Com S 228 Spring 2015

Project 4: Convex Hull (150 pts)

Due at **11:59pm**

Thursday, Apr 9

1. Project Overview

In this project, you are asked to implement Graham's scan to construct the convex hull of an input set of points in the plane. For a description of the algorithm, please refer to the lecture notes "`convex hull.pptx`" on Monday March 23rd.

We make the following two assumptions:

a) *All the input points have **integral** coordinates ranging between* -50 *and* 50*.*
b) *The input points may have **duplicates**.*

In a), integral coordinates are assumed to avoid issues with floating-point arithmetic. The rectangular range $[-50, 50] \times [-50, 50]$ is big enough to contain 10,201 points with integral coordinates. Since the input points will be either generated as pseudo-random points or read from an input file, duplicates may appear.

Convex hull construction is implemented by the `ConvexHull` class, for which there are two constructors:

```
public ConvexHull(int n) throws IllegalArgumentException
public ConvexHull(String inputFileName) throws FileNotFoundException,
                                               InputMismatchException
```

The first constructor generates a specified number of random points, or more precisely, points whose coordinates are pseudo-random numbers within the range $[-50, 50] \times [-50, 50]$. Please refer to Section 3 on how such points can be generated. The generated points will be stored in the private array `points[]`.

The second constructor reads points from an input file of integers. Every pair of integers represents the x and y-coordinates of a point. A `FileNotFoundException` will be thrown if no file by the `inputFileName` exists, and an `InputMismathException` will be thrown if the file consists of an odd number of integers. (There is no need to check if the input file contains unneeded characters like letters since they can be taken care of by the `hasNextInt()` and `nextInt()` methods of a `Scanner` object.)

For example, suppose a file `coordinates.txt` has the following content:

```
0  0 -3 -9  0 -10
8 4 3 3
-6 3
```

```
-2 1
10 5
-7 -10
5 -2
7 3  10 5
-7 -10 0 8
-1 -6
-10 0
5 5
```

There are 34 integers in the file. A constructor call `ConvexHull("coordinates.txt")` will initialize the array `points[]` to store 17 points below (aligned with five points per row just for display clarity here):

```
(0, 0)   (-3, -9)  (0, -10)   (8, 4)     (3, 3)
(-6, 3)  (-2, 1)   (10, 5)    (-7, -10)  (5, -2)
(7, 3)   (10, 5)   (-7, -10)  (0, 8)     (-1, -6)
(-10, 0) (5, 5)
```

Note that the points `(-7, -10)` and `(10, 5)` each appears twice in the input, and thus their second appearances are duplicates. The 15 distinct points are plotted in the figure below by Mathematica.

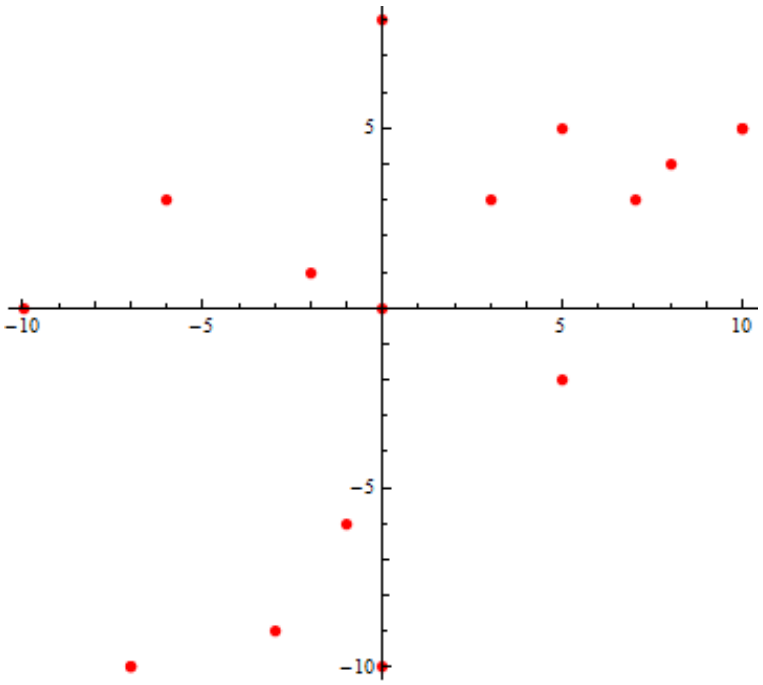


Fig. 1. Input set of 15 different points.

2. Convex Hull Construction

Before we perform Graham's scan, duplicate points from the input need to be eliminated. Also, if multiple points have the same polar angle with respect to the lowest point, all but the furthest one from this point could be removed.

2.1 Sorting by Polar Angle

All the input points are already stored in the private array `points[]` of the `ConvexHull` class. Find the lowest of these points, and store it in the private instance variable `lowestPoint`. This operation is carried out by the following method:

```
private void lowestPoint()
```

Note that multiple points from the input might be equal to `lowestPoint`. If multiple points share the smallest y-coordinate, let `lowestPoint` store the leftmost one of them.

We then sort all the points in `points[]` by polar angle with respect to `lowestPoint`. Point comparision is done using an object of the `PointComparator` class, which you are required to implement. The `compare()` method in this class must be implemented using cross and dot products not any trigonometric or square root functions. You need to handle special situations such as multiple points are equal to `lowestPoint`, have the same polar angle with respect to it, etc. Please read the Javadoc for the `compare()` method carefully.

Use quicksort by implementing the following three methods:

```
public void quickSort()
private void quickSortRec(int first, int last)
private int partition(int first, int last)
```

These methods operate on the array `points[]`. Note that quicksort has the expected running time $O(n \log n)$ and the worst-case running time $O(n^2)$, which will respectively be the expected and worst-case running times for this implementation of Graham's scan for convex hull construction.

In the example from Section 1, `lowestPoint` is set to (-7, -10). The array `points[]` after the sorting will have the 17 points ordered below:

```
(-7, -10) (-7, -10)  (0, -10)  (-3, -9)  (-1, -6)
(5, -2)   (10, 5)    (10, 5)   (7, 3)    (8, 4)
(5, 5)    (3, 3)     (0, 0)    (-2, 1)   (0, 8)
(-6, 3)   (-10, 0)
```

Note that (-1, -6) and (5, -2) have the same polar angle with respect to (-7, -10). That (-1, -6) appears before (5, -2) is because it is closer to (-7, -10).

Sorting is performed as the first step by the private method below.

```
private void setUpScan()
```

2.2 Point Elimination

After the sorting, equal points will appear next to each other in `points[]`. In the example above, the two `(-7, -10)s` appear together, so do the two `(10, 5)s.` We can easily eliminate duplicates, and store the remaining points in the array `pointsNoDuplicate[]`. The array `pointsNoDuplicate[]` stores only 15 points with `lowestPoint` at index 0:

```
(-7, -10)  (0, -10)  (-3, -9)  (-1, -6)  (5, -2)
(10, 5)    (7, 3)    (8, 4)    (5, 5)    (3, 3)
(0, 0)     (-2, 1)   (0, 8)    (-6, 3)   (-10, 0)
```

When multiple points have the same polar angle with respect to `lowestPoint`, only the one that is the furthest from `lowestPoint` can possibly be a vertex of the convex hull. So we can eliminate the remaining points from the group. Copy the elements from the array `pointsNoDuplicate[]` onto a new array `pointsToScan[]` while performing this type of point eliminations. In `pointsToScan[]`, every point is distinct and (not counting `lowestPoint`) has a different polar angle with respect to `lowestPoint`. Clearly, this array also has `lowestPoint` at index 0.

In the sample example, as mentioned earlier in Section 2.1, both `(-1, -6) and (5, -2)` in the array `pointsNoDuplicate[]` have the same polar angle with respect to `lowestPoint (-7, -10).` The point `(-1, -6)` is eliminated and does not appear in `pointsToScan[]`:

```
(-7, -10)  (0, -10)   (-3, -9)  (5, -2)   (10, 5)
(7, 3)     (8, 4)     (5, 5)    (3, 3)    (0, 0)
(-2, 1)    (0, 8)     (-6, 3)   (-10, 0)
```

The 14 points in `pointsToScan[]` are to be scanned. This completes the method `setUpScan()`.

For convenience of elimination, you may use `ArrayLists` and even self-defined comparators.

2.3. Graham's Scan

After two rounds of point eliminations, Graham's scan is performed on the array `pointsToScan[]` using a private stack `vertexStack`. The method

```
public void GrahamScan()
```

calls `lowestPoint()` and `setUpScan()` before the scan. As the scan terminates, the vertices of the constructed convex hull are on `vertexStack.` Pop them out one by one and store them in a new array `hullVertices[]`, starting at the highest index. When the stack becomes empty, the elements in the array, in increasing index, are the hull vertices in counterclockwise order.

Fig. 2 displays the convex hull constructed over `pointsToScan[]` for the same earlier example, together with all the points stored in `pointsNoDuplicate[]` (i.e., all the distinct points from the input).

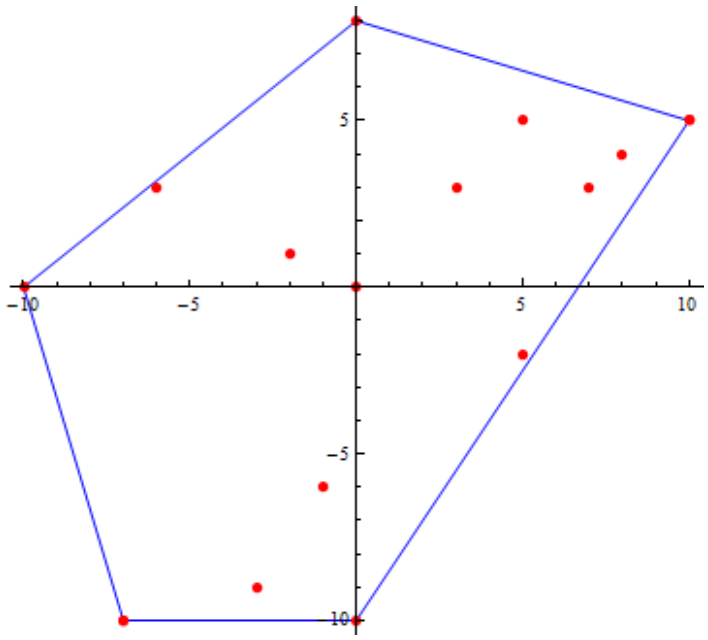


Fig. 2. Convex hull of the input points shown in Fig. 1.

The method `GrahamScan()` must handle the **special case** of one or two points only in the array `pointsToScan[]`. The corresponding convex hull is the sole point or the segment connecting the two points.

3. Random Point Generation

To test your code, you may generate random points within the range $[-50, 50] \times [-50, 50]$. Such a point has its x- and y-coordinates generated separately as pseudo-random numbers within the range $[-50, 50]$. You already had experience with random number generation from Project 1. Import the Java package `java.util.Random`. Next, declare and initiate a Random object like below

```
Random generator = new Random();
```

Then, the expression

```
generator.nextInt(101) – 50
```

will generate a pseudo-random number between -50 and 50 every time it is executed.

4. Display in Mathematica

The point set and its convex hull need to be displayed in Mathematica. The display helps you visualize the geometry to check the correctness of your output. The software is licensed by ISU and available to students. To install Mathematica on your computer, please follow the instructions in the online document below:

https://www.it.iastate.edu/sldb/docs/Mathematica-Install-for-Students.pdf

Setting up Mathematica and doing the test take around three hours, most of which will likely be spent on waiting through.

4.1 File Preparation

The method `hullToFile()` writes the vertices of the convex hull, stored in the array `hullVertices[]` in **counterclockwise** order, into the file `hull.txt`. Every line displays the x and y-coordinates of a single vertex, with **at least one blank space** between the two coordinates. Writing starts at `lowestPoint`. After all vertices are written, write the coordinates of `lowestPoint` **again on the last line**. This format ensures Mathematica to draw a closed polygon not one missing the edge connecting the last vertex and `lowestPoint`. For the convex hull shown in Fig. 2, the file `hull.txt` should have its content like:

```
-7 -10
0  -10
10  5
0   8
-10  0
-7  -10
```

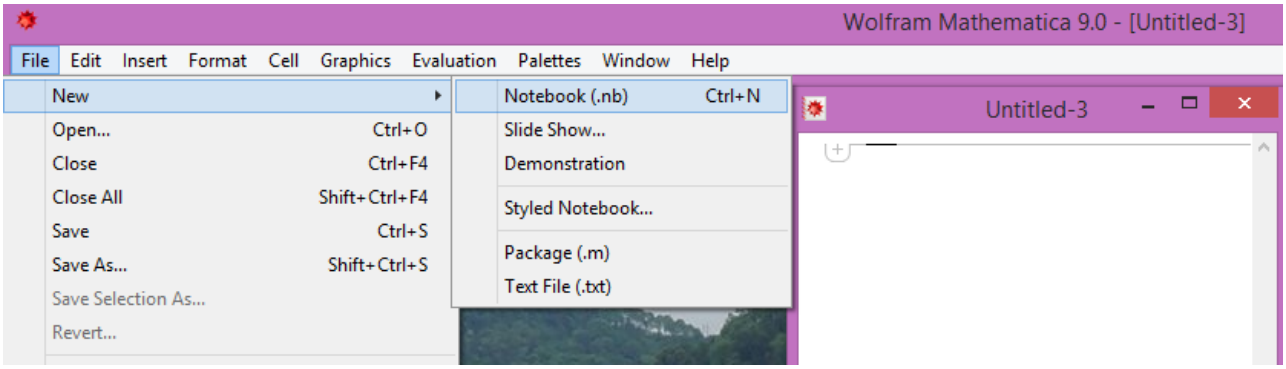
The method `pointsToFile()` writes the distinct points stored in the array `pointsNoDuplicate[]` to the file `points.txt`. The format is the same as required for `hull.txt`, except that the coordinates of the point `lowerPoint` should not be written twice. For the same input points from Fig. 1, the file `points.txt` has the following content:

```
-7 -10
0  -10
-3  -9
-1  -6
5 -2
10 5
7  3
8  4
5  5
3  3
0  0
-2  1
0  8
-6  3
-10  0
```

The above format for `hull.txt` and `points.txt` is for clarity. In fact, Mathematica can read in the integer coordinates to form points correctly no matter how many of them are displayed on each line.

4.2 Command Execution

After installation of Mathematica, run the software. Click "File" on the upper left of the top menu bar, and select "New" in the opened menu, and then "Notebook" in the resulting submenu. See the screenshot below. This creates an input window (like the one named "Untitled-3" in the screenshot), called a ***notebook***, within which Mathematica commands will be executed and the results (including graphics) will be displayed.



Within this notebook window, you may type a command like

```
Graphics[Line[{{1, 1}, {3, 3}}]]
```

and then hit "Shift+Enter" (i.e., hit the key "Enter" while pressing the key "Shift"). This will execute the command. A line segment connecting the two points (1, 1) and (3, 3) will be drawn (below the command line). Similarly, executing the following two commands separately will plot a red disk (of default radius 0.01) and a blue disk (of radius 0.02) at the same point (2, 2).

```
Graphics[{Red, Point[{2, 2}]}]
Graphics[{Blue, PointSize[0.02], Point[{2, 2}]}]
```

You may plot the above line segment and the red disk together, by executing the following compound command:

```
Show[Graphics[Line[{{1, 1}, {3, 3}}]],
     Graphics[{Red, Point[{2, 2}]}],
     Axes -> True, AspectRatio -> Automatic, PlotRange -> All]
```

The generated figure will show the x- and y- axes since the `Axes` option is set to `true`. You may turn the option off by setting it to `false`, or simply remove "`Axes -> True,`" from the command. The `AspectRatio` option specifies the ratio of height to width. By setting it to `Automatic`, Mathematica will choose the ratio based on the actual plot values. The `PlotRange` option is set to prevent Mathematica from exercising its liberty to show only a portion of the plot that it "deems" important.

In this project, the **Mathematica commands for display will be given to you** in Section 4.3. However, in case you want to learn more about the software (especially its graphics), you may search in the following online reference for many more plotting commands:

http://reference.wolfram.com/language/

Unfortunately, the relevance of the returned links from a search directly within the above website is often low. Googling with a keyword/question plus "mathematica" would usually locate more relevant documents in this website.

4.3 Display Points and Convex Hull

In this project, the files `points.txt` and `hull.txt` generated by the method `GrahamScan()` are in the same folder that contains the `src` folder. To let Mathematica know where to look for these files, you need to set this folder containing the `src` folder as the working directory. Since Mathematica has its own string format for a directory, the easiest way to find the string representing the working directory is as follows.

- On the top menu bar of Mathematica, click "Insert".
- Select "File Path" in the opened menu.
- This opens up a window. Opens the file, say, `points.txt`.
- In the notebook window, the full path of the file will be shown as, for instance,

```
"C:\\Users\\jia\\Documents\\courses\\Com S 228\\java\\228 S15 Projects\\
points.txt"
```

  Note the double backslashes separating directory names in the above string. To obtain the string for the directory, remove `"\\points.txt"`.

- Back in the notebook, execute the `SetDirectory` command below via cut-and-paste:

```
SetDirectory["C:\\Users\\jia\\Documents\\courses\\Com S 228\\java\\228 S15
Projects"]
```

This will set up Mathematica's working directory. To draw the input points (in the file "points.txt") and the convex hull (vertices stored in "hull.txt"), simply **cut-and-paste the command sequence below into the notebook** and then hit "Shift+Enter":

```
Clear[points];
points = ReadList["points.txt", {Number, Number}];
Clear[hull];
hull = ReadList["hull.txt", {Number, Number}];
Show[Graphics[{Blue, Line[hull]}],
     Graphics[ListPlot[points, PlotStyle -> {PointSize[0.015], Red}]],
     Axes -> True, AspectRatio -> Automatic, PlotRange -> All]
```

The first command above clears the current content of the Mathematica variable `points`. The second command reads the input points from the file "`points.txt`" with the format `{Number,`

`Number}` specifying that every two numbers corresponding to the x- and y-coordinates of one point. Tabs and newline characters are ignored in the reading.

The third command clears the variable `hull`. The fourth command reads from the file "`hull.txt`" the coordinates of the vertices of the convex hull in counterclockwise order.

The fifth command `Show` draws both the convex hull and the input points. The result of drawing for the earlier example is already shown in Fig. 2. If you remove the option **"**`Axes -> True,`**"** from the compound command, the axes will be turned off. For example, the same convex hull and point set will be displayed in Fig. 3 next.

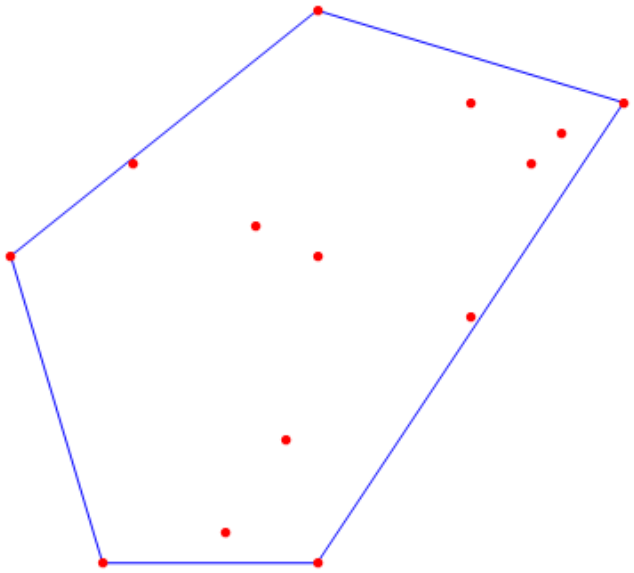


Fig. 3. Same convex hull from Fig. 2 displayed with no axes.

5. Submission

Write your classes in the `edu.iastate.cs228.hw4` package. **Turn in the zip file not your class files.** Please follow the guideline posted under Documents & Links on Blackboard Learn.

You are not required to submit any JUnit test cases. Nevertheless, you are encouraged to write JUnit tests for your code. Since these tests will not be submitted, feel free to share them with other students.

Include the Javadoc tag `@author` in every class source file you have made changes to. Your zip file should be named `Firstname_Lastname_HW4.zip`.