1 Install, Handin, Docs, and Demo

1. To install, type `cs0160_install <project_name>` into a shell. The script will create the appropriate directories and deposit stencil files into them.

2. To compile your code, type `make` in your project directory. To run your code and launch the visualizer, type `make run` in the same directory.

3. To run tests, run `make run_tests` from your project directory.

4. To hand in your project, go to the directory you wish to hand in, and type `cs0160_handin <project_name>` into a shell.

5. To run a demo of this project, type `cs0160_runDemo <project_name>` into a shell.

6. The NDS4 (you’ll be using some data structures from this package) documentation can be found here: [http://cs.brown.edu/courses/cs016/static/files/docs/nds4/index.html](http://cs.brown.edu/courses/cs016/static/files/docs/nds4/index.html). This is also linked off of the class website.

7. Remember to not include any identifying information in your hand in.

2 Introduction

2.1 Silly premise

As the next king of the Pride Lands, Simba is trying to figure out what land lies within his kingdom. His father Mufasa told him everything that the light touches lies within his hull, and the light shines down in points. With his anchor at Pride Rock, help Simba find the boundary of his kingdom (cuz ya know he just can’t wait to be king).

2.2 What?

In this assignment, you will implement an algorithm like we discussed in class to find the convex hull of a set of points and to update the hull as necessary when new points are added to the set. **Note that this project’s anchor point is calculated differently than in the lecture slides.** (See 4.1.1 for clarification.) You may optionally (for extra credit) also implement an Static Graham Scan, which computes the hull of a whole set of points at once, rather than updating the hull incrementally.
2.3 Why?

The purpose of this assignment is to let you:

- Gain experience with computational geometry.
- Design and implement a complicated algorithm.
- Learn how annoying edge cases are, and how to deal with them.
- Understand the use of an auxiliary data structure in an algorithm.

2.4 The Big Picture

When your project is complete, the visualizer will allow you to place points using a mouse click, display the convex hull of the placed points, clear the points, etc.

The visualizer has two parts: a graphical user interface and an instance of a class that can do convex hull computations. The GUI, when you interact with it, makes calls to your convex-hull-finding class, MyHullFinder, which calculates the current convex hull. The GUI then draws it.

3 Overview of Your Tasks

- You must implement methods in the MyHullFinder class. This class implements support.convexhull.ConvexHullFinder, i.e., it’s a class that performs incremental and static convex-hull computations. It has only four methods:

  - public void insertIncremental(HullPoint vertex)
  - public void clear()
  - public Iterator<Entry<Angle, HullPoint>> hull()
  - public void staticGrahamScan(Vector<HullPoint> vertices)

  and you need only implement the first two (the third one is done for you and the last one is for optional extra credit).

- Things to consider as you implement MyHullFinder:

  1. A ConvexHullFinder must maintain the convex hull of a set of points. (Note that it need not maintain all the points – just their convex hull – although it may be useful for debugging to maintain the point set as well.)
  2. The points in the convex hull will be stored in a CircularTree<Angle, HullPoint>, which is why the hull() method returns an iterator over Entry<Angle, HullPoint>s; we’ll explain this momentarily.
  3. The point set may also be reset to the empty set (using clear()).
3.1 Requirement Details

- You must implement an incremental Graham Scan algorithm. When a point is added to the set, your algorithm will determine whether it belongs in the convex hull, and if so, will then add it.

- Please do not change the method signatures of anything in the stencil code, or the visualizer won’t work. You can, however, add as many helper methods as you want. We encourage this.

- The convex hull of a point set is, in our algorithm, defined by a collection of vertices. If three or more of the input points are collinear, only the two outermost points should be on the convex hull returned by your hull() method.

- Your insertIncremental() method, which adds a new point to the point set and updates the convex hull, must run in $O(k \log n)$ time, where $k$ is the number of points removed from the hull to accommodate the new point, and $n$ is the number of points on the hull prior to the insertion.

- Our test cases will not add the same point twice (i.e. two points with identical coordinates), so you do not need to account for this edge case.

- Your algorithm will compute angles from an anchor point; this anchor point must be a point within the convex hull. We recommend the following (note that the anchor point for this project is calculated differently than the anchor point in the lecture slides):
  
  - When there are only one or two points in the hull, use the first point as the anchor.
  
  - When there are more than two points, use the average of the first three points added to the hull as your anchor. The angles for the first three points will all need to be recomputed, because they are measured relative to the anchor point. After you add your third point, the anchor point stays the same and any more points added to the hull will be ordered based off of this anchor point.

3.2 Extra Credit: Static Graham Scan

The staticGrahamScan method is extra credit: you are not required to implement it. The Static Graham Scan takes in a set of points and finds the convex hull for that set all at once, rather than determining the convex hull as points are added. This should operate in $O(n \log n)$ time. Note that extra credit will not be awarded if your method simply adds all the static vertices to the hull incrementally. You can recieve at most 10 points of extra credit.
4 Designing Your Code

4.1 Data Structures

The data structure you must use to store the hull is the CircularTree that we provide for you in the support.convexhull package.

Note that, unlike in the Heap project, you are not designing a generic data structure for Convex Hull. Rather, you are instantiating the CircularTree class that is itself generic, so it is your responsibility to fill in the concrete classes that the tree will use for its key K and value V. For the visualizer to properly display your hull, it is required that the key be of type Angle and the value or element stored be of type HullPoint.

You can assume that the before and after methods of the CircularTree data structure have runtimes of $O(\log n)$.

4.1.1 Angles

You will be storing points in the CircularTree using the Angle between the new point and the anchor point as the key. These Angles will keep them organized in counterclockwise order. (See the CircularTree and Angle classes in the Support Docs on the website for further clarification. Also note that because of how Java’s coordinate system works, the hull points will actually show up in clockwise order on the screen.) Things to consider:

- Initially, the anchor point will be the first point you add to the hull.
- This will change once you have three non-collinear points in the hull:
  - A new anchor point should be calculated by finding the midpoint of the three points currently in the hull.
  - Then you will need to update the Angles of the points already in the hull and reinsert them into the tree based on these new keys.
- To make a new Angle, you will instantiate a new instance of the Angle class, passing it the difference of the x-coordinates and the difference of the y-coordinates between the new point and the anchor point.

For thinking through test-cases, remember that the upper left-hand corner of the canvas is (0,0); this means that points which are lower on the screen will have a higher y-value than the points above them. Note that this is unlike the standard coordinate system upon which the Angle class is based. (Points on the screen follow the same x-coordinate convention as in the Angle class.) That means that a polygon that’s counterclockwise in ordinary coordinates is clockwise in screen coordinates, and vice versa. We recommend that you do all your work in ordinary Cartesian coordinates, so that your convex hull, if drawn from point to point (in the order the points show up when printed using the Visualizer – more on this later), would show up in clockwise order on the screen.
4.2 Orientation Test

As described above, when you add a new point to your CircularTree, it will be ordered according to the Angle calculated between itself and the anchor point. Using this ordering, you must be able to judge whether three ordered points are oriented clockwise or counterclockwise. (You can also see the Angle class in the Support Docs for more help on this topic.) When checking the orientation of three points, you should return an \texttt{int}—this will allow you to use the built-in $<$, $>$, $==$ etc. operators. For example, you could return $1$ to indicate counterclockwise, $-1$ to indicate clockwise, and $0$ to indicate collinear points.

Every time a point is added to the visualizer’s canvas, a HullPoint is created and passed into your insert method as a parameter. This method must determine whether your new point should be added to the hull. The class \texttt{support.convexhull.HullPoint} provides the coordinates that your orientation tester should be using when performing tests on points.

5 Visualizer

We are providing you with a visualizer that will display a set of points and the convex hull you have computed for those points. You will be able to load pre-defined sets of points, and will also be able to create these sets directly in the visualizer, and save them to a file for later use. You can also print the set of points, and the set of hull points, to your console. This can be useful in determining what test cases you should write—see Section 6 for more information.

For the visualizer to work, you must correctly implement the Graham Scan algorithm in the MyHullFinder class. When you run the visualizer, a window will appear with a column of buttons and a graphical background. Left click on the background to add points. As you add points, the hull you’ve computed will be drawn with red lines, as long as the “incremental” radio-button at the bottom is selected. If “Static Graham Scan” is selected, then the hull is only displayed when you press the “Solve” button.

When a point is added (in incremental mode), it is classified and colored as follows:

- **Red:** The point is outside the current convex hull and gets added to the convex hull.
- **Blue:** The point is inside or on the current convex hull, so nothing changes.
- **Green:** Error: the point is invalid. This can be caused by internal Java errors (e.g. \texttt{NullPointerException}s), or by violating the requirements of this assignment. A fully functional project should not have any green points.

In static mode, the points are classified the same way, but only after the “Solve” button is pressed. Initially they are all drawn blue.

In addition, the visualizer has these features:

- You can add a point by specifying its $x$ and $y$ coordinates in the boxes on the upper left side of the window. This is especially useful for checking special cases.
• Right-clicking anywhere on the background will display the $x$ and $y$ coordinates of the click-point in the boxes on the right.

• If you click on “Clear,” it will remove all points currently in the drawing and allow you to start from scratch. (Be careful – this can’t be undone!)

• “Load File” allows you to input a pre-made test case to the visualizer. When you select “Load File,” a dialog box will appear, allowing you to select a file. To make your own files, you must adhere to a specific format. The file must consist of pairs of $x$ and $y$ pixel coordinates representing the points $1$, formatted as in the following example:

\begin{verbatim}
100  100
100   50
234  453
  74  222
\end{verbatim}

Loading this file will create 4 points, with the coordinates (100, 100), (100, 50) . . . . There must be only one pair of numbers per line, and there must be one pair of numbers on every line. (That is, no blank lines are allowed in the file.) Creating files is highly encouraged, as it makes repeat testing of special cases very simple.

6 Testing

Just like in Heap, we’re requiring you to write your own JUnit tests for this project. You should write these tests in MyHullFinderTest. We’re not requiring a specific number of test cases, but rather a well thought-out set of cases which demonstrate that your project is fully functional, no matter what input it is run on. Make sure that your tests are modular, this is, make sure you’re only testing one component of your Hull at a time. Make sure to test completely as well, meaning that you should test all methods of your MyHullFinder class and every code path within those methods. We’ll be rigorously testing your convex hull with the same test system, so make sure you think of everything. Demonstration of well thought-out test cases is a key component of your grade.

In the MyHullFinderTest file, you will find one test already written for you - testAddIncremental(). As in Heap you will need to add tests to this file in the same format. You should also be commenting thoroughly on what each test is testing and in the header comment give an overview of your tests and why you believe you have tested all cases.

A reminder about the collaboration policy and testing: You are allowed to talk with other students about the conditions you want to test for, but you may not share test cases.

\footnote{The point (0,0) is in the upper left corner of the canvas, and the $y$ coordinate increases as you move down, while $x$ increases as you move right. Limit your values to lie between 0 and 500, or the points may appear offscreen, which could confuse you.}
This includes sharing just to cross-check against other students’ tests, even if you don’t turn them in.

7 README

The README for this project has some specific points that you need to address. Make a text file entitled “README.txt” which should be saved in the same directory as your project; it will be handed in along with your code. In addition to the general items in the README guide on the website, please address the following questions. Each of the questions should be answered in about 2-4 lines, unless otherwise stated.

- Don’t include any identifying information such as your name, login or banner ID.
- Give an overview of your design for the project.
- Describe, briefly, any special cases which your insertion algorithm had to handle and how you chose to handle them.
- Write here anything you did not mention in the comments of your testing files that you feel is notable.
- Mention any bugs that you have been unable to solve and what steps you have taken to resolve them.
- If you have anything else noteworthy about your program you haven’t mentioned yet, talk about it here (write as much or as little as you like).

8 Using Eclipse

If you would like to use eclipse, you may certainly do so. In order to set up your project and make eclipse work with the support code, you’ll need to do the following:

- Select File → New → Java Project
  - Enter “<project_name>”, lowercased, for example ”heap”, for the project name.
  - Un-check the box saying “Use default location” and in the “Location” box, enter /gpfs/main/home/<your-login>/course/cs0160/<project_name>
  - Click “next”
  - Under the “libraries” tab choose “Add External JARs...”
  - Select /course/cs0160/lib/cs0160.jar
  - Select /course/cs0160/lib/nds4/nds4.jar
  - Select /course/cs0160/lib/junit-4.12.jar
– Select /course/cs0160/lib/hamcrest-core-1.3.jar
– Click “Finish”
– If it isn’t already made for you, use File → New → Source Folder to create a new source folder in your new project named “src”
– Use File → New → Package to create a new package in your new source folder named, for example heap, and move all the stencil java files into this package. Ignore any errors.

• Right-click on App.java and select Run As → Java Application. Now you can run your program by pressing the green play button at the top of your screen and selecting “Java application” if prompted
• To run the tests in eclipse, you can right-click on TestRunner.java and click Run As → Java Application.
• Alternatively, if you want to run one test file, you can right-click on that file and select Run As → Junit test
• To configure your Eclipse projects to run over FastX or SSH, follow these setup steps
  – Right click on the package icon next to the project name. Go to properities.
  – Go to Run/Debug Settings, select the main window App and click Edit.
  – Go to the arguments tab and, and enter -Dprism.order=sw in the VM arguments block
  – Hit Apply and OK
  – You should be all set to work on this project remotely with Eclipse. Make sure to do this for each new project.

9 Working from Home

If you wish to work locally, you should be able to set up the project in Eclipse on your home computer by following the directions that are listed in the ‘Using Eclipse’ section of the handout; however, in order for the project to compile correctly and access the pictures, you will have to do the following:

• Go to the App class and find the start method.
• Inside the start method, the first line will set a variable called imagePath to the directory that contains pictures that convexhull accesses.
• Reset this imagePath to a local directory on your computer that contains .jpg files. For example, if you wanted to access a photo in your Pictures directory, replace “/course/cs0160/lib/convexhull-images/” with “/filepath/to/your/picture/Pictures/”
• Repeat this same step for MyHullFinderTest.java by updating the _imagePath variable.

• After you are done working on your local computer, make sure you reset the imagepath back to “/course/cs0160/lib/convexhull-images/” If you don’t, your program will not run on the department machines and you will lose points.

Note that you will need to have the support libraries listed in that section copied onto your local computer in order to reference them – you can copy the libraries from department machines using SFTP or SCP commands, provided that you have SSH set up. If you are unfamiliar with using SFTP or SCP commands, please see a TA or Sunlab Consultant for assistance.

As always, be sure to test your code on department machines before turning it in!

10 What to Hand In

1. All code installed by the install script.

2. A “README.txt” pointing out any bugs or other problems in your code (or an assertion that there aren’t any), as well as describing any significant design choices. For this README, make sure to also include everything discussed in Section 7 and in Section 6.

3. A set of test cases which demonstrate the full functionality of your project. (Don’t forget to include descriptions of your test cases in the README, as described in Section 6)