# Objective

To use basic processing skills to create an underwater sea creature and then use inheritance to add that creature to the Fish Tank.

## Topics: Basic Processing Draw Commands, Control Structures, Inheritance

# Instructions

The fish tank is looking a little empty right now. It’s our job to come up with some creative underwater sea creatures to add to the tank.

Our first step is to simply draw our sea creature using Processing. First, create a Processing sketch that is approximately 250x125. It will be important later to know where our creature is on the screen, so inside this sketch, create two variables x and y that will be the center position for our creature. Spend some time drawing your creature making sure all of your shapes relate to the (x,y) position you created. Save this sketch as <YourLastname>Fish.

Second, we will put our fish into the fish tank using a Computer Science topic called inheritance. Inheritance allows programmers to make code more flexible by creating a template that other people can use. This is a multi-step process, but it will be worth it to fill the tank. We start by creating a new tab in our fish tank to put our code. Open the lab14\_Fish\_Tank. Notice the little arrow that points to the right under the word “standard.” Press it and select the first option “New Tab”. Processing will ask you to name your new file. Call it <YourLastname>Obj.

Next we will make the blueprint for our underwater sea creature. In Processing, this is called a class. We start our code by writing the word ‘class’ then putting the name of our file. Don’t forget to put the open and closing squiggly braces after your first line. It should look like this:

class LastnameObj  
{  
  
}

In our fish tank, the template we use to make sure everyone “fits” in the tank is called AnimatedObject. Everyone who wants to belong to our fish tank must be an AnimatedObject deep down inside, so we must inherit all the information from the AnimatedObject class. Processing makes this easy. On your first line of code, add the words “extends AnimatedObject” that’s all it took to let the computer know our class is really an AnimatedObject.

Now it’s time to take our sea creature code and add it to our class. Let’s start by taking our two variables x and y and put them just after the opening squggly brace. Next we will override a method so our creature will show up on the screen. You will notice that we inherited an abstract method called **display()** from AnimatedObject. Now it’s time for us to add a body to that method. Copy that line of code from AnimatedObject to our class. Remove the word ‘abstract’ and the semicolon at the end then add a set of squiggly braces. Next copy the code from your original 250x125 sketch into the body of the display method. Your code will look something like this:

class LastnameObj extends AnimatedObject  
{  
 float x;  
 float y;  
  
 void **display()**  
 {  
 //your cool sea creature code here  
 }  
}

You’ll notice that we inherited two more abstract methods from AnimatedObject called **getX()** and **getY()**. Repeat the same process of copying, deleting, and adding that we just did for the **display()** method. These methods simply return our X or Y value so someone else can use them. In order to return a value in a method, use the word ‘return’ and the name of the variable we need to send back.

Finally we need to add something called a constructor. The word sounds like construction doesn’t it? And just like a construction company builds things in the real world, a constructor builds things in the Processing world. Constructors have the same name as the class followed by a set of (). Let’s put our constructor right below our two variable declarations. Add a set of squiggly braces so our constructor has a body. Last, let’s use our constructor to fill our X and Y variables. Use the **random()** function we discussed earlier to put values into X and Y. Our code should look like this when we are finished:

class LastnameObj extends AnimatedObject  
{  
 float x;  
 float y;  
  
 **LastnameObj()**  
 {  
 x = random(width);  
 y = random(height);  
 }  
  
 void **display()** {  
 //your cool sea creature code here  
 }  
  
 float **getX()** {   
 return x;  
 }  
  
 float **getY()** {  
 return y;  
 }  
}

But if we press the play button, we still don’t see anything?! Why?! It is because we’ve only built a blueprint (aka class). Just like a blueprint for a house tells us how to build a house, the blueprint *isn’t the house*. Our class only tells the computer *how* to build our object. Now it is up to us to actually “build” our object in code.

Find the **setup()** method in the Google\_Fish\_Tank\_2012 tab. You will notice a comment line that says //add YOUR object here. This is where we will do our last two lines of code. First we must make our object. We will start by making a container that is the same type as our class. Use your initials to name the container. Next, fill that container with a new object by using the keyword new and then calling the constructor. Confused? Let me give you an example:

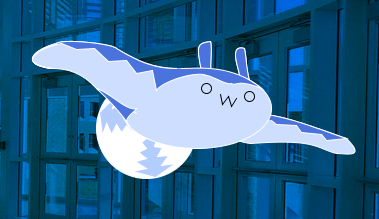
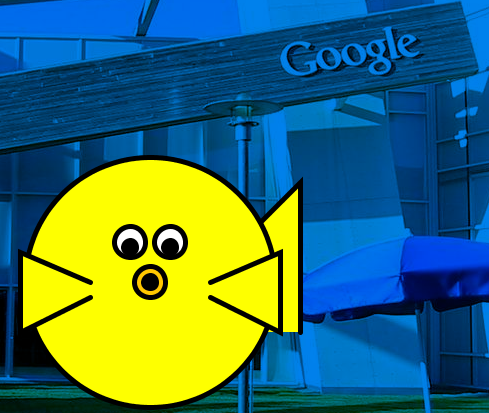
CadleObj abc = new CadleObj();

Finally, tell the list of AnimatedObjects to add your object to the list of creatures in the tank. The name of the list is called objs and it has a convenient method called **add()**. Your code should look similar to this:

objs.**add(**abc**)**;

Press play to make your object appear!

# Examples



# Hints

none

# Challenge

**Move #1** – Now that you have your sea creature in the tank, make it move! You’ll notice that our object inherited a method called **move()**, however, there is nothing in the **move()** method! In your class, override the **move()** method by copying it from the AnimatedObject class and pasting it in your class. How do we get our object to more? By changing its own (x,y) location. Start by simply adding one to the x value to make your creature move to the right. Try changing the y value as well. Use your conditional statements to reset x and y values if your creature moves out of the screen.

**Animation** – Now that your object is moving, let’s actually make it look like it really moves instead of just hovering around the tank. We need to make another picture of our object that is slightly different than the first, maybe a fin changed position or a tentacle moved. Write a method called **costume2()** that has the code you just wrote for the different picture. Next write a another method called **costume1().** Cut and paste the body of the **display()** method into **costume1()**. Now we need to re-work our **display()** method so that it switches between costumes. Create a variable at the top called fCount (short for frame count). In your constructor, set fCount to 0. Next, have your **display()** method add one to count at the very beginning. This will count up by one every time the display method is called. Next in **display()**, we need to know whether to call **costume1()** or **costume2().** Use a cascaded if/else statement to check fCount. If it is less than 30 we should call **costume1()** if it is greater than 30 but less than 60 call **costume2()** and if it gets to be bigger than 60 reset fCount to 0. Now you should have a creature that actually looks like it is moving.

**Move #2** – You might be wondering why there are to **move()** methods. This is called method overloading. It is when two methods have the same name, but different number of parameters. Your goal now is to make a smart sea creature. We will use the second **move()** method to do this. Copy the second **move()** method from the AnimatedObject class to your class. This **move()** method had one parameter called locs. locs is an array that contains the (x,y) coordinate of every other object in the tank. Now that you know where everyone is, try coming up with a creative way to use this information. Maybe you could “follow” another object in the tank or if there are too many other objects you can “hide.”

# Special Thanks To:

Bryn Mawr College and SMU this lab was developed under their NSF TUES I grant The grant has

a website at [www.cs.brynmawr.edu/visual](http://www.cs.brynmawr.edu/visual). Updates made by Aaron Cadle