. You should be familiar now with the shopping cart application. Notice that there are additional methods not provided in P01 added to this project, for instance `indexOfInsertionPos()`, `addItemToMarketCatalog()`, `saveCartSummary()`, `parseCartSummaryLine()` and `loadCartSummary()` methods.

Copy the provided code into your `ExceptionalShoppingCart` class before proceeding to the next steps.

2.1 Update the lookupProductById(), and lookupProductByName() methods

Start by updating the implementation details of the instance methods `lookupProductById()` and `lookupProductByName()` according to their detailed javadocs description provided within these javadocs.

These methods are expected to work as described in P01 when they terminate without errors. Read carefully the provided javadoc method headers, and pay close attention to the exceptions that should be thrown by the constructor and the public methods.

Add appropriate `@throws` annotations to the javadoc style method header of `lookupProductById()`, and `lookupProductByName()` methods. Keep in mind to NOT catch an exception within the implementation details of a method if that exception has been declared to be thrown using `@throws` annotation in its javadoc method header. Let the exception propagate to the method's caller.

```
/**
 * Returns a string representation of the item whose id is provided as input
 *
 * @param key id of the item to find
 * @return "itemId name itemPrice" if an item with the provided name was found
 * @throws IllegalArgumentException with descriptive error message if key is not
 * a 4-digits int
 * @throws NoSuchElementException with descriptive error message if no match found
 */
public static String lookupProductById(int key){
}

/**
 * Returns a string representation of the item whose name is provided as input
 *
 * @param name name of the item to find
 * @return "itemId name itemPrice" if an item with the provided name was found
 * @throws NoSuchElementException with descriptive error message if no match found
 */
public static String lookupProductByName(String name){
}
```

To check the correctness of your `ExceptionalShoppingCart` class so far, add and implement a tester method called `testLookupMethods()` in your `ExceptionalShoppingCartTester` class.

None of your unit test methods should ever throw any exception. Your test method should catch any exceptions that may be thrown by the call of the constructor or the methods defined in the `ExceptionalShoppingCart` class, and returns true if the expected behavior has been satisfied, and false otherwise. A typical template of tester method involving exception handling was provided in Question 5 of the p03 pre-assignment quiz.

2.2 Implement and test addItemToMarketCatalog() method

The `addItemToMarketCatalog()` is one of the new methods that we are going to add to this expanded version of the shopping cart project. This method takes three Strings as input parameters which represents an item to be added the exceptional shopping cart.

You have to complete the implementation of this method with accordance to the details provided in its javadoc style method header comments available in these javadocs.

- If correctly formatted, the id String provided to addItemToMarketCatalog() method should be parsable to a 4-digits int.
- The name String provided to addItemToMarketCatalog() method should not be null or empty String.
- The price String provided to addItemToMarketCatalog() method should be starting with $ and the number part be parsable to a positive double.

Do not forget to implement the test methods related to check the correctness of the `addItemToMarketCatalog()` defined in the `ExceptionalShoppingCart` class. Feel free to name your tester methods at your convenience. Recall that all your tester methods must be public static return boolean and does not take any input.

2.3 Update the remaining methods which need to throw exceptions

Now, update the `getProductPrice()`, `addItemToCart()`, `nbOccurrences()`, `contains()`, `removeItem()`, `emptyCart()`, `checkout()`, and `getCartSummary()` methods according to the javadocs. Do not forget to update the javadoc method headers of your methods to indicate which exceptions are thrown without being caught and in which cases.

2.4 Implement and Test the saveCartSummary(), parseCartSummaryLine() and loadCartSummary() methods

Now, implement the following methods.

- `saveCartSummary()`, `parseCartSummaryLine()` and `loadCartSummary()` methods in your `ExceptionalShoppingCart` class with accordance to the details provided in these javadocs.
- Their test methods named `testSaveCartSummary()` and `testLoadCartSummary()` in your `ExceptionalShoppingCartTester` class.

Note that `saveCartSummary()` and `loadCartSummary()` methods take a reference to an object of type java.io.File as an input parameter. In your test methods which may call these methods, create the file object using the constructor of the java.io.File class that takes a path name String as input parameter.

- Your *loadCartSummary()* method should be able to load a file saved by your *saveCartSummary()* method.
- Your *loadCartSummary()* method should skip any mal-formatted line in the provided file. A correctly formatted line is as follows:

```
"( " + positiveInteger + " ) " + marketItemName
```

The `positiveInteger` represents the number of occurrences of the market item. It must be > 0 and $<= 10$.
You can disregard extra whitespaces at the beginning and end of each line.
We consider one space as delimiter for tokens in each line.
String.trim() and String.split() methods can help you parsing each line in the file to load.

- Your *saveCartSummary()* method should not throw any exception. If an IOException may be thrown in the *saveCartSummary()*, you have to catch it.

```
/**
 * Save the cart summary to a file
 *
 * @param cart an array of strings which contains the names of items in the cart
 * @param size the number of items in the cart
 * @param file the file to save the cart summary
 * @throws IllegalArgumentException with descriptive error message if size is less than zero
 */
public static void saveCartSummary(String[] cart, int size, File file){
}

/**
 * Parse one line of cart summary and add nbOccurrences of item to cart
 * correct formatting for line:"( " + nbOccurrences + " ) " + itemName
 * delimiter: one space (multiple spaces: wrong formatting)
 *
 * @param line a line of the cart summart to be parsed into one item to be added
 * @param cart an array of strings which contains the names of items in the cart
 * @param size the number of items in the cart
 * @throws DataFormatException with descriptive error message if wrong formatting
 *         (including nbOccurrences not parsable to a positive integer less or equal to 10)
 * @throws IllegalArgumentException with descriptive error message if itemName
 *         not found in marketItems
 * @throws IllegalStateException with descriptive error message if cart reaches
 *         its capacity
 */
protected static int parseCartSummaryLine(String line, String[] cart, int size){
}
```

```java
/**
 * Load the cart summary from the file. For each line of summary,
 * add nbOccurrences of item to cart. Must call parseCartSummaryLine to operate
 *
 * @param file file to load the cart summary from
 * @param cart an array of strings which contains the names of items in the cart
 * @param size the number of items in the cart
 * @return Returns the size of the cart after adding items to the cart
 * @throws IllegalArgumentException with descriptive error message if size is
 *         less than zero
 * @throws IllegalStateException with descriptive error message if cart reaches
 *         its capacity
 */
public static int loadCartSummary(File file, String[] cart, int size){
}
```

3 Assignment Submission

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the CS300 Course Style Guide, you should submit your final work through Gradescope. The only TWO files that you must submit include: `ExceptionalShoppingCart.java` and `ExceptionalShoppingCartTester.java` . Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline. Finally, the third portion of your grade for your submission will be determined by humans looking for organization, clarity, commenting, and adherence to the CS300 Course Style Guide.