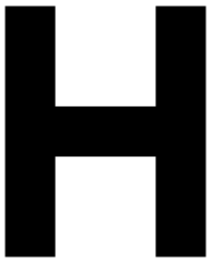
Programming Assignment 2 (100 Points)

Due: Thursday, October 6 by 11:59pm



PA2 is a two part assignment. The first part is designing a program which prints out the smallest positive number entered and the largest negative number entered. For instance, if the user enters 5 15 -13 -3, the program would output 5 as the smallest positive number entered and -3 as the largest negative number entered. The second part involves creating a graphical user interface (GUI) which responds to mouse clicks and mouse drags to create and manipulate a block H, as seen on the right.

README (10 points)

Starting from this assignment, every assignment will require a README to be turned in along with whichever programs the PA requires. You are required to provide a text file named **README,** NOT Readme.txt, NOT README.pdf, and NOT README.doc, with your assignment in your pa2 directory. There should be no file extension after the file name "**README**". How would you create this file? Use vim: vim README

Your README should include the following sections:

Program Description :

Write this part of your README as if it was intended for someone who is computer illiterate (like a great grandmother). Do not assume your reader is a computer science major. The more detailed the explanation, the more points you will receive.

In future PAs we will simply ask you to explain how your programs work, what types of input they take and what types of output they have. However, for this PA, simply explain the specific topics mentioned in the following paragraph.

Explain how the output of PosNeg changes when the input is either all positive numbers, all negative numbers, a mix of positive/negative numbers, zeros, or no numbers entered at all. Also, explain what fun things the user can do in the program DraggingBlockH.

Short Response : Answer the following questions:

Note: Please cite any sources used for any of the short response answers. You can simply write a URL. Also, it is not necessary to re-write the questions in the README, but you can if you would like to.

1. Why is it considered an integrity violation if a student submits code copied from someone/somewhere else?
2. What is the Linux command to rename a file? What is the command to copy a file? Give examples.
3. What are three ways to enter insert mode in vim?
4. What is a method?
5. What happens when you type ":split" in command mode in vim?
6. Imagine that you're using a terminal. When you type "ls", the only files listed are Meow.java and Dog.java. You realize you want to edit something inside Dog.java, so you need to open the file. You type "vim Dog.j". What key can you press so that the filename is completely written out and reads "vim Dog.java"?

STYLE (20 points) [Much of this comes from the Google Java Style Guide]

From now on, we'll start being more critical of the style used in each programming assignment. Each bullet point statement is worth some amount of the 20 points so please carefully read and implement each of them.

- **Write reasonable in-line comments to make your code clear and readable**. This is so that if you go back to your code in a couple months, it'll still be understandable to you or someone else maintaining your code. You don't need to comment every line, but you should comment sections of code that you struggled with writing so that you understand your logic later. Also, you should comment sections of code that are not immediately obvious as to their purpose. Example: Say you increment a variable (count++). A poor comment would say "incrementing variable". A better comment would explain the purpose of the incrementation so it could be "keeping track of how many odd numbers have been encountered". Also, you can either put the comments above or on the line you're referring to. Just make sure to keep the location consistent and don't switch between writing it above/in-line once you've picked a location.

```
count++; // commenting commenting commenting
```

or

```
// commenting commenting commenting
count++;
```

- Write **class header comments** and **method/constructor header comments** to describe the purpose of your program and methods. Also, each file should have a **file header comment** at the top of the file (including your README). In PA1, we endorsed a different type of method/class header. **For PA2 and all the following assignments, we want to see only a Javadoc style of**

commenting for the class and method/constructor headers. Note these start with **/**** See below for examples on those.

(Note: coloring may differ depending on what color scheme your vim has):

Example file header comment:

```
/*
 * Name: Jane-Joe Student
 * Login: cs11fXXX  <<< --- Use your cs11f course-specific account name
 * Date: Month Day, Year of last edit date
 * File:  Name of this file, for example: OddEven.java
 * Sources of Help: ... (for example: names of people, books, websites, etc.)
 *
 * Write a program description here. Keep this short but generally explain what
 * your program does
 */
```

Example class header comment:

```
/**
 * Description of the class. Keep this to one to two lines.
 */
```

Example method/constructor header comment:

```
/**
 * Description of the method or constructor, including functionality
 *
 * @param param1 State what this parameter represents.
 * @param param2 State what this parameter represents.
 * @return Specify what the return value represents.
 */
```

If you have multiple parameters, then use multiple `@param` tags.
If there are no parameters, then there should be no `@param` tag.
If there is no return value, then there should be no `@return` tag.

- **Use reasonable variable names**. If you have a variable, don't call it "variable", "var", "thing", or anything generic. Also, avoid one letter variable names as they don't describe variables well either. The better your variable name, the less commenting you'll have to do to explain it and also you won't confuse yourself later on.

- Use of blank lines around logical chunks of code makes your code much easier to read and debug. For instance, put a blank line between methods. Also, a blank line between if and else statements/blocks helps too. It's up to you to decide how many blank spaces to use but remember too many is excessive (example: 6 lines between methods) and too few makes the code look like a

large chunk of characters and it becomes hard to read. Also, as you can see in the example below, comments should factor into spacing as well. Essentially, make sure your code is readable and doesn't look like a wall of text.

---

example

GOOD:

```
// commenting commenting commenting commenting
// commenting commenting commenting commenting
// commenting commenting commenting commenting
if (meow) {

  // everytime meow is true, we increment count
  count++;

  // commenting about bunnies
  bunnies--;

} else {

  count--;

}
```

BAD:

```
if(meow){
  //commenting commenting commenting commenting
    //commenting commenting commenting commenting
  //commenting commenting commenting commenting
      //everytime meow is true, it increments
   count++;
    //commenting
      bunnies--;
  }
    else{
count--;  }
```

- Keep all lines **less than 80 characters** (Note: this applies to README too).
- **Use 2-4 spaces (not tabs) for each level of indentation**. What is a level? Every time you have to go to the right. The if/else code above shows this well. The "if(meow){" line is one level of indentation. The next level of indentation is the comment "//commenting commenting commenting commenting". As you can see, the code is using 2 spaces per each level of indentation. Google mandates 2 spaces only!

- Also, make sure **each level of indentation lines up evenly**. As you can see in both if/else examples above, the closing bracket for the if statement lines up directly underneath the i in the if statement.
- Note: You can put the opening curly bracket { either on the same line as the if statement (as shown in the example code above), or on its own separate line immediately after the if statement. Either is good; just keep it consistent.
- Every time you open a new block of code (use a '{'), indent a level. Go back to the previous level of indenting when you close the block (use a '}'). See code example above.
- Do not use magic numbers or hard-coded numbers other than 0, 1, and -1. Use constants in place of numbers. Piazza post for more detail (thanks Jesse Q)

CORRECTNESS (70 points)

Setting up your pa2 directory:

You will need to create a new directory named **pa2** in your cs11f home directory on ieng6.ucsd.edu (file server used by the lab workstations). The '$' character represents the command line prompt. Type the commands that appear in bold after the command line prompt.

```
$ cd
$ mkdir pa2
```

The first command (cd) changes your current directory to your home directory. cd stands for **c**hange **d**irectory. By default, if you do not specify a directory to change to, the command will put you in your home directory.

The second command (mkdir pa2) makes a new directory named pa2. This new directory will be in your home directory since you did a cd beforehand.

Now type

```
$ cd pa2
```

This will change your current working directory to the new pa2 directory you just created. All files associated with this programming assignment must be placed in this directory. And in general, you should do all your work on this programming assignment in this pa2 directory.

After executing the cd pa2 command, copy over required jar files from the public directory (note the `.` as the last argument)

```
$ cp ~/../public/objectdraw.jar .
$ cp ~/../public/Acme.jar .
```

`(~)` is your home directory. `(..)` is the relative parent directory. `(.)` is the relative current directory. So this path starts at your home directory (~) no matter what directory you are currently in, then moves up one directory (`..`), then down into the class public directory (`public`), and accesses the named file (for example, Acme.jar). That file is copied (the `cp` command) to your current working directory (`.`). To change your current working directory back to your home directory, you can type **cd** as you did before.

Program 1 - PosNeg.java

Write a Java application with a main() method to find the smallest positive integer and largest negative integer entered at the terminal by the user. All of your code can be in main(). See PA1 for an example of the definition of main(). No objectdraw and No Acme.MainFrame for this program. We are not running a graphic application. We are just creating a simple standalone application.

Use a **Scanner** object to read user input from standard input (**System.in**).
The Scanner Javadocs: http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

The basic framework to use a Scanner is:

```
// Construct a Scanner that produces values scanned from standard input
Scanner input = new Scanner(System.in);

while (input.hasNext()) {         // while there is more input (user has not hit EOF)
  number = input.nextInt();       // read next integer
    …
}
```

Make sure your prompt and output messages exactly match the sample prompts and messages word-for-word. See below for examples.

Any valid integer may be entered. We will only enter/test with valid integers.

Numbers entered may be repeated.

There is no need to handle invalid input (for example, non-numeric input). We will not be testing for it. We will only be testing for valid positive and negative integers and 0, of course. We will handle invalid input in future assignments.

If there is an EOF (end-of-file) keyboard sequence [<Ctrl>+D on Unix/Mac or <Ctrl>+Z on Windows] as the first character sequence on a line and then <Enter>, that indicates no more input to the program (^D represents the user entering <Ctrl>+D). As can be seen in the expected output section below, make sure your program responds appropriately to this keyboard sequence.

Handle any number of entries (special case of a zero entered (zero is neither positive nor negative), no positive numbers, no negative numbers, or no numbers entered).

You **cannot** use any data structure (like an array, ArrayList, Vector, List, etc.) to hold the input. Just use two variables to keep track of the smallest positive number and largest negative number entered and any other variables you may need. Use logic to determine if and how the variables are assigned.

Only valid integers will be tested. You do not need to do any error checking for invalid input.

Compiling

To compile your code, use the following:

```
$ javac PosNeg.java
```

Running

To execute your application, use the following:

```
$ java PosNeg
```

Expected Output (user input in **BOLD**)(we will use more/other test cases for grading):

Example 1:

```
java PosNeg
Enter a series of integers; EOF to Quit
99 -5 99
^D
Smallest positive number entered was 99
Largest negative number entered was -5
```

Example 2:

```
java PosNeg
Enter a series of integers; EOF to Quit
3
^D
Smallest positive number entered was 3
```

Example 3:

```
java PosNeg
Enter a series of integers; EOF to Quit
8 0
5 -19
55
13
-2
^D
Smallest positive number entered was 5
Largest negative number entered was -2
```

Example 4:

```
java PosNeg
Enter a series of integers; EOF to Quit
44 44
^D
Smallest positive number entered was 44
```

Example 5:

```
java PosNeg
Enter a series of integers; EOF to Quit
^D
No positive or negative numbers entered
```

Example 6:

```
java PosNeg
Enter a series of integers; EOF to Quit
44 0 46 48
^D
Smallest positive number entered was 44
```

Example 7:

```
java PosNeg
Enter a series of integers; EOF to Quit
-44 0 -46 -48
^D
Largest negative number entered was -44
```

Example 8:

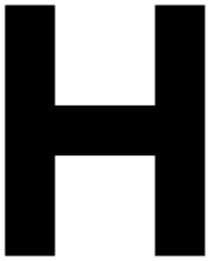
```
java PosNeg
Enter a series of integers; EOF to Quit
0 0
^D
No positive or negative numbers entered
```

Program 2 - DraggingBlockH.java

Write a Java application that does the following (use the textbook examples as guides):

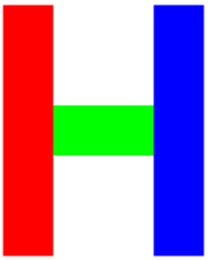
When the application first starts up, display instructions on two lines using Text objects. See the screenshot below for exact wording of the instructions. See the constants later in this section for the offset where these instructions need to be displayed.

Click to display a Block H centered at the mouse click.
Mouse press in any part of the image and drag to move image around.



On a mouse click, hide the instructions, and if there is no block H already drawn then draw a block H with the center bar of the block H centered at the point of the mouse click. The block H is made up of 3 FilledRects - a center bar/rect, a left bar/rect, and a right bar/rect. See the screenshots and constants defined below.

Only one block H figure should be drawn no matter how many mouse clicks occur.



On a mouse press, if the block H has already been drawn, check if the point of the mouse press is anywhere in the block H (in other words, you are grabbing the block H). You can grab any one of the left, center, or right bars. Of course, on a mouse release you are no longer grabbing it. Also if on a mouse press if you are grabbing any part of the block H, the different parts of the block H should change color - the left bar should turn RED, the center bar should turn GREEN, and the right bar should turn BLUE.

On a mouse drag, if the block H figure has been grabbed (mouse press is inside the figure's boundaries), drag the multi-colored block H around with the mouse drag. You will need to move all 3 parts of the figure together. See the examples in the textbook. The figure should continue to look like a block H during and after the drag.

Once the mouse button has been released, then the block H should revert back to being BLACK.

On a mouse exit (when the mouse pointer exits the canvas), if the block H had been drawn, remove the entire figure from the canvas and reset the logic you are using to keep track of whether a block H has been drawn or not (you can do this in many ways).

On a mouse enter (when the mouse pointer (re)enters the canvas), show the instructions again.

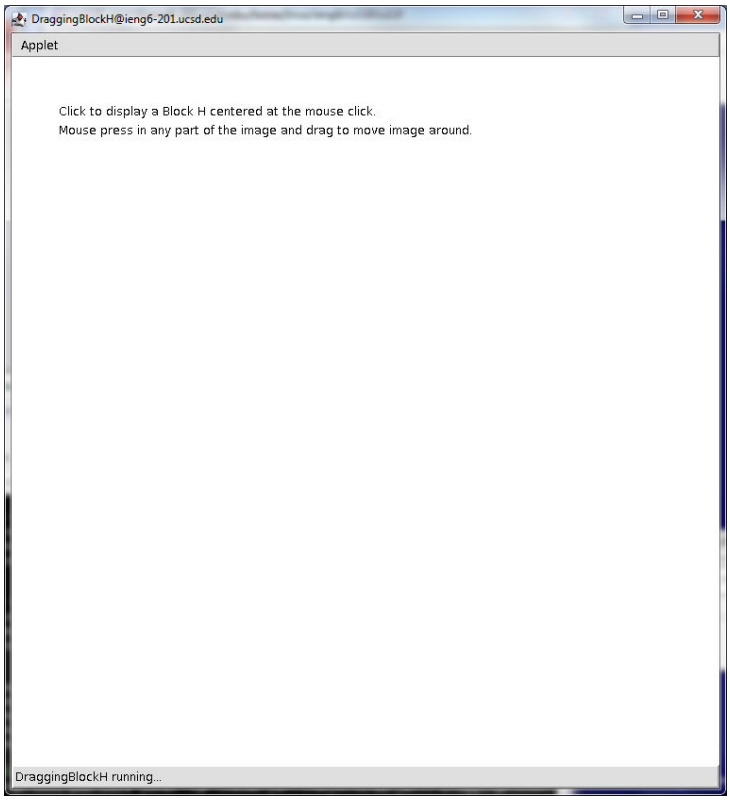
Many of the checks in the event handlers above are to prevent blindly trying to remove non-existing objects from the canvas, prevent displaying more than one figure with multiple mouse clicks, checking if you are trying to grab a non-existing figure, etc. Without these checks you will likely get exceptions thrown in the terminal window. **No exceptions should be thrown!**

You **MUST** use these instance variable names and constants (for our testing purposes):

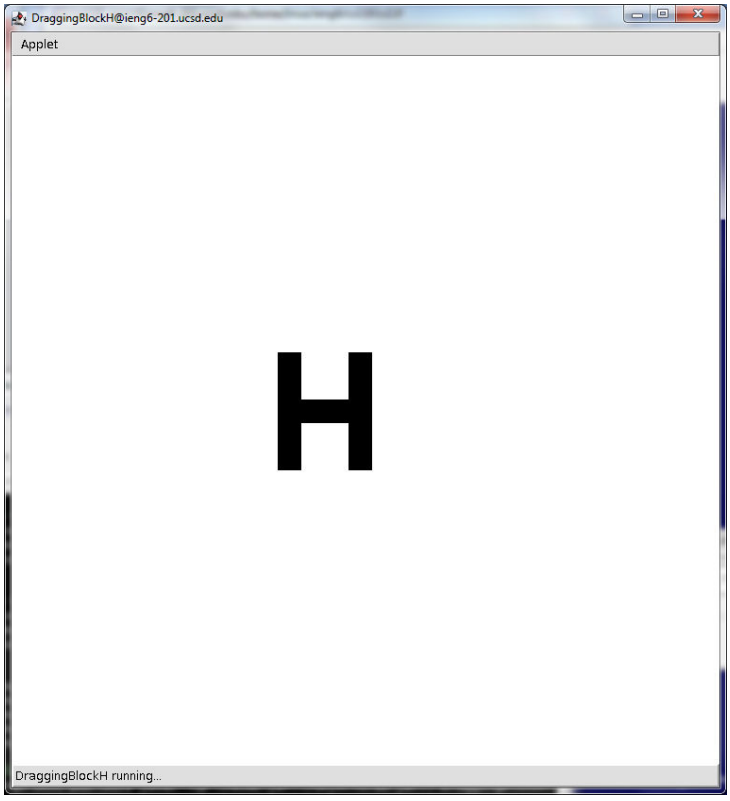
```
private Text instr1 = null;
private Text instr2 = null;

private FilledRect leftBar = null;
private FilledRect rightBar = null;
private FilledRect centerBar = null;

private static final int INSTR1_X = 50;
private static final int INSTR1_Y = 50;
private static final int INSTR2_X = INSTR1_X;
private static final int INSTR2_Y = INSTR1_Y + 20;

private static final double CENTER_BAR_WIDTH = 50;
private static final double HALF_CENTER_BAR_WIDTH = CENTER_BAR_WIDTH / 2;
private static final double CENTER_BAR_HEIGHT = 25;
private static final double HALF_CENTER_BAR_HEIGHT = CENTER_BAR_HEIGHT / 2;

private static final double SIDE_BARS_WIDTH = 25;
private static final double SIDE_BARS_HEIGHT = 125;
private static final double HALF_SIDE_BARS_HEIGHT = SIDE_BARS_HEIGHT / 2;
```

You will most likely need additional instance variables, but these must be used - **instr1**, **instr2**, **leftBar**, **rightBar**, **centerBar**

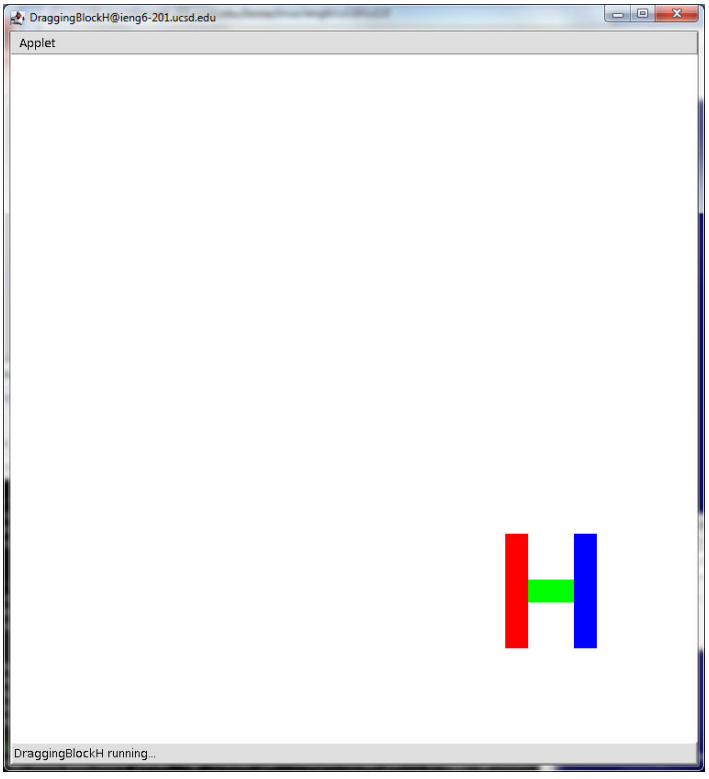
Here are some example screenshots:



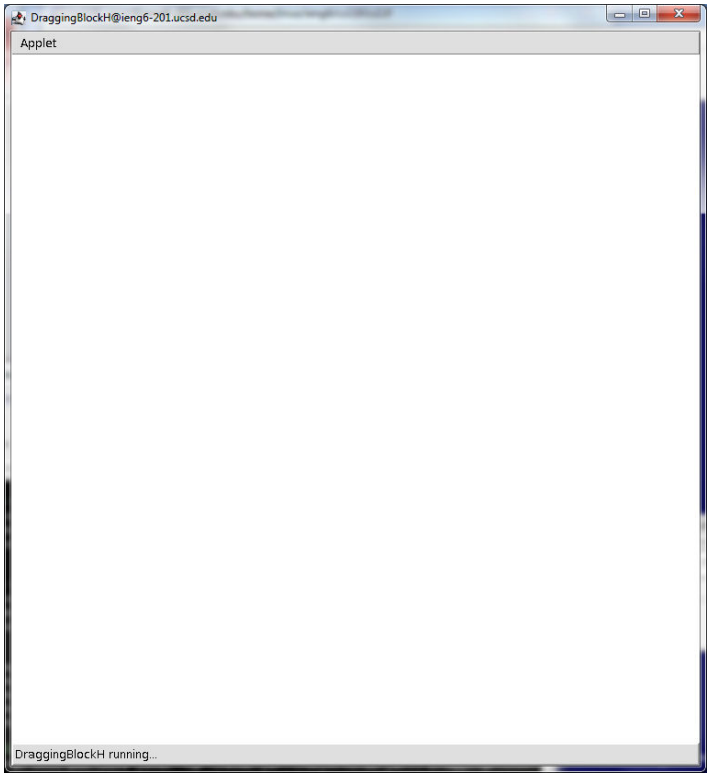
At start-up.



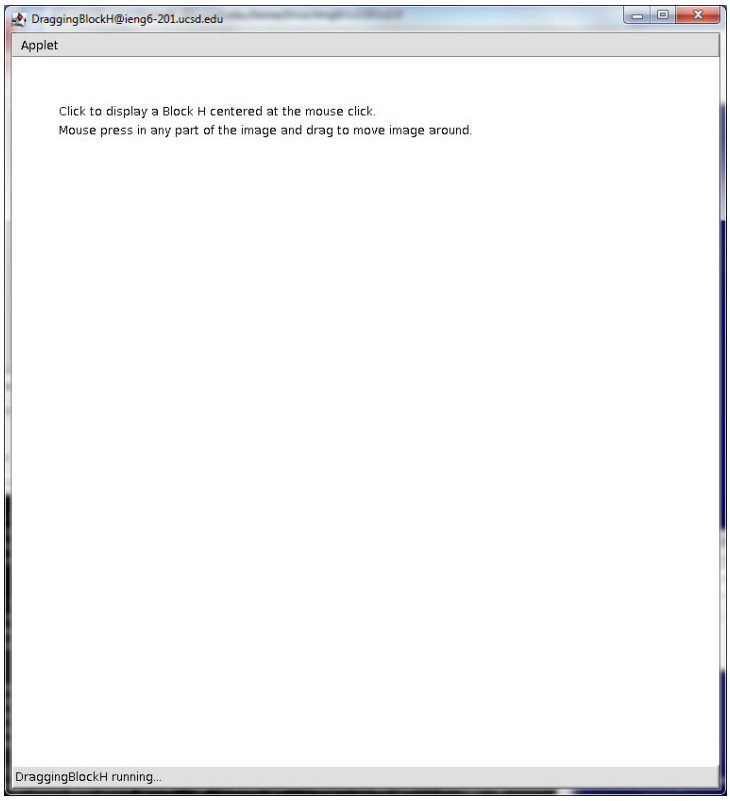
On mouse click.



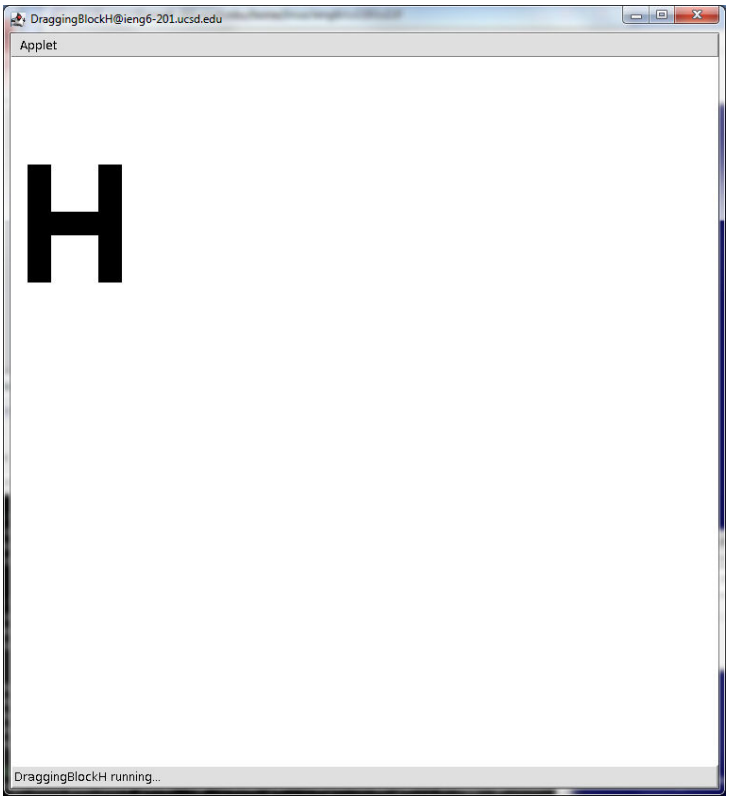
Dragging the block H around.



On mouse exit.



On mouse (re)enter.



On mouse click.

Ideas for How to Get Started

- Read the online documentation for the Text and FilledRect objects from the objectdraw library
- Read Chapters 4-6 in the textbook and create the DraggingBlockH.java file

  ```
  public class DraggingBlockH extends WindowController
  ```
- Be sure to use the instance variables **instr1**, **instr2**, **leftBar**, **rightBar**, and **centerBar**

Making DraggingBlockH into an Application

This is missing from the textbook examples. Don't forget to add the following code to DraggingBlockH.java. Since we want to run this program as an application which uses objectdraw graphics and uses the WindowController class, we need to add a main() that runs this program in an Acme.MainFrame.

```
public static void main( String[] args ) {
    new Acme.MainFrame( new DraggingBlockH(), args, FRAME_WIDTH, FRAME_HEIGHT );
}
```

This code allows DraggingBlockH to run as an application. Define final constants for the initial number of pixels width and height for the frame/window and set both to 750.

Compiling

To compile your code, use the following:

```
$ javac -cp ./Acme.jar:./objectdraw.jar:. DraggingBlockH.java
```

Running

To run your program, use the following:

```
$ java -cp ./Acme.jar:./objectdraw.jar:. DraggingBlockH
```

Extra Credit (5 points)

Write a Java program with a main() method to find the two smallest positive distinct integers and two largest negative distinct integers entered in the terminal window by the user. All of your code can be in main(). Before you start the extra credit, make sure that PosNeg.java is running perfectly. See PA1 for an example of the definition of main.

Before starting, make a copy of your PosNeg.java into a file named EC_PosNeg.java:

```
$ cd ~/pa2
$ cp PosNeg.java EC_PosNeg.java
```

By using the previous unix commands, you are creating a copy of PosNeg.java and naming that copy EC_PosNeg.java. BOTH FILES ARE NEEDED IF YOU ATTEMPT THE EXTRA CREDIT.

After this, make sure to change the class definition in EC_PosNeg.java to the new class name. ie:

```
public class EC_PosNeg {
      …
}
```

Make sure your prompt and output messages exactly match the sample prompts and messages word-for-word. See below for examples.

Any valid integer may be entered. We will only enter valid integers. Numbers entered may be repeated.

There is no need to handle invalid input (for example, non-numeric input). We will not be testing for it. We will only be testing for valid positive and negative integers.

If there is an EOF (end-of-file) keyboard sequence [<Ctrl>+D on Unix/Mac or <Ctrl>+Z on Windows] as the first character sequence on a line and then <Enter>, that indicates no more input to the program (^D represents the user entering <Ctrl>+D). As can be seen in the expected output section below, make sure your program responds appropriately to this keyboard sequence.

Handle any number of entries.

You cannot use any data structure (like an array, ArrayList, Vector, List, etc.) to hold the input and then sort them. Just use four variables to keep track of the smallest and second smallest distinct positive values entered, and largest and second largest distinct negative values entered, and how many zeros were entered, and any other variables you may need.

Only valid integers will be tested. You do not need to do any error checking for invalid input (like letters).

What is a distinct value? For example, in the series of values 1, 2, 1, 1 there are only two distinct values (1 and 2). In the series of values -1, -1, -1, -1 there is only one distinct value (-1). Thus, for the first example the smallest positive number is 1 and the second smallest positive number is 2, but there are no largest and second largest negative numbers and no zeros. Similarly for the second example, the largest negative number is -1, but there is no second largest negative number, no smallest and second smallest positive numbers, and no zeros.

Expected Output (user input in **BOLD**)(we will use more/other test cases for grading):

Example 1:

```
java EC_PosNeg
Enter a series of integers; EOF to Quit
99 -5 99
^D
Smallest distinct positive number entered was 99
No zeros were entered
Largest distinct negative number entered was -5
```

Example 2:

```
java EC_PosNeg
Enter a series of integers; EOF to Quit
3
^D
Smallest distinct positive number entered was 3
No zeros were entered
```

Example 3:

```
java EC_PosNeg
Enter a series of integers; EOF to Quit
8 0
5 -19
55
13
-2
^D
Smallest distinct positive number entered was 5
Second smallest distinct positive number entered was 8
There was 1 zero entered                                    # Note plurality
Largest distinct negative number entered was -2
Second largest distinct negative number entered was -19
```

Example 4:

```
java EC_PosNeg
Enter a series of integers; EOF to Quit
-44 -44
^D
No zeros were entered
Largest distinct negative number entered was -44
```

Example 5:

```
java EC_PosNeg
Enter a series of integers; EOF to Quit
^D
No positive or negative numbers entered
No zeros were entered
```

Example 6:

```
java EC_PosNeg
Enter a series of integers; EOF to Quit
44 0 -22 0 -4
^D
Smallest distinct positive number entered was 44
There were 2 zeros entered                                  # Note plurality
Largest distinct negative number entered was -4
Second largest distinct negative number entered was -22
```

Example 7:

```
java EC_PosNeg
Enter a series of integers; EOF to Quit
2 -2 4 -4 0 4 0 2 -2 -4 0
^D
Smallest distinct positive number entered was 2
Second smallest distinct positive number entered was 4
There were 3 zeros entered                                  # Note plurality
Largest distinct negative number entered was -2
Second largest distinct negative number entered was -4
```

Note the plurality of the output statements for the number of zeros.

Turnin

Always recompile and run your program right before turning it in, just in case you commented out some code by mistake.

Do not have any other .java source files in your pa2 directory other than those that are part of this assignment!

To turn-in your code, navigate to your home directory and use the following command:

```
$ cse11turnin pa2
```

You may turn in your programming assignment as many times as you like. The last submission you turn in before the deadline is the one that we will collect.

Verify

To verify a previously turned in assignment,

```
$ cse11verify pa2
```

If you are unsure if your program has been turned in, use the verify command. **We will not take any late files you forgot to turn in**. Verify will help you check which files you have successfully submitted. It is your responsibility to make sure you properly turned in your assignment. Also, make sure that you don't misspell any file names.

Files to be collected for grading:

- DraggingBlockH.java
- PosNeg.java
- README

Additional files to be collected for Extra Credit:

- EC_PosNeg.java

NO LATE ASSIGNMENTS ACCEPTED.

DO NOT EMAIL US YOUR ASSIGNMENT!

Start Early and Start Often