Do the Final exam, submitting both PDF answers and code you have written:

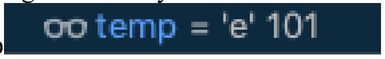
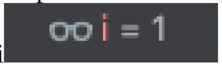
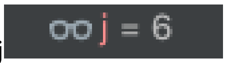
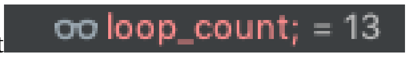
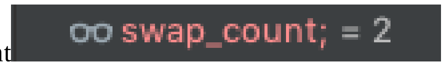
- Get the latest PROG1400 repo. Minor changes were made to allow direct use for this final.
- Add a **PROG1400/Final** directory for answers in your Repo [project Final, java packages Q1, Q2, … etc…]
- Questions answered as text, are placed in this file, made PDF, and saved in the final project directory with your final code.
- Also upload your PDF to D2L, showing you finished the Final.

You have access to all of your git repo, and mine for any questions. You also have access to the internet, but be mindful of the poor solutions you can get in the internet.

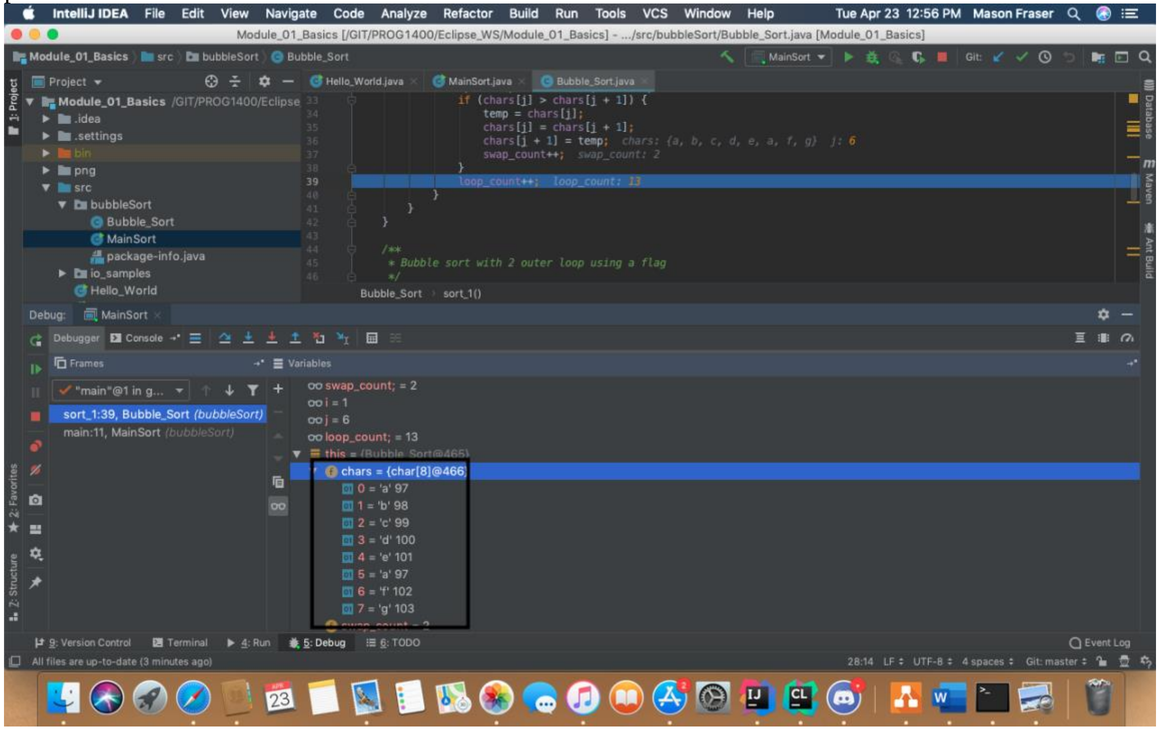
MainSort Main line snippet

```
public class MainSort {

    public static void main (String[] args) {
        Bubble_Sort b1 = new Bubble_Sort("abcdefga");
        b1.sort_1();
        b1.printAll();

        Bubble_Sort b2 = new Bubble_Sort("abcdefga");
        b2.sort_2();
        b2.printAll();
    }
}
```

1) Find The BubbleSort Java Code in the PROG1400 repo, and run the MainSort Main line [Located at … "**PROG1400\Eclipse_WS\Module-01-Basics\src\bubbleSort**"]
Run the program several times, changing the string to sort, use the debugger to "see" how the sort is taking place.
   (a) **[10%]** Show a debugger screen snippet during the sort of "abcdefga". Use snippet highlight or circle to indicate the value of the loop variables during the swap. Use any iteration of either sort….I'm looking to see that you know how to see the algorithm at work using the debugger.

   1. Show temp 
   oo temp = 'e' 101
   //Forgot to add watcher to bigscreen.
   2. Show i 
   oo i = 1
   3. Show j 
   oo j = 6
   4. Show loop-count 
   oo loop_count; = 13
   5. Show swap-count 
   oo swap_count; = 2

6. Show the array contents at any mid-sort point



```
                if (chars[j] > chars[j + 1]) {
                    temp = chars[j];
                    chars[j] = chars[j + 1];
                    chars[j + 1] = temp;  chars: {a, b, c, d, e, a, f, g}  j: 6
                    swap_count++;  swap_count: 2
                }
                loop_count++;  loop_count: 13
            }
        }
    }

    /**
     * Bubble sort with 2 outer loop using a flag
     */
```

```
sort_1:39, Bubble_Sort (bubbleSort)
main:11, MainSort (bubbleSort)
```

```
swap_count; = 2
i = 1
j = 6
loop_count; = 13
this = {Bubble_Sort@465}
  chars = {char[8]@466}
    0 = 'a' 97
    1 = 'b' 98
    2 = 'c' 99
    3 = 'd' 100
    4 = 'e' 101
    5 = 'a' 97
    6 = 'f' 102
    7 = 'g' 103
```

(b) **[5%]** Explain, in text paragraph form, the ***algorithm*** difference between sort-1, and sort-2 (the difference between how sort-1and sort-2 are coded, and their use of flags).

sort_1, if it finds that the ascii value of the current position in the array is larger than the following positions ascii value, it stores the current position temporarily and swaps them with each other, putting it further down the list.

sort_2, sets a bool to true, and while its true a loop runs, inside the loop the bool gets set to false, and the loop runs through every character in the array once, if the if condition is met the characters do the same as above, store in a temp and swap as before, the flag gets set back to true and the process repeats until the if condition is no longer met within the loop, E.g. the letters are sorted.

The main difference here is that instead of comparing every letter to every other letter like sort_1, it only checks to see if the next letter is smaller value wise compared to its next immediate letter

(c) **[5%]** Explain, in text paragraph form, the ***performance*** difference between sort-1, and sort-2. Would one be faster than the other…all the time?

sort_2 will be faster overall since it doesn't need to compare unless a char is flagged, where sort_1 will always compare all values to all values.

sort_2 will always be more efficient than sort_1
unless only 1 character is out of place at the very end of the string.

1. Sort "abcdefg" with each sort…what's the difference?

   sort_1 has to compare every letter to every other letter before it can make a change, where sort_2 only needs to make a change if a letter is flagged, making sort_2 much more efficient. Going from a loop count of 36 in sort_1 to 6 in sort_2.

2. Sort “abcdefga” with each sort…with the added “a” at the end, what’s the difference?

   Other than the way these are computed, there is no difference.

2) Find and run the KaPing mainline at "**PROG1400\Eclipse-WS\Module-02-oo-KaPing\src\sampleKaPing**"

KaPing Main line snippet

```
	public static void main(String[] args) {

		Random rnd = new Random();

		for (int count = 0; count < 10; count++) {

			Ball ball = new Ball(rnd.nextInt(30));
			Paddle paddle = new Paddle(rnd.nextInt(30), rnd.nextInt(5) + 5);

			System.out.println("========================");

			ball.printit();
			paddle.printit();

			if (paddle.doesItCollide(ball)) {
				System.out.println("Ka-Ping");  // Collided
			}

		}
	}
```

BaseDisplayClass snippet of declared fields

```
public class BaseDisplayClass {

	private int startPos;
	private int length;
	private char ch = 'X';
…
```

(a) **[5%]** Looking at BaseDisplayClass, what does "`private`" mean?

private declares that this variable can ONLY be seen by that class specifically, nothing outside of that class can see that variable.

BaseDisplayClass snippet of doesItCollide() method

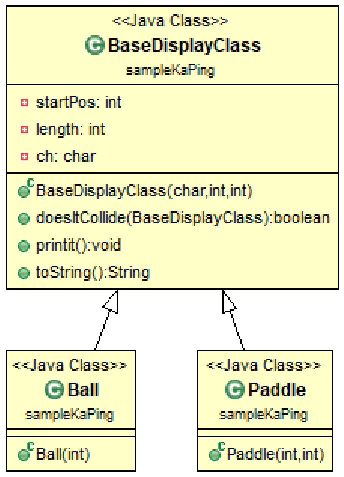
```
	/**
	 * Return true if display objects overlap.
	 *
	 * @param otherDisplayObject - display object to check against this.
	 * @return
	 */
	public boolean doesItCollide(BaseDisplayClass otherDisplayObject) {

		if (this == otherDisplayObject) {
			System.out.println("Same as me!!!");  // same object, don't compare
		}

		return ((this.startPos < (otherDisplayObject.startPos + otherDisplayObject.length + 20))
				&& (this.startPos + this.length > otherDisplayObject.startPos));

	}
```

(b) **[5%]** Looking at BaseDisplayClass doesItCollide() method, why can this method access the "private" field of both objects, given one object is a "Ball", and the other is a "Paddle"?

Both Ball and Paddle extend BaseDisplayClass, inheriting everything from their base class.

(c) **[10%]** Given (a) and (b) answers, what can you conclude about how you would go about Object Oriented Design?

This means that both Ball and Paddle have those variables as well, making them accessible.



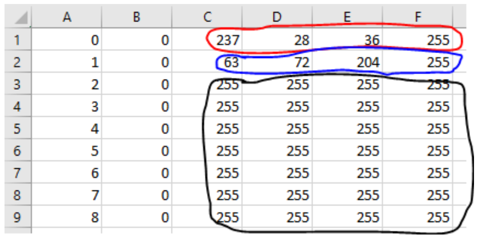
(d) **[10%]** Comment on the relative size of the 3 java files in the class diagram above, why is the BaseDisplayClass larger? Is this good?

The BaseDisplayClass is larger because it's the BASE CLASS that the other two inherit from, this is both good and bad, it's fine but if you were to add more functionality, depending on your needs this would determine where you would need to add the functionality. For example you may want the ball to bounce, and the paddle wouldn't need to bounce so you'd be better off adding a method to Ball instead of BaseDisplayClass where you would need to declare more for Paddle, even though it's completely unessasary for a paddle to bounce!

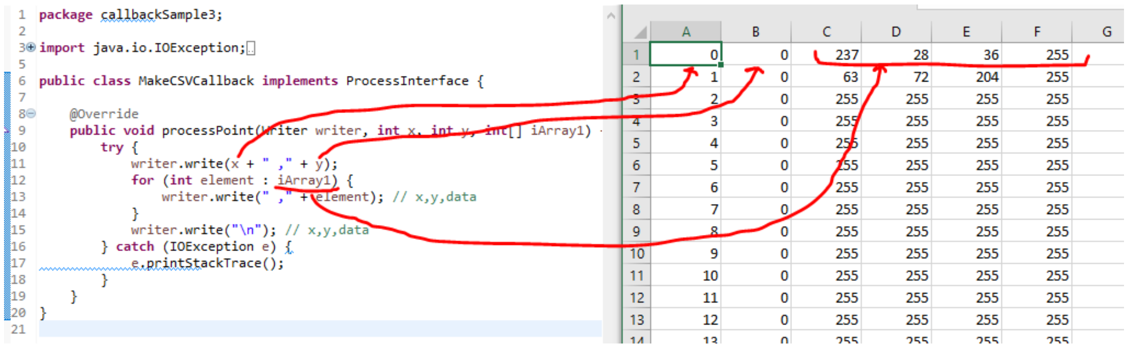
3) Find and run the mainline at "**PROG1400\Eclipse-WS\Module-15-Callbacks\src\readPNG**"…note that the PNG is read using two separate but similar methods.
Also run the mainline at…
"**PROG1400\Eclipse-WS\Module-15-Callbacks\src\callbackSample3**"

(a) **[30%]** The current callback implementations (MakeCSVCallback and MakeTXTCallback) write a point for every raster pixel. Make your own callback (MakeFINALCallback) that puts a point only if the pixel is not white. The data below shows that white points are 255s, while red and blue have different values. Your callback detects these values, and writes a CSV file (comma separated) with only dots that are red and blue. Note that such an output file can be used as input to your grouped items assignment (…and you can read the secret message).

SEE GIT REPO.



	A	B	C	D	E	F
1	0	0	237	28	36	255
2	1	0	63	72	204	255
3	2	0	255	255	255	255
4	3	0	255	255	255	255
5	4	0	255	255	255	255
6	5	0	255	255	255	255
7	6	0	255	255	255	255
8	7	0	255	255	255	255
9	8	0	255	255	255	255



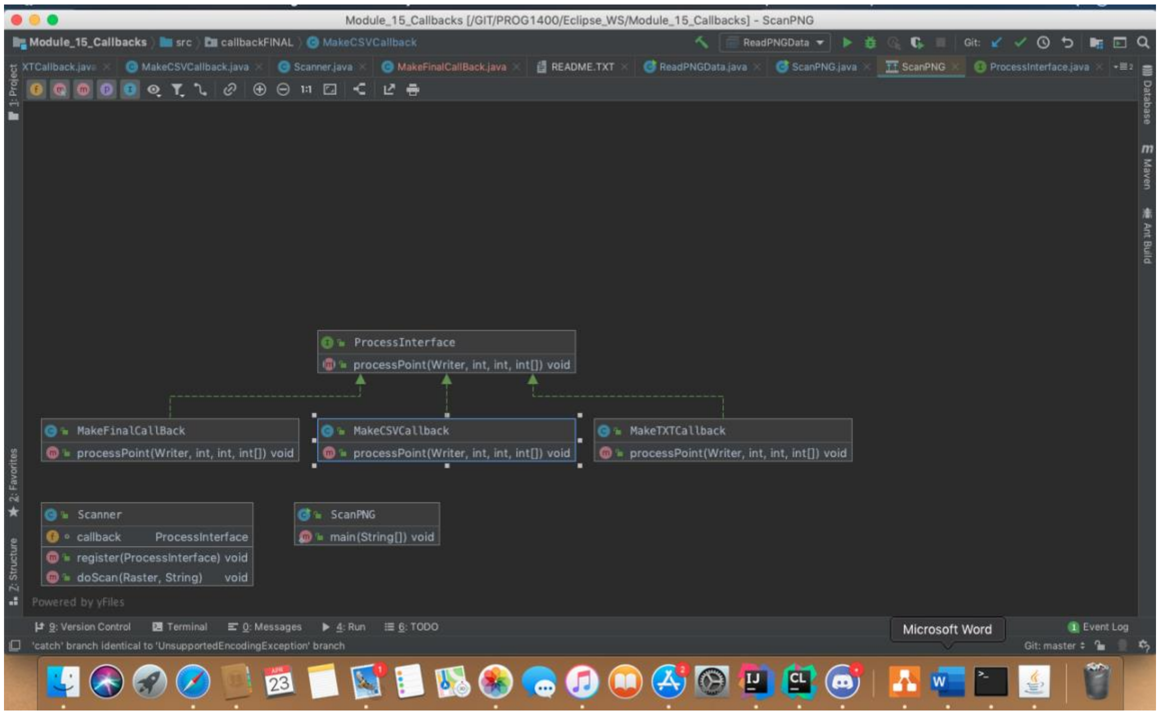
```java
package callbackSample3;

import java.io.IOException;

public class MakeCSVCallback implements ProcessInterface {

    @Override
    public void processPoint(Writer writer, int x, int y, int[] iArray1)
        try {
            writer.write(x + " ," + y);
            for (int element : iArray1) {
                writer.write(" ," + element); // x,y,data
            }
            writer.write("\n"); // x,y,data
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

	A	B	C	D	E	F	G
1	0	0	237	28	36	255	
2	1	0	63	72	204	255	
3	2	0	255	255	255	255	
4	3	0	255	255	255	255	
5	4	0	255	255	255	255	
6	5	0	255	255	255	255	
7	6	0	255	255	255	255	
8	7	0	255	255	255	255	
9	8	0	255	255	255	255	
10	9	0	255	255	255	255	
11	10	0	255	255	255	255	
12	11	0	255	255	255	255	
13	12	0	255	255	255	255	
14	13	0	255	255	255	255	

(b) **[10%]** Compare the differences between the original ReadPNGData java that used two similar methods with the use of a callback. Consider "Information Hiding".

ReadPNGData is all in one class file, no OOP involved. Where the methods all use some form of OOP for readability. Everything is also public and shouldn't be, things should be package private.

(c) **[10%]** Update the class diagram below to include your new callback.


Module_15_Callbacks [/GIT/PROG1400/Eclipse_WS/Module_15_Callbacks] - ScanPNG
ProcessInterface
processPoint(Writer, int, int, int[]) void
MakeFinalCallBack
processPoint(Writer, int, int, int[]) void
MakeCSVCallback
processPoint(Writer, int, int, int[]) void
MakeTXTCallback
processPoint(Writer, int, int, int[]) void
Scanner
callback ProcessInterface
register(ProcessInterface) void
doScan(Raster, String) void
ScanPNG
main(String[]) void
Powered by yFiles