Learning Goals

- LG1: Implement the “peer instruction” process by A) clicking in on an answer B) discussing with your team C) coming to consensus and re-clicking in
- LG2: Describe the difference between the Dr. Java interaction pane and definition pane (using IM and email as a metaphor).
- LG3: Describe how = is different in programming than it is in math.
- LG4: Be able to track a turtle’s path by reading and tracing code
- LG5: Identify common errors in Java code and use computing terminology to describe them
- LG6: Read and modify a method to perform a given task with a Turtle. Specifically understand the use of a parameter to a method.
- LG7: Know which methods get stored in Turtle.java and which get stored in their own files (and why).
- LG8: Describe return values of methods and identify ones from methods you have used
- LG9: Nest several method calls in one statement (in contrast to doing multiple assignments)
- LG10: Explain how bits (0s and 1s) in a computer’s memory can be interpreted in a variety of ways (depending on issues of type and representation)
- LG11: Use a for each loop in Java to loop over all pixels in a picture and perform a transformation on those pixels (e.g. decrease their red component).
- LG12: Apply Java syntax to cast a variable to another value (and explain how it’s useful in setting picture colors).
- LG13: Read and understand a while loop for modifying a Picture.
- LG14: Describe the single array representation of pixels in Picture.java
- LG15: Be able to draw the CSE8A memory model for a program that uses a loop to modify a picture.
- LG16: Find bugs in code including: a common “empty loop” bug, using the “how many times does this statement get executed” analysis technique.
- LG17: Argue about the best location of a print statement to debug a problem with a loop.
- LG18: Read and trace execution of a code with a for loop (using a single array of pixels).
- LG19: Apply knowledge of how people pass racketballs to swap values of variables
- LG20: Trace the execution of code through multiple method calls.
- LG21: Find and fix array index out of bounds exceptions in a for loop
- LG22: Compare and contrast a growth mindset and a fixed mindset as applied to programming.
- LG23: Read and trace execution of a nested loop that loops over a Picture object
- LG24: Read and code nested loops to perform mirroring (and related) transformations using nested loops.
- LG25: Employ the technique of drawing examples to explore various cases with 2-D array manipulation.
- LG26: Read, trace, and write code that performs a transformation over a restricted set of pixels (like a “box” of pixels).
- LG27: Read, trace and write a method that uses more than one Picture object (including where one Picture is the calling object).
- LG28: Modify a method to make it more “flexible” by replacing constants with parameters.
- LG28 (repeat): Read, trace, and write code that performs a transformation over a restricted set of pixels (like a “box” of pixels).
- LG29: Read, trace and write a method that uses more than one Picture object (including where one Picture is the calling object).
- LG30: Modify a method to make it more “flexible” by replacing constants with parameters.
- LG31: Compare and contrast code which modifies pictures conditionally based on pixel index location, based on Color of the pixel or both.
- LG32: Write and read code that either loops over a subset of pixels, or loops over all pixels and controls changes to pixels with an if statement
- LG33: Use parameters to control looping in conditional setting of pixels
- LG34: Identify the flow of control in the three types of Java if statements: if, if-else, and if-else if-else
- LG35: Compare and contrast two solutions to a problem using for loops and if statements
- LG36: Be able to identify which pixels you want to check for chromakey
- LG37: Identify, compare and contrast if-else if –else statements and separate if statement blocks.
Sample Final Exam CSE8A Winter 2009: This exam not indicative of actual exam length

- LG38: Explain the relationship between amplitude, frequency, compression and rarefaction as it describes sound.
- LG39: Defend a choice of sampling rate and sample size for a digital sound file based on human hearing abilities and translate that into an estimate of the size of a sampling array for storing sound.
- LG40: Compare an array of Pixels which forms a Picture to an Array of SoundSamples which forms a Sound including the result of indexing into each.
- LG41: Read, trace, and write code to change the volume of a Sound object (using for each, while and for loops)
- LG42: Compare and contrast the difference between changing the Color of Pixels in a Picture to changing the volume of SoundSamples in a Sound.
- LG43: Trace execution of sound code using multiple loops and if statements.
- LG44: Be able to use either array-based or method-based (API) access for (getting/reading, setting/writing) sound File.
- LG45: Be able to explain in plain English what code modifying Sounds does.
- LG46: Trace indexing patterns in code to perform Sound manipulations (e.g., reversing a sound).
- LG47: Identify scenarios where object transformations lead to objects of different “sizes”.
- LG48: Create a Sound of a specific “length” or duration using new
- LG48: (again) Read, trace, and write methods which creates and returns new objects (rather than modifying calling objects).
- LG49: Identify issues with overwriting arrays when trying to “modify in place”.
- LG50: Describe how a Java class is made up of instance variables or fields, constructors, and methods and brainstorm a given class design.
- LG51: Identify common errors made by novices in defining their own classes.
- LG52: Compare and contrast String class methods and Sound class methods for testing char elements of String (hint: helpful for homework!)
- LG53: Identify common structure of “getter” and “setter” methods.
- LG54: Be able to draw the memory model of an object (at a more detailed level than before) – based on what happens in a constructor.
- LG55: Identify legal and illegal instances of method overloading, so you can know what “variations” on methods you can write.
- LG56: Create arrays of objects of any length or type and be able to use good software design for using arrays as fields
- LG57: Identify scenarios where arrays may be “linked” by index
- LG28: Read, write, and find “difficulties” with setter and constructor methods using arrays.
The following questions concern the following piece of code.

```java
1: public Sound sillyModification()
2: {
3:     Sound newMusic = new Sound(this.getLength());
4:     int yyy = 0;
5:     int qqq = this.getSampleValueAt(0);
6:     int rrr = qqq;
7:     int curr;
8:     for (int i = 0; i < this.getLength(); i++)
9:     {
10:        curr = this.getSampleValueAt(i);
11:        yyy += curr;
12:        if (curr > qqq)
13:           qqq = curr;
14:        else if (curr < rrr)
15:           rrr = curr;
16:     }
17:     int zzz = yyy / this.getLength();
18:     for (int i = 0; i < this.getLength(); i++)
19:     {
20:        curr = this.getSampleValueAt(i);
21:        if (curr > zzz)
22:           newMusic.setSampleValueAt(i, qqq);
23:        else if (curr < zzz)
24:           newMusic.setSampleValueAt(i, rrr);
25:     }
26:     return newMusic;
27: }
```

1. What would be a better variable name for qqq (pick the BEST answer)?
   A) sum
   B) average
   C) value
   D) min
   E) max

2. What would be a better variable name for zzz (pick the BEST answer)?
   A) sum
   B) average
   C) value
   D) min
   E) max

3. How long is the SoundSample array in the object newMusic?
   A) 0
   B) zzz
   C) zzz - 1
   D) this.getLength()
   E) this.getLength()-1
4. Describe in plain English what this code does.

5. Assume that the calling object SoundSample array looks like this. What does the newMusic SoundSample array contain after this method executes?

6. If the sillyModification method was called by the following piece of code in a main method, what would be true?

```java
Sound foo = new Sound(FileChooser.pickAFile());
Sound x;
x = foo.sillyModification();
```

A) the foo sound is unchanged, the x sound is empty (silent)
B) the foo sound is unchanged, the x sound is a modified version of foo
C) the foo sounds is changed and empty(silent), the x sound is what foo originally was
D) the foo sound is changed (a modified version of its original values), the x sound is what foo originally was
E) None of the above.

7. Suppose I want to write a different version of sillyModification to do basically the same thing. However, I want the method to be called from a main program like this:

```java
Sound foo = new Sound(FileChooser.pickAFile());
Sound x = new Sound(foo.getLength());
foo.sillyModification(x); 
```

What line numbers would you change in the program and how would they change?
8. Suppose I have the scenario where I have an array of student grades in the form of an array of `double` values. My goal is to fill in a different array of character variables (`char`s) filling in an appropriate letter grade (A,B,C,D, or F).

Which would be the best form of conditional control statement to use in Java (of the ones listed below):

A) A series of if statements
B) A series of if else statements
C) A series of if else if statements with no else
D) A series of if else if statements with an else
E) A single, complex if statement

9. Complete the following code segment to create an appropriate `letterGrade` array. The code you write should work no matter how big the grade array is made.

```java
public static void main(String[] args)
{
    double[] grade = {89.5, 93, 76, 98, 45};
    //Add your code here
}
```
10. Write a method of the Picture class that modifies the calling object so that every “pair” of columns of the picture is swapped with the column next to it. That is – the original picture’s column 0 should be swapped with column 1, column 2 should be swapped with column 3, etc. The code you write should work for both Pictures of odd and even widths.

(Variations to consider – do you know how to…)
- Do the same thing but with rows?
- Swap columns, but only those pixels that have a strong green component (say green is more than 200?)
- Instead of modifying the calling object, create a new object (of just the right size) and make it be the “swapped column” result?
11. What does this code do? Explain in plain English. The following is a method in the Picture class:

```java
public Picture sillyStuff(Picture p, Picture q)
{
    int w = Math.min(Math.min(p.getWidth(), q.getWidth()), this.getWidth());
    int h = Math.min(Math.min(p.getHeight(), q.getHeight()), this.getHeight);
    Picture x = new Picture(w, h);
    for (int a = 0; a < x.getWidth(); a++)
    {
        for (int b = 0; b < x.getHeight(); b++)
        {
            x.getPixel(a, b).setRed(p.getPixel(a, b).getRed);
            x.getPixel(a, b).setGreen(q.getPixel(a, b).getGreen);
            x.getPixel(a, b).setBlue(this.getPixel(a, b).getBlue);
        }
    }
    return x;
}
```

(Note: I probably wouldn’t ask both the above question and the next series of questions on the same exam, but they are good practice).

12. In the code above, what does line 3 do?

13. In the code above, what does line 5 do?

14. Does the code above make changes to the Picture based on

   A) The pixel locations
   B) The pixel color values
   C) Both pixel location and pixel color value
   D) Ordering of parameter values

15. Suppose I want to change the above code to make a picture that is as LARGE is the largest picture. What problems will I have with my loop as written? Explain your concerns in English.
16. Suppose I have some wacky Picture method that creates a new Picture from the calling object Picture in a kind of wacky way. Complete the code with an appropriate if statement to make sure that I have no index out of bounds errors.

```java
1:   public Picture wackyStuff()
2:   {
3:     Picture p = new Picture(this.getWidth(), this.getHeight());
4:     for (int x = 0; x <this.getWidth(); x++)
5:     {
6:       for (int y = 0; y < this.getHeight(); y++)
7:         {
8:           Pixel newPix =  p.getPixel(x,y);
9:           Pixel oldPixLeft = this.getPixel(x-13,y);
10:          Pixel oldPixDown = this.getPixel(x,y+4);
11:         int red = (int)(oldPixLeft.getRed() + oldPixDown.getRed() ) / 2;
12:          Color c = new Color(red,0,0);
13:          newPix.setColor(c);
14:       }
15:     }
16:   return p;
17: }
```

What line of code would you want to include? _______________________________________________________

In between what lines would you put it? _______________________________________________________

What lines of code would be “conditionally executed” when this if statement evaluates to true?___________________________

Maybe change the first question to be what if condition would you want to include for clarity?

17. An alternate to using an if statement to control execution so that no array index out of bounds exceptions occur is to change the loop control structures. Assume you cannot (for some bizarre reason) use an if statement in your code. How would you change the loops on lines 4 and 6 to ensure code that executes with no errors?
Suppose I have designed an Automobile class. Every automobile has the following instance variables:

```java
int numDoors;
String VIN;
Color bodyColor;
```

18. Write a constructor method that takes three parameters – one of type `int`, one of type `String` and one of type `Color`.

19. Write a setter method for the `numDoors` instance variable.

20. Draw the memory that is created when the following code is called:

```java
Automobile bethsCar = new Automobile(2, "ZNC2345SDSFF", new Color(255,0,0));
```

21. Bonus: what color is Beth’s car? 😊
22. Write a method in Sound.java called repeat three times. This method will return a new Sound object that is the repetition of the calling object three times.

23. Write a method in Picture.java that does basic vertical edge detection on a grayscaled image. Again, you should not modify the calling object but return a new image with vertical edges detected. You can recall that the basic edge detection method from your PSA did something like:

```
newCurrentPixelValue = currentPixelValue - nextPixelValue
```

Where the currentPixel and nextPixel are next to one another horizontally. Perform the same operation, but now do it vertically.

This code in Picture.java will apply to the next question

```java
public void bumpRight()
{
    Pixel[] pA = this.getPixels();
    for (int i = 1; i < pA.length; i++)
    {
        pA[i] = pA[i-1];
    }
}
```

24. If the pixel array contains: [2,3,4,5] what is the pixel array after calling bumpRight()?
We are creating a new Zoo class. The zoo can only hold elephants and lions. It has has the following instance variables:
   int numElephants;
   int numLions;
   int numAnimals;

25. If we have a Zoo(int elephants, int lions, int animals) constructor, name one check you would perform to ensure the legitimacy of the parameters (like you did with your CreditCard number):

26. Your boss tells you that you need to be more efficient and get rid of the numAnimals variable. He then tells you that you still need a get method “getNumAnimals”. Write this method:
Also review the sample midterm and the actual midterm. This exam is cumulative. Questions people struggled with on the midterm have been known to show up on the final exam (perhaps in a modified form).