CSc 210: Software Development
Section 9: Inheritance with Raffle Tickets
October 30th, 2017

You've probably bought a raffle ticket at least once – one of those paper tickets with a several digit number on it. This week, you and a partner will be writing classes that support the creation of such tickets. Because there are many similar kinds of tickets with common behaviors, an interface the specifies the common operations is a good place to start.

Part I: Creating a Ticket Interface

1. We will be starting from scratch. Open your editor of choice to begin working.

2. Remember back to in class when we talked about interfaces. You now need to create an interface named **`TicketGeneratable.java`** containing the following methods:
   a. **`public String issueTicket()`**
   b. **`public int quantityIssued()`**
   c. **`public int firstIssued()`**
   d. **`public int lastIssued()`**

3. Save your interface in a file named **`TicketGeneratable.java.`**

Part II: Creating a `TicketGenerator` Class

1. We will now create a basic ticket generator that only supports the methods in our interface. Open a new file called **`TicketGenerator.java`** and create a new public class named **`TicketGenerator`** that implements the **`TicketGeneratable`** interface.

2. A TicketGenerator object is going to a need a few variables:
   a. A public integer constant **`NON_ISSUED`** representing the value -1
   b. A boolean **`anyIssued`** initialized to false
   c. An integer variable named **`nextNumber`**

3. Next, we will need a no argument constructer that sets **`NextNumber`** to zero.

4. Because our class implements **`TicketGeneratable`**, we must implement that interfaces four methods. Create all four of them and have them do the following:
    a. **`issueTicket()`**: Use **`String's format(,)`** method with a format string of **`"%06d"`** to create a six-digit string from **`nextNumber`**'s current value. Then increment the **`nextNumber`** counter, change **`anyIssued`** to true (because we're issuing a ticket), and return **`nextNumber`**'s six character string representation.
    b. **`qtyIssued()`**: Return the number of tickets this object has issued. (Hint: It's closely related to the number of the next ticket.)
    c. **`firstIssued()`**: This method returns the number of the first ticket this object issued. For **`TicketGenerator`**, that's always zero . . . unless no tickets have been issued, in which case the method returns the value of **`NONE ISSUED`**.
    d. **`lastIssued()`**: This method returns the number of the last (most recently-generated) ticket the object issued. Again, if no tickets have been issued, the method returns the value of **`NONE ISSUED`**.

5. Visit the class web page, find the Section9.java program, bring it into the directory you are working in, and open it in your editor. This program does testing of the class you are writing.

6. Compile and run Section9.java. If all of your methods are coded correctly, the output should end with the message: `==>` **`Congratulations! All tests passed!`**.

Part III: Creating a `RaffleTicketGenerator` from `TicketGenerator`

1. With **`TicketGenerator`** done, we can turn our attention to raffle tickets. These need a little more functionality than do basic tickets, while retaining the same basic behaviors. Specifically, we need to be able to generate ticket numbers in new ranges, so that people with numbers from previous raffles don't try to re-use their numbers. And, of course, we need to draw a winner of the raffle. Extra functionality means extra methods, but to retain the functionality that **`TicketGenerator`** provides, we need . . . inheritance

2. In **`Section9.java,`** there are two comment lines with lots of equal signs and the text "Remove this line at the start of Part III!" Do it; delete both of those lines, but only those two lines.

3. On the class web page is the file **RaffleTicketGenerator.java** Download it, move it to your working directory, and open it in your editor. The first line of the **RaffleTicketGenerator** class is missing. Create it, keeping in mind that this class inherits from **TicketGenerator** and also inherits from **TicketGeneratable**.

4. **RaffleTicketGenerator** needs two constructors. The no-argument constructor is very nearly the same as the no-argument constructor for **TicketGenerator**; we've supplied it for you. The second constructor accepts the starting number for the sequence of raffle numbers, and uses it to initialize **nextNumber** and **startNumber** (the new instance variable in **RaffleTicketGenerator**). Using the no-argument constructor as a guide, write this second constructor.

5. Thanks to inheritance, we don't need to re-write any of the methods from **TicketGenerator** that don't change. Specifically, **issueTicket()** and **lastIssued()** can be reused. We do need to 'replace' **quantityIssued**() and **firstIssued()**, because they depend on the sequence starting number, which in this class doesn't need to be zero. Question: Is this 'replacement' of these two methods an example of overriding or overloading?

6. Using the **TicketGenerator** version as a starting point, write the new version of **firstIssued();** we've given you the new version of **qtyIssued()**.

7. **RaffleTicketGenerator** needs two new methods. Create both of them, such that they each do what we need them to do:
    a. **reset()** is a void method that accepts a new starting number for a raffle ticket sequence and resets the state of the current object so that it appears to be a brand-new **RaffleTicketGenerator** object, ready to issue the given starting number as the first number of the sequence. Thus, **reset()** is almost the same as the second constructor; the difference is that **reset()** does not create a new object; instead, it changes the state of the current object.
    b. **drawWinner()** randomly selects and returns one number from the range of ticket numbers that have been issued by this object. If no numbers have yet been issued, this method returns the value of **NON_ISSUED**.
8. Time to test the **RaffleTicketGenerator** methods! In **Section9.java**, we've provided a start on the testing of **RaffleTicketGenerator**, but you need to complete it. Near the bottom is a comment that says:

```
/* Part III: Your job is to add the assertion-based testing
 * for the second constructor. Fortunately, you have the
```

```
     * above tests as guides. Suggestion: Copy-n-paste the
     * code that tests Constructor 1 of RaffleTicketGenerator,
     * and adjust it to test Constructor 2. */
```

Do it!

9. Compile and run **`Section9.java`**. When you no longer have syntax and logic errors — that is, when you see the congratulations message — you're done.