—Assignment #6—

Make sure to review the submission requirements on pg. 4 of the Lab #1 document.

Follow the Style Guide as given in the Resources section of our conneX course page.

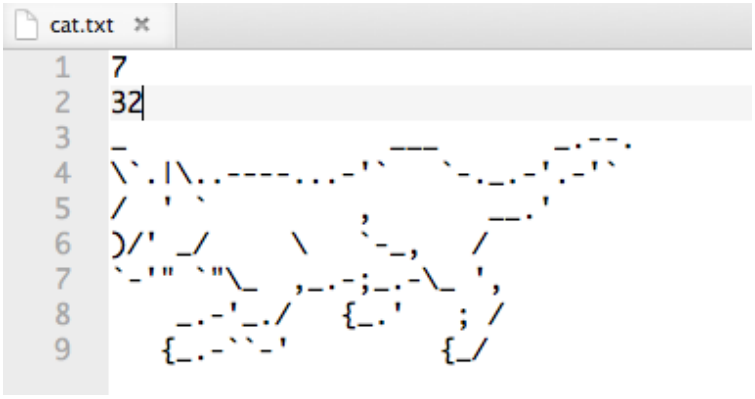
Learning Outcomes

When you have completed this assignment, you should understand:

- How to break down a problem into parts using Classes.
- How to write your own Class.
- How to use your own Classes in a client program.
- How to use 2D arrays and text files to store data.
- How to print a text art image using 2D arrays and Objects.
- Some basic procedural distribution in graphics.

You will write a program that draws a ascii text art picture to the console. Download the template files `Picture.java` and `Sprite.java` from Assignment #6 on conneX, along with the other files `Screen.java` and `Point.java`. Write the missing code for the methods in the `Sprite.java` template, then use it to complete the client program `Picture.java`, and as usual, compile and test your code often as you write it up.

The template in `Sprite.java` is set up so that you will write code to read in the lines of a text file that contains some small 2D ascii art. The format of the file should read in two numbers first, the number of lines followed by the maximum number of characters in a line. For example:



There should be no extra spaces at the end of any line, and note that you will be writing code that reads files one line at a time, instead of one token at a time. The reason for this is so that you can clearly see the ascii text art in the file you will be using, and allows you to read in space characters. Remember, that Java has a small issue with `Scanner` calling `nextLine` after `nextInt`, and you have to make one extra call to `nextLine` because of it, as we've seen before.

The `Sprite.java` class has a 2D char array called `art` to hold the characters read in from your

text file. Make sure when you fill it with the file information that you finish each row of the 2D array by filling it with any necessary space characters. There is also a `Point` field called `p` that should tell a `Sprite` instance where to draw the bottom-left corner of the 2D array to an instance of a `Screen` (called `panel`).

The `draw` method should only set a char value to the `panel` if it is not a space character. The `get` method is only to make accessing the 2D `art` array avoid the confusion of using indices out of order with typical coordinate systems (`x` for horizontal, then `y` for vertical). Use the `get` method in the `draw` method to make your code easier to read.

The `setCorner` and `put` methods can be written in one statement, and are very simple. Do not overthink them. They give their corresponding field variables an instance of an object from the parameter. You need to use variable shadowing in `put`.

Use the `Sprite` class to create three instances in your `Picture.java` client code, with three corresponding text files that each contain ascii art like the cat shown above. Please stick to art that is within 15 rows by 30 columns. Feel free to use something you find on the Internet or make your own. When you initialize a `Sprite` using the constructor, make sure to pass in the filename so it knows where to find the data it needs to create itself.

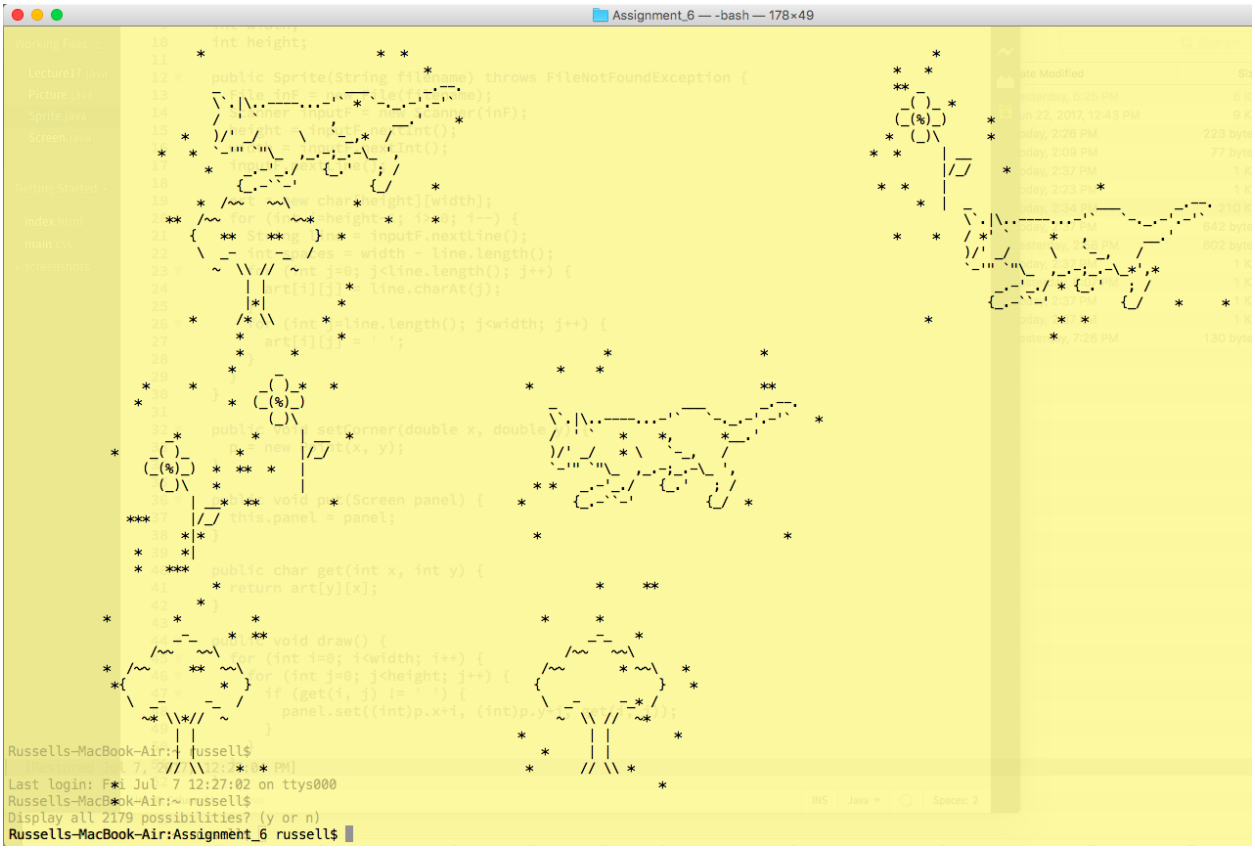
You need to make sure to call `put` for each `Sprite` instance so that it knows which `Screen` to draw itself. You also need to call `setCorner` so it knows where on the `Screen` to draw itself. You can reset the corner and call `draw` as many times as you want to have the `Sprite` appear in different places on the `Screen`.

Repeat the three different instances of `Sprite` each in three random places drawn on the `Screen` using an instance of Java's `Random` class (so you will have nine images in total). Restrict the placement so that none of the `Sprite` instances draw over the edge of the boundaries of your `Screen` instance. It is okay if the copies draw on top of each other when randomly placed.

Part of procedural graphics (generating images from math) is a concept called "Barnacling." It is a design choice to randomly distribute smaller objects nearby larger ones, because nature tends to do this naturally on its own instead of being completely random.

Use Barnacling to draw, for each copy of your `Sprite` drawings, 20 asterisk (*) characters placed randomly nearby the copy (approx. at most 5 chars away from its border). Use whatever formulae you feel will do the best to achieve this. It is okay for some of the asterisks to print on top of each other.

Your finished code should output something similar to the image on the next page.



Please submit both your finished `Picture.java`, `Sprite.java`, and your three sprite text ascii art files to conneX.

Grading

Marks will be allocated out of 30 points:

- (0 points) You do not submit any program file.
- (5 points) Your program does not compile. The marker cannot fix the complier error easily, or if they can, the output is severely mismatched with the assignment requirements.
- (10 points) Your program does not compile, but the marker can fix the error easily.
- (15 points) Your program compiles, but only outputs a few of the assignment requirements.
- (20 points) Your program compiles, outputs most of the assignment requirements, and is missing documentation comments or has formatting issues.
- (25 points) Your program compiles, outputs everything required, but is missing documentation comments or has formatting issues.
- (30 points) Your program compiles, outputs everything required, and has proper documentation and formatting.

This assignment gives you the basic ideas behind designing graphics for a game. Make any modifications on your own, and have fun! You can even animate things following a similar loop structure seen in our other work. I specifically made the `Point` class use the `double` data type to make sure you can adjust them with any velocity in an animation you want, although the result with text is kind of crude, it still gives you a lot you can do.