CSC 102 Final Exam

The exam is open-book and open-note. Most of your answers will be submitted via handin. Answers to the last two questions are written on the final exam paper. Instructions for the individual questions indicate the form of your answer. In all cases, you may use the computer to generate and/or check your answers.

No Javadoc documentation is required for any of your answers.

The exam is worth 110 points total. Individual point values are indicated for each question.

For questions 1 through 5 you will be implementing methods for the following classes:

`PointList` -- a collection of points in 2-dimensional space
`Point` -- a single point in 2-space

Class `PointList` contains an ArrayList of `Point`s, and provides the following public methods:

PointList() -- the default constructor
void add(Point) -- add the given point to the list
void remove(int x, int y) -- remove the first point with the given x,y coordinates; do nothing if no such point
void removeAll() -- remove all points from the list
Point find(int x, int y) -- return the first point with the given x,y coordinates; return null if no such point
Point get(int i) -- return the ith point in the list, null of no such point; first element is at index 0
int numPoints() -- return the number of points in the list
void sort() -- sort based on the order defined by `Point.compareTo`
boolean equals(Object other) -- return true if each point of this list is `equals` to each point of the the *other* list
String toString() -- return the `toString` of each point, separated by newlines
void inputFromStdin() -- read points from standard input and put them into a list
void inputFromTextArea() -- read points from a JTextArea and put them into a list

Class `Point` has private int data fields for the x and y coordinates. When a point is displayed on the screen, the 0,0 origin of the point is in the center of the GUI frame. `Point` provides the following public methods:

public Point(int x, int y) -- initializing constructor
public int getX() -- return the x coordinate
public int getY() -- return the y coordinate
public boolean equals(Object other) -- return true if x and y values are equal
public int compareTo(Point p) -- details described below
public String toString() -- details described below

The `Point` `compareTo` method operates as follows for two points p1 and p2:

- if p1.x < p2.x, then return -1
- if p1.x == p2.x, then return the comparison of p1.y and p2.y
- otherwise return 1

The "comparison of p1.y and p2.y" means the values are compared in the "compareTo" sense: return -1 for p1.y < p2.y, 0 for p1.y == p2.y, and 1 for p1.y > p1.y. In the `compareTo` method, p1 is represented by `this`.

The `Point.toString` method returns a line of the form "Point: x = *xval*, y = *yval*". E.g., for a point with x,y values of 10 and 20, `toString` returns "`Point: x = 10, y = 20\n`" (note terminating newline character).

There are additional testing and GUI classes that will be used in your answers to the final. Details of these classes are presented in the questions below. The Java source code for all of the testing classes, as well as HTML and PDF for the exam are in the directory `~gfisher/classes/102/final`.

In addition to the classes presented above and in the questions, you may add any other classes you feel are necessary to produce a working solution, i.e., a solution that meets the requirements of the questions on the final exam. You may also add additional methods and data fields to any presented class.

The way the final is written, you are working on incremental implementation of the `PointList` and `Point` classes. There are a number of testing classes provided for each step of the implementation. If you do not get all tests to work, then be sure to submit something that compiles and runs for the tests you can complete.

When you have completed your work, submit *all* .java files, including the tester files that are provided. Do so via handin on vogon, as `handin gfisher final *.java`.

1. (15 points) Implement the `inputFromStdin` method in the `PointList` class. Also implement as many other of the methods in `PointList` and `Point` as are necessary to successfully execute the testing program provided in `PointListStdinTester.java`. Compile these three files together:

   ```
   PointList.java Point.java PointListStdinTester.java
   ```

   and run `PointListStdinTester`. The tester will prompt with "Input a sequence of points, ending in 'q'", after which you enter pairs of values on the standard input terminal, ending in the letter 'q'. A sample run looks like this:

   ```
   Input a sequence of points, ending in "q"
   1 2
   3 4
   5 6
   q
   Point: x = 1, y = 2
   Point: x = 3, y = 4
   Point: x = 5, y = 6
   ```

   The values do not need to be in pairs on a line. Blanks, tabs, and newlines are all considered whitespace. If the user enters an odd number of numeric values, the last y value is considered to be 0.

   You have successfully answered this question on the final when you can complete one such run. The next question deals with error handling.

2. (10 points) In this question, you are to add additional test cases to the `inputFromStdin` class. Specifically, you need to test that your implementation of `inputFromStdin` handles all occurrences of the following exceptions, in the designated manner:

Exception	**Manner Handled**
NullPointer	User never sees it happen
InputMismatchException	Output "illegal input" to standard error stream
NoSuchElementException	Output "illegal input" to standard error stream

   In addition, the program should consider any form of non-numeric input except the letter 'q' to be an error, for which it outputs "illegal input" to stderr. When any input value is illegal, that value is discarded, but input processing continues.

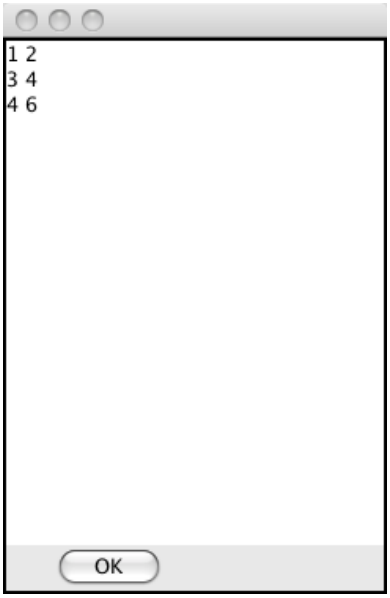
   Compile the same three files together as for question 1, i.e.,

   ```
   PointList.java Point.java PointListStdinTester.java
   ```

   and run `PointListStdinTester`.

   The specific test cases are up to you. When your program is graded, it will be tested against test cases that are designed to generate exceptions. Your job for this question is to make sure that you catch the three exceptions listed in the table above, and handle them as described in the table.

3. (15 points) In this question, you input values from a GUI instead of `stdin`. Specifically, you are to use a JTextArea for point value input, with an OK button at the bottom. The GUI looks like this:



In this example, the user has typed the same values as in the stdin test case. When the user presses the 'OK' button, the program outputs the `PointList` containing the point values typed into the text area. E.g., in the picture above, pressing the OK results in the following lines on the stdout stream:

```
Point: x = 1, y = 2
Point: x = 3, y = 4
Point: x = 5, y = 6
```

which should be the result of calling the `PointList toString` method. Note well that the program takes input from the GUI, but outputs to stdout.

The JTextArea can have a fixed size of 20 x 20 rows and columns. The JButton can appear anywhere on the horizontal axis, as long as it's below the text area.

To test your implementation, compile these files together:

```
PointList.java Point.java ... PointListGuiInputTester.java
```

where the "..." are one or more additional .java files that you write to display the GUI and process its inputs. Run the program by running `PointListGuiInputTester`. The source code for the tester program is provided in the final source directory. You may modify the test program as you see fit, but you should not need to.

4. (30 points) Implement all of the remaining methods in `PointList` and `Point` that you have not yet implemented for the previous questions. Then test your implementation by compiling these files together:

   ```
   PointList.java Point.java ... PointListModelTester.java
   ```

   where the "..." are any additional files you think are necessary. (Hint: no additional files should be necessary; also, you don't need the GUI class(es) from the previous question.)

   In this question, testing in performed entirely from within the testing program with no terminal or GUI input from the user. This is the style of testing we did on programs 1, 3, and 4.

   There is a very skeletal testing file provided in `PointListModelTester.java`. Your job for this question is to add tests where the comments indicate, i.e., in the places where it says

   ```
   /*
    * ADD TEST CASES HERE.
    */
   ```

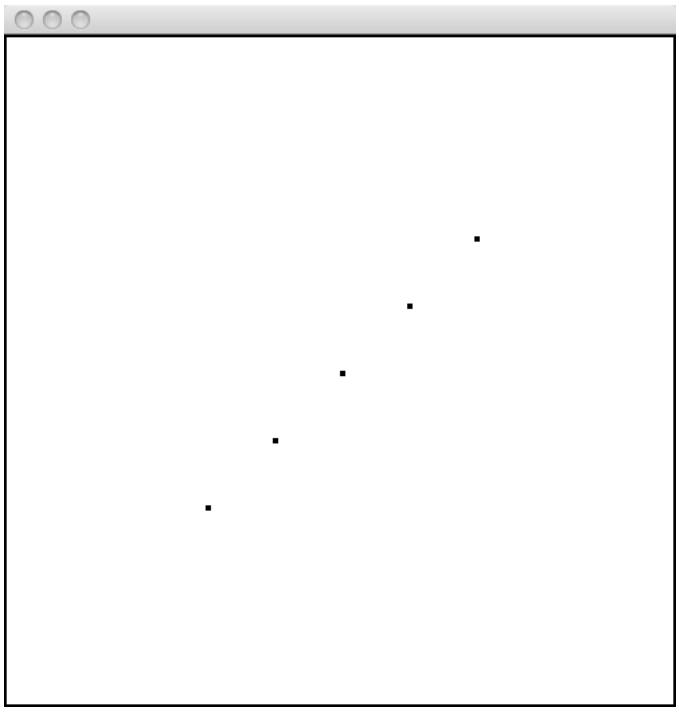
The test cases are entirely up to you. When your program is graded, it will be tested against test cases that are designed to exercise the `PointList` model. Your job for this question is to make sure that you implement the model operations correctly.

5. (20 points) In this question you are to implement two additional classes to display elements of a `PointList` in a 2-dimensional plot -- `PointListCanvas` and `PointListFrame`. Theses classes play analogous roles of the canvas and frame classes in Programs 5 and 6.

   For this exam question, the GUI classes display a plot of the points in a `PointList`. For example, with data generated by the following testing loop

   ```
   for (i = -100, j = -100; i <= 100; i += 50, j += 50) {
       plist.add(new Point(i,j));
   }
   ```

   a point plot looks like this:



   In the display, the points are plotted as 4x4-pixel filled black rectangles, with the 0,0 origin of the display in the center of the window. The display size is a fixed 500 x 500 pixels.

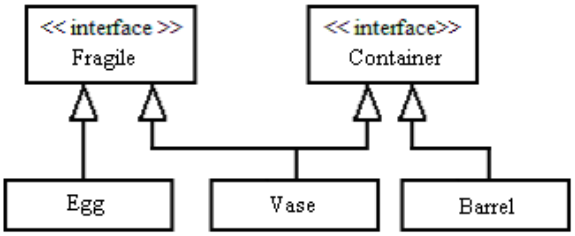
   To test your implementation, compile these files together:

   ```
   PointList.java Point.java PointListCanvas.java PointListFrame.java ...
   PointListPlotTester.java
   ```

   where ... are any additional files you feel are necessary. PointListPlotTester.java is provided in the final testing directory. When your program is graded, it will be tested against test cases that generate plots in addition to the one plot that is in the provided testing program. Your job for this question is to make sure that you can generate a point plot for any point list. As with Programs 5 and 6, the points can be off of the visible display, which means they are drawn, but not visible on the screen.

The following questions are to be answered on paper.

6. (10 points) Assume the following class hierarchy:



Further assume that

- the interface Fragile contains the method `public double forceToBreak()`
- the interface Container contains the method `public void addContents(Container other)`
- the class `Vase` contains the method `public double value()`

Finally, assume that the references below are in some test method and all object instantiations are legal. The actual objects created are purposely not shown.

```
public void test () {
    Fragile fragile = new ...
    Container container = new ...
    Egg egg = new ...
    Vase vase = new ...
    Barrel barrel = new ...
    // additional code
```

For each of the following code fragments, write **always**, **sometimes**, or **never**, in the right hand column. Always means the fragment will compile and run. Sometimes means the fragment will compile, but may fail at runtime. Never means the fragment will not compile. Each statement is stand alone, in that the statements are in no particular order.

Code Fragment	**Validity (always, sometimes, or never)**
`barrel.addContents((Vase)fragile)`	
`vase = container`	
`fragile = vase`	
`container.value()`	
`((Vase)fragile).value()`	
`container = (Vase)egg`	
`egg.forceToBreak()`	
`vase.addContents((Container)egg)`	

7. (10 points) Define in O-notation, running times for the following `PointList` methods, as implemented by you:

```
add
remove
removeAll
find
get
sort
```

Briefly justify your answers, where brief means in a sentence. For example, an answer and justification for the `equals` method would look like this:

> `equals` is O(n) because it checks each point in the list, calling an O(1) equals method for each point

For methods that rely on Java library methods, your justification should state your assumptions about the running time of the library methods, based on the library documentation of the methods. That is, you must read the library method documentation to determine the method's runtime performance.