CS1331 Homework 5 - Online Shopping

Download zip (/spring2017/hw5/hw5-shopping.zip)

Table of Contents

Introduction

In 2016 Kylie Jenner not only “realized some stuff” (like the difference between a chicken and a pig (http://www.refinery29.com/2016/10/127271/kylie-jenner-pig-chicken-vine)) but also gained major success with the launch of her new cosmetics brand, Kylie Cosmetics. However, not every restock of her products has gone smoothly, and at times the traffic surging her site has even caused Google Analytics to crash (http://www.seventeen.com/beauty/celeb-beauty/news/a39194/kylie-jenners-most-recent-lip-kit-restock-literally-broke-google/). This is where you come in. In a couple of weeks, Kylie is launching a collection of limited edition CS1331-inspired products as well as restocking some of her customers’ personal favorites, and she needs your help to ensure that her website can handle the amount of number of orders that will be coming in. Those limited edition products are going to be flying off the shelves!

Problem Description

You will be writing classes that represent an online shopping system. You will be implementing your own ArrayList, Iterator, and custom exceptions.

Background

This section contains some helpful information about material that appears on this homework. If you are more experienced with Java, you may already know everything here. However, we still suggest reading this section to make sure all the techniques are clear.

ArrayList and Generics

By now you are already pretty familiar with arrays in Java. Arrays are great for storing data, but, once they are created, they cannot grow. Thus, you have to know in advance how many elements the array will need to hold. An array list is a data structure, a object that stores and organizes data, that is created with an initial size, but, when the size is exceeded, the array list expands in capacity to accomodate the new elements.

Java has its own implementation of an array list called ArrayList which has methods like `boolean add (Object o)` which adds an object to the list, `Object get(int index)` which return the object at the passed in index in the list and `void clear()` which removes all the elements from this list. However, in this assignment, we will be implementing our own version of an array list. This means that we will **NOT** be using Java's ArrayList class, so you should never be importing `java.util.ArrayList`.

Our array list called MyArrayList will have a generic type `E`. When we define a class with a generic type, we use `<>`.

```
public class Box<E>
```

Since we are not defining a concrete type of the array list, we are letting Java replace the generic types with concrete types at compile time. Our generic class can have methods utilising its generic type. For example, we could write a method that puts something in Box and another which removes something from the box:

```
public boolean put(E o) {
        // implementation
}

public E remove() {
        // implementation
}
```

In this example we are passing in an element of type `E` for put and returning an element of type `E` for remove. Now we can create a Box of any type.

Whenever we declare a new instance of Box, we can specify a new type. To define the concrete type of a generic class, we use `<>` again. Thus, if we wanted to create a Box which stores Book objects, we would instantiate it like this:

```
Box<Book> books = new Box<>();
```

However, if we then wanted to have a Box which stores Candy objects, we can instantiate it in much the same way without changing any of the code for Box.

```
Box<Candy> candies = new Box<>():
```

Since the compiler replaces all occurences of `E` with the concrete type, we do not have to cast the return type of any of our methods:

```
\\ Not necessary
Book b = (Book) books.remove();

\\ The return type is already a Book
\\ since we defined books to be of the concrete type Book.
\\ Thus, we don't need to cast the return value.
Book b2 = books.remove();
```

Iterable and Iterator

To cycle through its elements using a `for-each` statement, the collection must implement Iterable (https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html) which has one required method that returns an iterator, an object that implements Iterator (https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html). Iterator has two methods which we must implement: `boolean hasNext()` and `E next()`. These methods allow us to use the `for-each` statement on our object.

For this assignment ArrayListInterface implements Iterable. Thus, we must make MyArrayList also implement Iterable and define our own MyArrayListIterator as a private inner class inside MyArrayList. Private inner classes are declared within the main class and have their own methods and instance data which can be accessed by the main class.

Exceptions

An exception is a problem that arises during the execution of a program. When an exception occurs, the normal flow of the program is interupted. To avoid the termination of our program, we must handle these exceptions. You have likely already seen examples of exceptions. If you have ever tried to call a method on a instance of an object with a null value, a NullPointerException arises. If you think back to Homework 1, both `main` and `processGradesFromFile` had the phrase `throws Exception` in their method headers. In this case the exception that could have been thrown was a FileNotFoundException if the csv file that needed to be opened and scanned could not be found. All exceptions are subclasses of Exception which is a subclass of Throwable. All objects that are instances of Throwable or one of its subclasses can be thrown by the JVM or by the `throw` keyword.

Checked Exceptions

A checked exception is an exception that occurs at compile time. These exceptions cannot be ignored and must be handled by the programmer. They must be caught or declared to be thrown.

A method catches an exception using a try-catch block which has the format:

```
try {
        // Code that may throw an exception
} catch (ExceptionName e) (
        // Code that handles this exception
} finally { // optional
        // Code that always executes
}
```

A try block can be followed by multiple catch blocks for different exceptions and can end with a finally block which contains code that will always be executed regardless of if an exception occurs.

The method that actually throws the exception must declare it in its method header with the `throws` keyword and utilize the `throw` keyword when invoking the exception.

```
public void myMethod() throws MyException {
        throw new MyException();
}
```

The responsibility to handle this exception is passed to the method that called this method where either that method will catch the exception or declare it.

In this assignment, you will be implementing your own checked exceptions. An exception can be created by extending Exception or extending a subclasses of Exception. Check out the API for Exception (https://docs.oracle.com/javase/8/docs/api/java/lang/Exception.html) and Throwable (https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html) for constructors and methods that your classes can reuse.

Unchecked Exceptions

An unchecked exception is an exception that occurs at runtime and does not have to be caught or declared. All unchecked exceptions extend RuntimeException (https://docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html), a subclass of Exception.

In this assignment, the methods in your array list will throw certain unchecked exceptions if the user passes in invalid data.

Solution Description

Make sure that you read this whole document before starting! There are many components to this homework assignment, so you may need to read over the solution description several times. Also take a look at the summaries above if you skipped over them and need more help.

ArrayListInterface

This interface is already written for you.

This interface has the following instance fields:

- `INITIAL_CAPACITY` which is an int that represents the initial capacity of the array list.

This class has the following public methods:

- `void add(E e)` which adds the passed-in element to the last position in the array list and throws an IllegalArgumentException if the passed-in element is null
- `E removeAll(E e)` which removes **ALL** instances of the passed-in element from the array list and then returns the element that was removed or null if the element was not contained in the array list. Otherwise, throws an IllegalArgumentException if the passed-in element is null.
- `E remove(int index)` which removes the element at the passed-in index. Otherwise, throws an IndexOutOfBoundsException if the index is less than zero or greater than or equaled to the number of elements of the array list.
- `E get(int index)` which returns the element at the passed-in index of the array list but does **NOT** remove the element from the array list. Otherwise, throws an IndexOutOfBoundsException if the index is less than zero or greater than or equaled to the number of elements of the array list.
- `void clear()` which removes all elements from the array list and sets the array list back to its original capacity.
- `int size()` which returns the number of elements contained in the array list.
- `boolean isEmpty()` which returns false if the array list contains one or more elements and returns true if the array list contains no elements.

MyArrayList.java

Represents a custom array list with generics. We will use this class to store instances of Product for a customer's order, but it can take in any data type. This class implements ArrayListInterface. You should reuse as much code as possible when implementing the methods in ArrayListInterface.

NOTE: ArrayListInterface implements Iterable. Thus, MyArrayList should implement Iterable and define its own MyArrayListIterator as a private inner class.

This class has the following private fields:

- `elements` which is an array of generic type `E`
- `numElements` which is an int that represents the number of elements in `elements`

This class has the following constructors:

- One that takes in no arguments and sets the instance fields.

CheckoutPage.java

Represents the checkout page where the order will be processed.

This class has the following private fields:

- `shoppingCart` which is an array list of products of the customer's order.
- `itemsInStock` which is an array list of products that represents the items that are currently available

for purchase.

This class has the following constructors:

- One that takes in no arguments and sets the instance fields.

This class has the following public methods:

- `double getSubtotal()` which sums up the prices of all of the products in the shopping cart.
- `double getShipping()` which calculates the cost of shipping. Shipping is $8.95, but, if the total of the shopping cart is greater than $50 or the shopping cart is empty, then shipping is $0.
- `double getTotal()` which calculates the sum of the subtotal and the shipping.
- `void addToCart(Product product)` which attempts to add a product to the shopping cart. It checks the availability of the product if the product is limited edition. If the product is available, then it is added to the shopping cart, and the user is notified of the successful addition. Otherwise, if a ProductOutOfStockException occurs, then print out its corresponding message and remove the item *which caused the exception* from the stock.
- `void payForCart(CreditCard card)` which attempts to pay for the order. If the shopping cart is not empty, then the site traffic is checked, and payment is attempted. If the payment is successful, then the shopping cart is cleared, and the user is notified of the successful payment. Otherwise, if a SiteOverloadException or an InsufficientFundsException is thrown, then print out its corresponding message.
- `void removeFromCart(int position)` which removes an item from the shopping cart at a given position and notifies the user of the successful removal.
- `String toString()` which returns a String representation of the CheckoutPage and has already been implemented for you.
- getter methods for both instance fields.

CreditCard.java

Represents a credit card that can be used to pay for an order.

This class has the following private fields:

- `balance` which is a double that represents the amount of money currently available to spend on the credit card.

The class has the following constructors:

- One that takes in a double for the card balance and sets its corresponding instance field.

The class has the following public methods:

- `void pay(double amount) throws InsufficientFundsException` which substracts amount from balance if amount is less than or equaled to balance or throws an InsufficientFundsException if subtracting the amount from balance would cause balance to be negative.

Product.java

Represents a Kylie Cosmetics product. This class cannot be instantiated.

This class has the following private fields:

- `color` which is a String that represents of the color of the product.
- `isLimitedEdition` which represents whether or not the product is a part of the limited edition CS1331-inspired collection.
- `price` which is a double that represents the price of the product

The class has the following constructors:

- One that takes in a String for color, a boolean for whether or not the product is limited edition, and a double for cost and sets their corresponding instance fields.

The class has the following public methods:

- getter methods for all instance variables.
- `String toString()` which properly overrides Object's toString method and returns a String in the format `Limited edition color` if the product is limited edition or just `color` if it is not.
- an equals method which checks for equality based on color, price, and isLimitedEdition value.

Subclasses of Product.java

All of subclasses of Product have the following constructors:

- One that takes in a String for color and a boolean for whether or not the product is limited edition

All of subclasses of Product have the following public methods:

- `String toString()` which correctly overrides Product's toString() and returns a String in the format `Limited edition color subclass: $price` or `color subclass name: $price` if the item is not limited edition. For example, a Kyliner with color Java would have a toString() that returns `Limited edition Java Kyliner: $26.00` . Price should have two decimal points.

There are the three subclasses of Product which differ from one another in price:

- Kyliner.java: $26.00
- Kyshadow.java: $42.00
- LipKit.java: $29.00

ElementNotFoundException.java

Represents an exception that is thrown when an element cannot be found. This is a **checked** exception and has already be written for you.

This class has the following instance fields:

- `object` the Object that caused the exception to occur

This class has the following constructors:

- One constructor that takes in the Object that caused the exception and a String for the exception message and sets the corresponding instance fields.

This class has the following public methods:

- a getter method for `object`

ProductOutOfStockException.java

Represents an exception that is thrown when a product is no longer in stock and cannot be purchased. It is an ElementNotFoundException.

This class has the following constructors:

- One that takes in the product that caused the exception and sets a custom exception message that details the problem that occurred.
- One that takes in the product that causesd the exception and a String for the exception message.

InsufficientFundsException.java

Represents an exception that is thrown when the balance of a credit card is too low to cover the cost of an order. This is a **checked** exception.

This class has the following constructors:

- One that takes in no arguments and creates a custom exception message that details the problem that occurred.
- One that takes in a String for an exception message.

SiteOverloadException.java

Represents an exception that is thrown when the site is experiencing an increased amount of traffic. This is a **checked** exception.

This class has the following constructors:

- One that takes in no arguments and creates a custom exception message that details the problem that occurred.
- One that takes in a String for an exception message.

Server.java

Represents the website server. This class only contains methods that can be used without creating an instance of the class and has been **PARTIALLY** implemented for you.

This class has the following private fields:

- `PERCENT_FAIL` which is an int that represents the 50% chance of the site experiencing an increased amount of traffic. This variable does not change.
- `PERCENT_OUT_OF_STOCK` which is an int that represents the 10% chance that a limited edition product goes out of stock before the customer can add it to the shopping cart. This variable does not change.

This class has the following public methods:

- `void checkTraffic() throws SiteOverloadException` throws a SiteOverloadException `PERCENT_FAIL` % of the times the method is called.

- `void checkStock(Product product) throws ProductOutOfStockException` throws a ProductOutOfStockException `PERCENT_OUT_OF_STOCK` % of the times the method is called.
- `ArrayListInterface<Product> getProducts()` scans in all of the products from `products.csv` into an array list. This method has already been implemented for you.

products.csv

A csv file of all Kylie Cosmetics products. Each line of the file is of the format `product-color-isLimitedEdition`. Feel free to add your own if you wish to do so.

Running and Testing

`Driver.java` has been provided for you. It creates a instance of `CheckoutPage` and allows the user to interact with it. You can run the `main` method to start a simulation and test from there. The tester may not cover all cases, so be sure to write your own code to test your simulation.

Javadocs

For this homework, you will need to write Javadoc comments along with checkstyling your submission.

- Every class should two class level Javadocs, `@author <GT Username>` and `@version 1.0`.
- Every method should have a Javadoc explaining what the method does and includes any of the following tags if applicable.
    - `@param <parameter name> <brief description of parameter>`
    - `@return <brief description of what is returned>`
    - `@throws <Exception> <brief explanation of when the given exception is thrown>`

See the CS 1331 Style Guide (http://cs1331.gatech.edu/cs1331-style-guide.html) section on Javadoc comments for examples.

##Checkstyle

As mentioned in the previous homeworks, you will be running a style checking program on your code. For each violation the tool finds, you will lose one point on your total grade for this assignment.

To make things easier for you in the beginning of the semester, the first few homeworks will have a checkstyle cap, or a maximum amount of points that can be lost to checkstyle. For *this homework*, the **checkstyle cap is 100**, meaning you can lose up to **100** points, the full value of this assignment, due to style errors. **Run checkstyle early, and get in the habit of writing style compliant code the first time.** Don't wait until 5 minutes before the deadline to find out that you have 100+ violations.

- If you encounter trouble running checkstyle, check Piazza for a solution and/or ask a TA as soon as you can!
- You can run checkstyle on your code by using the jar file found on the course website (http://cs1331.gatech.edu/cs1331-style-guide.html):
  `java -jar checkstyle-6.2.2.jar -a *.java`.

- Javadoc errors are the same as checkstyle errors, as in each one is worth a single point and they are counted towards the checkstyle cap.
- **You will be responsible for running checkstyle on *ALL* of your code.**
- Depending on your editor, you might be able to change some settings to make it easier to write style-compliant code. See the bottom of the customization tips page (http://cs1331.gatech.edu/customization-tips.html) for more information.

Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution
- You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

- **approved**: “Hey, I’m really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?”
- **disapproved**: “Yo it’s 10:40 on Thursday… Can I see your code? I won’t copy it directly I promise” In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Submitting

You should not import any libraries or packages that trivialize the assignment. This includes data structures other than arrays (so no `List`, `Map`, `Set`, etc). If you are unsure of whether something is allowed, ask on Piazza. In general, if something does a large part of the assignment for you, it is probably not allowed. Important: `java.util.Arrays` is not allowed. However, that is different from a Java array (e.g `int[] nums = new int[10]`), which is necessary for this assignment.

- The submission tool is included with the HW files. Run it by typing `java -jar hw5-submit.jar`. You can submit as many times as you want so feel free to submit as you make substantial progress on the homework.
- As always, late submissions will not be accepted and non-compiling code will be given a score of 0. For this reason, we recommend submitting early and then confirming that you submitted **ALL** of the necessary files by navigating to the link that the submission tool gives you.