CSCI 221, Spring 2017 Due: Friday, April 14, 11:59 pm
Programming HW No penalty date: 24 hours later

Modify the **Ticket**[1] class from HW 6 to:

1. Throw a **BadTicketNumberException** if anyone attempts to store a negative ticket number or reuse an already used ticket number. (Note this will require implementation of the **BadTicketNumberException** class and **Ticket** will have to keep track of all used ticket numbers). **BadTicketNumberException** should display one of the messages (with the actual number value replacing the ##).:
   "Exception: Cannot sell ticket number ##. Ticket number already in use."
   "Exception: Cannot sell ticket number ##. Invalid number provided."
2. (5 points) The subclasses should not catch or deal with any **BadTicketNumberException** thrown by Ticket. They should throw them (include "throws **BadTicketNumberException**" as part of the signatures of their constructors) to their caller.
3. (10 points) Modify the **AdvanceTicket** class to throw a **BadDaysRangeException** if anyone attempts to store a number of days in advance that is non-positive or greater than 180. Note this will require implementing the **BadDaysRangeException** class. The **BadDaysRangeException** class should display a message like the following (with the actual number value replacing the ##):
   "Exception: Cannot sell a ticket with ## days in advance."
   "Exception: ## Days in advance specified must be positive."
4. (10 points) Modify **Ticket** to implement the **Comparable** interface – tickets are compared based on ticket number. Ticket with number 11 "is less than" ticket with number 12, etc. The natural ordering of integers should be used.

Hint: Make sure that the above works, before moving on, perhaps by including test drivers in the subclasses.

5. (45 points) Develop a test program called **TicketTester**. The shell is provided for you. **TicketTester** reads input, and creates and stores Ticket in an ordered list as described below. **TicketTester** gets its data from an input file (named "inputFile.txt") that includes an integer that indicates the type of ticket requested: 1, 2 or 3 (are valid) for WalkupTicket, AdvanceTicket, or AdvanceStudentTicket, respectively, followed on the same line by a ticket number, and number days in advance, if appropriate.

Sample input file:

```
1 25
2 25             // bad data, ticket number already used
2  35 -10        // bad data, number of days in advance not acceptable value
4 35             // bad data, 4 has no meaning.
3 23 200        // bad data, 200 is out of range of acceptable days
…
```

If an input file contains a ticket code other than 1, 2 or 3, **TicketTester** throws a **BadInputCodeException** (you must write this one too!) which displays a message such as
"Bad code # encountered on line XX of <filename>. Line ignored."

[1] I will post a solution to ProgHW6 on Sunday, April 9, that you can work from, if you wish.

TicketTester ignores bad lines of data (due to any exception type) – does not store a ticket from that line, just moves on to the next line of data.

When good data is input, **TicketTester** creates a **Ticket** of the appropriate type and stores it in an <u>ordered</u> list of Tickets. The tickets are ordered by ticket number, since this is how Ticket we defined the compareTo method in Ticket.

TicketTester handles all exceptions thrown by the Ticket classes (**BadTicketNumberException** and **BadDaysRangeException**) or detected on its own (**BadInputFormatException**, **FileNotFoundException**, if appropriate, and general exception above not specifying input file, as done for you in shell). **REPEAT:** EXCEPTIONS ARE NOT HANDLED IN THE TICKET CLASSES.

How to store the list of Tickets:
The list is implemented using an ArrayList (NOT AN ARRAY - see the Java api for how to use an ArrayList). (Note, you must provide the code to find the right location for the newly purchased ticket and put it in the list at the correct location – you **may not** add it to the end and then call a sort method.)

The **compareTo** method in Ticket will be useful in finding the correct position, and ArrayList methods make it easy to add the new element wherever it is needed, without you have to move the other elements. (NOTE: the ArrayList is similar to the StringsList we created earlier. Read the documentation, it is easy to use.)

Point breakdown for TicketTester: Read input from file and handle exceptions as necessary (10 points), BadInputFormatException (5 points), use create ordered list of tickets as specified (15 points), catch exceptions and handle as described (15 points).

If TicketTester terminates normally, it prints out the contents of the ordered list of tickets before terminating.

As always:

- Document each class as normally expected. (20 points)
- We'll grade only if you do the usual:
    - Put all java files into a folder named: yourlastnameHW6 and zip it up. For example, my folder would be McCauleyHW7 and when zipped the file would be named McCauleyHW7.zip.
    - Upload to OAKS by the due date and time.

The TicketTester shell is provided in TicketTester.java