Lab 5: SortableVector

Due on 3/28/24 (Thu) at 11:59pm. Refer to Part 4 for submission instructions.

Learning Objectives:
- You can write classes that implement the Comparator interface.
- You can write code to sort by various criteria.
- You can write code to read from a text file.
- You can handle exceptions using a try-catch block.

Grading:
- **Correctness:** 40 pts
  - **1.2:** 7 pts
  - **1.3:** 7 pts
  - **2.1:** 3 pts
  - **2.2:** 3 pts
  - **2.3:** 3 pts
  - **2.4:** 3 pts
  - **(- 2.5:** 2 pts)
  - **3.2:** 4 pts
  - **3.3:** 10 pts
- **Style:** 7 pts (See below)
- **Reflections:** 3 pts (See Part 3)

Compiling. Code that does not compile will earn at most 25% of the assigned points. So, please make sure your code compiles! If you have too many compile-time errors, it will be hard to identify the real source(s) of error. The trick is to write a short compilable code first. Then, every time you add some functionality, compile again. You'll be compiling very frequently, but it'll save your time in the long run. Remember, save often! compile often!

Style. Make sure your code is legible. You are extremely unlikely to be programming alone post-graduation, and even if that were the case, your future self will thank you for keeping the code legible! Here are some ways to make your code more legible:

Code
- Use meaningful variables names. "a", "b", "x" and "y" are not meaningful names. Use informative names like "name1", "name2", "num1", "num2", "isBigger", "isSmaller", etc.
- Use straightforward logic and flow-of-control.
- Avoid magic numbers. A magic number is a numeric literal that appear in the code, out of blue. It is better to store them in a variable, so that others know what they are. For example, if 12 randomly appears in the code, other programmers (or your future self) will wonder what it is, but if it is stored in a variable, say MONTHS_IN_A_YEAR, you'd know that it is supposed to represent the number of months in a year.

- Declare variables to minimize scope. Basically, declare variables when you need them; don't declare all of them at the beginning of a class or method. Otherwise, when the value is different from what you expect, you will have more lines of code to examine to fix the problem.

Spacing
- Use whitespace for readability; insert blank lines to keep related lines together and others apart. Every blank line should be used with a purpose of separating less related lines of code.
- Indent your code consistently. All the lines between "{" and "}" should be indented one level deeper.

Commenting
- Class header comments: Include a comment describing the given class above each class declaration. The comment should be surrounded by `/**` and `*/`. Be sure to include `@author author name` to specify the author of the program.
- Method header comments: Include a comment describing the given method above each method. The comment should be surrounded by `/**` and `*/`. Be sure to include `@param param_name description` for each parameter, and `@return description` for the return value if it is not void.
- Member variable comments: Include a comment describing the given variable, above each member variable. The comment should be surrounded by `/**` and `*/`.
- Inline comments: Include a brief comment describing multiple lines of code that serve a purpose. Use // for short in-line comments and `/*` and `*/` for longer ones.

PART 0: Setup

0.1 Starter Package. Download the starter package from Box.

PART 1: SortableVector

In this lab, you will extend `Vector` to build a new class `SortableVector` that adds a sorting functionality to the `Vector` class. You will then use this class to answer questions about 2020 presidential election results! (Ok, it's a bit outdated..)

1.1 SortableVector. Write SortableVector class that extends the Vector class in the java.util package. Since we are creating a generic class, the class header for SortableVector should be as follows:

```
public class SortableVector<E> extends Vector<E>
```

Also, be sure to import Vector and Comparator from java util package above the class header comment:

```
import java.util.Vector;
import java.util.Comparator;
```

Note that we will **not** define any constructors in this class. As we have discussed before, when a class does not define a constructor, Java automatically adds a default constructor, which is a constructor with no parameters with an empty body. Since super(); is automatically added when you do not explicitly call super() or this(), super() is added to this constructor. In other words, by not defining any constructors, you have effectively defined the following constructor:

```
public SortableVector(){
  super();
}
```

1.2 selectionSort. Write `selectionSort(Comparator<E> c)` method. This method should sort the vector from the least to the greatest using the selection sort algorithm and a Comparator object c. (It is ok to look up the algorithm, but not actual implementations.)

[Note: Since selectionSort() is non-static, selectionSort() is always called using a SortableVector instance. And that instance is the "this" instance in selectionSort(). Thus, inside selectionSort(), you can call non-static methods like get() without creating a SortableVector object; e.g. you simply call this.get(0) to get the first element.

Here, get(0), this.get(0), and super.get(0) all refer to the get() method defined in the Vector class. If you're not sure why this is the case, please review the lectures on inheritance.]

[Hint: Take a look at the javadoc for Vector to see which methods are inherited. For instance, toString() can be handy for printing the content during testing.]

1.3 insertionSort. Write `insertionSort(Comparator<E> c)` method. This method should sort the vector from the least to the greatest using the insertion sort algorithm and a Comparator object c.

PART 2: Comparators

In the starter package, you will find a file called `VotesByState.java`. An instance of this class can store presidential election results for a state. You will now define several Comparator classes to allow multiple means of comparing VotesByState objects. In each file, be sure to import the Comparator interface as follows:

```
import java.util.Comparator;
```

[Note: For these Comparator classes, you don't need to add comments.]

2.1 StateComparator. Write StateComparator class that implements Comparator<VotesByState>. The compare() method should compare 2 VotesByState objects based on the state name (alphabetical order). [Hint: The String class implements Comparable, as we have seen in class.]

2.2 TrumpVotesComparator. Write TrumpVotesComparator class that implements Comparator<VotesByState>. The compare() method should compare 2 VotesByState objects based on the count of electoral votes won by Trump.

2.3 BidenVotesComparator. Write BidenVotesComparator class that implements Comparator<VotesByState>. The compare() method should compare 2 VotesByState objects based on the count of electoral votes won by Biden.

2.4 TotalVotesComparator. Write TotalVotesComparator class that implements Comparator<VotesByState>. The compare() method should compare 2 VotesByState objects based on the count of electoral votes won by Trump and Biden.

2.5 [Extra Credit] Unit Testing. Create a JUnit test file with test cases to test sort methods in SortableVector using each of the comparators. A comprehensive test will test SortableVectors of different lengths, and you do not have to test with a SortableVector that contains all states.

PART 3: Driver

We are now ready to play with the 2020 presidential election results! You will create a new Driver class to read from a data file and answer a few questions using the classes you implemented in this lab. This class consists of main() and other static methods called from main().

3.1 Driver. Write Driver class. It is a simple class that does not extend another class nor implement an interface. However, you will need to import the following for file input/output:

```
import java.util.Scanner;
import java.io.File;
import java.io.IOException;
```

3.2. readFromFile. Write method readFromFile(String fileName, SortableVector<VotesByState> data) that reads in content from the file and adds to the vector.

(a) To read from a file, you will use Scanner, again. But, this time, you read from a file instead of the terminal. To do this, you need to use a Scanner constructor that takes in a File object; the File object in turn is created by a File constructor that takes in a file name in the String format:

```
Scanner scanner = new Scanner(new File(fileName));
```

[Hint: if the file is in the same directory, fileName can simply be the file name, e.g. “data.txt” If not, fileName has to be the path including the directories.]

(b) Once you create a Scanner object, the way you use it is no different than how you would use it when reading from the terminal; call an appropriate method from the Scanner class.

[Hint: A common practice, however, is to read line by line. While there are lines remaining, read the next line. Which method can you use to see if there are remaining lines? Which to read the next line?]

(c) One complication with reading from a file, rather than the terminal, is that many things can go wrong: The specified file may not exist or be corrupted. As these erroneous conditions cannot be checked during compile time, you won't get a compiler error. Instead, you get something called an "exception." Certain constructors and methods (the File constructor in this case) can throw exceptions, and they need to be properly handled using a try-catch block. Simply put, you first "try" to do something, and if an exception is thrown, "catch" and deal with it. The way you deal with it is typically to print a detailed error message using the printStrackTrace() method.

Putting together (a)~(c), we have the following:

```
try (Scanner scanner = new Scanner(new File(fileName))){
  while (scanner.hasNextLine()){
    String line = scanner.nextLine();

    // create a VotesByState object using line
    // and add to the vector
  }
}catch(IOException e){
  e.printStackTrace();
}
```

Lastly, each line in the file should be in the following **format**: State, Electoral votes won by Trump, and Electoral votes won by Biden. Take a look at data.txt to see how it is formatted.

[Hint: someString.split(",") splits someString whenever "," appears and returns it as a String array. Take a look at the Integer class to see how you can convert a String to an int.]

3.3. main(). Write the main() method to **compute and print** answers to the following questions by reading from data.txt to create a SortedVector of VotesByState, and sorting it as necessary. You may define additional methods to answer the questions if you wish. **Be sure to include the answers in the class header comment** for the Driver class to help with grading**.**

a. What is the 25th state when states are ordered alphabetically?
b. What are the top 3 states where Trump won the most votes? Print the states in the order of decreasing number of votes. If there's a tie, print any 3.
c. What are the top 3 states where Biden won the most votes? Print the states in the order of decreasing number of votes. If there's a tie, print any 3.

d. What is the most common number of electoral votes (i.e., the mode), and which states have this number of electoral votes? [Hint: If the SortableVector is properly sorted, you can compute the mode by going through the Vector just once!]

PART 4: Submission

4.1 Lab Submission.

Steps:

- Open up a browser, and go to https://richmond.account.box.com/login
- Log in using your UR login.
- Navigate to a directory named after you netid. (It should have "Jon Park" as the owner)
- Click on [New] in the upper right-hand corner of the browser and select "File Upload"
- Upload a directory named "**lab5**" containing the following files, **and no other files**:
  - `SortableVector.java`
  - `SortableVector.html`
  - `StateComparator.java`
  - `TrumpVotesComparator.java`
  - `BidenVotesComparator.java`
  - `TotalVotesComparator.java`
  - `Driver.java`
  - `(- SortableVectorTest.java)`
  - `VotesByState.java`
  - `data.txt`

3.2 Meta-cognition. Now that you have completed the lab, it's time to reflect on your learning experience; please submit a short paragraph here: http://reflect.221.joonsuk.org Remember, this is for your own good. You may reflect on anything you wish, but be sure to answer the following mandatory questions.

- Have you met the learning objectives?
- If you were to go back in time and redo this lab, what would you continue doing? What would you do differently?
- Can you think of any interesting dataset(s) that can be sorted in multiple ways?