UNIT 21. INTRODUCTION TO OBJECTS

Objects are not an official part of this course, and this section is not testable. However! Objects are awesome and so I thought it would be great to put them in the notes.

Objects are the next natural progression in this material. So far, we learn a new technique, find the limitations, and learn another new technique to overcome those limitations. Our latest limitation is that it is very hard to pass around a lot of data to and from functions. Also, you end up getting larger and larger messes of variables for components of your program, which becomes unmanageable as your program grows. For example, let's make a bad guy that flies around a screen. The bad guy has an `x` value, a `y` value, a width, a height, a color, and a move speed.

```
int badGuyX;
int badGuyY;
int badGuyW;
int badGuyH;
int badGuyClr;
int badGuyMaxSpeed;
```

We have learned how to scale this up to 1000 bad guys, using arrays. That is easy. There are remaining problems, however. Imagine a function to draw this bad guy. It may look like this:

```
void drawBadGuy(int x, int y, int w, int h, int clr,
    int maxSpeed);
```

this is tedious, but doable. Every time we call this we have to type in all of those bad guy properties. It gets worse if we have more properties (rotate speed? Bullets left? Health? Shape? 10 others?).

Another problem is that we can only return one value from functions, and we cannot change those values in a function. So the following function simply is not possible.

```
void moveBadGuy(int x, int y, int maxSpeed);
```

We can use max speed to move the bad guy, but we cannot get the modified x and y out of the function. It is lost, because only the local copies in the function are modified.

The solution to this is objects. Object oriented programming is a huge topic, and there are many advanced avenues and dusty corners – you will learn about it in great detail in a Computer Science degree. However, basic objects are not so bad, and

you have learned a lot of the skills already needed to make them.

Basically, objects are a nice way to collect data in a clean wrapper. You can "chunk" data together. For example, a `BadGuy` object would have a bunch of properties, such as an `x`, `y`, `w`, `h`, `clr`, `maxSpeed`, and many others. We setup an object to have all this in one package, and then we can use it like other types. We can pass an object to a function, and pass an object back, and everything stays grouped together.

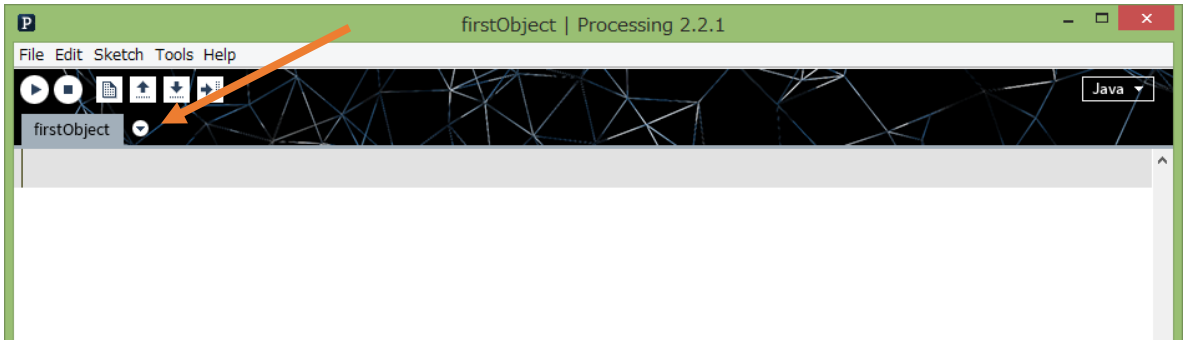
21.1 **Creating Your First Class**

We have many data types in processing. We have the primitive types (`int`, `float`, etc.), and the array types. We also have `String`, which we mentioned before, is an object.

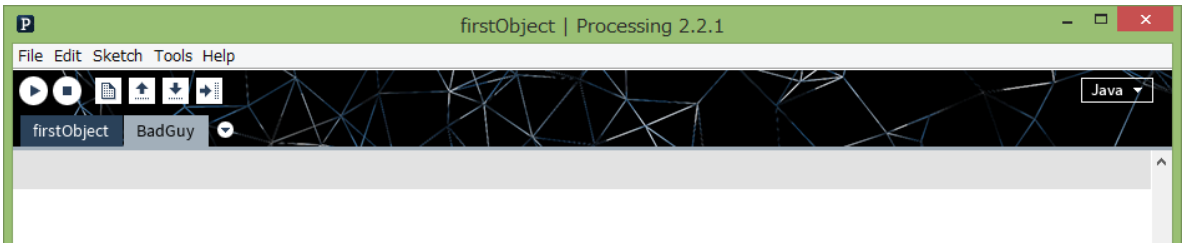
To create our own objects, that work a little like the String type, we need to define our own, new, data type. This is called a `class`. Then, once we create this new type, we can create new variables that use this type. Finally, there is a new step for actually making the object work.

Perhaps the best way to explain this is to show it.

First, in processing, you need to add a new tab, which adds a new file, to your project. Click on the down arrow next to your project name:



And choose "new tab". It will ask for the file name. Type in `BadGuy`. A new tab appears that has no code in it.



Select this tab, and we can type more processing code in here.

To create our own object type, we need to learn new special syntax to make our first

class. The syntax of a class is very simple:

```
class ClassName
{
} // end class
```

We can specify a class name, and now our program has a whole new type that we can use. Convention has classes start with a capital letter (like `String`!) and it is good practice to call the class the same as the tab / file name (this is required in pure Java). We will call our class `BadGuy`:

```
class BadGuy
{
}
```

So far, this is not very exciting. We have created our own class, but we did not put anything into it. It turns out that we can put stuff inside the class, and it all gets grouped together under the `BadGuy` name. For one, we can add variables (which is where this example started). We can also add other things, like functions (called methods when they are part of an Object), but that is for a later course.

Let's add a collection of variables that define the bad guy.

```
class BadGuy
{
  int x;
  int y;
  int w;
  int h;
  int clr;
  int moveSpeed;
}
```

We can make any variables in here we want, just like in our main program. Arrays, too!

Now, **be careful:** You cannot store data in these (yet). At this point, you are just defining your brand new data type. A class just specifies what will be in an object if you create one, what kinds of data can be stored. This is just a blue print, a template. We need to learn how to make objects first!

21.2 **Instantiating Your First Objects**

Classes are a new data type that we created. We cannot actually store data in a type, we need to make variables with that type.

Go back to your main tab. Now that we created a class called `BadGuy`, we can use this like any other data type in our program – just like `String`, for example. Let's make three global variables for the bad guys:

```
BadGuy b1;
BadGuy b2;
BadGuy b3;
```

So making variables using your new class is easy. However, we need to learn some new things before you can use them. Just like arrays, you need to *instantiate* your object before you can use it. Also, these variables, just like arrays, only store the address of where the object is in memory – this has all the implications that arrays have, e.g., for use with functions, and the `==` operation.

- You need to *instantiate* an object before you can use it, similar to how you must instantiate an array.
- The object variable only stores a memory address of where the object is located in memory.

To instantiate an object, that is, to create a new object, we need to use the `new` keyword. There is a small difference from arrays: you put () after, like a function call:

```
ClassType variable = new ClassType();
```

For example:

```
BadGuy b1 = new BadGuy();
```

The `new` command goes off to memory (just like with arrays), allocates enough memory to store those variables you defined in `BadGuy`, and comes back with the memory address. The memory address is then stored in the variable for use later. Let's do it for all three:

```
BadGuy b1 = new BadGuy();
BadGuy b2 = new BadGuy();
BadGuy b3 = new BadGuy();
```

Just like with arrays, when we call `new` three times, we get three different spots in

memory. We have three bad guys, all with their own copy of the variables inside of them. They are stored at different locations, and changing one does not change the other, they are separate!

Here is a rundown of the terminology we use:

class – the blueprint of the data to be stored and functionality to be included

object – an actual instance of a class in memory that can be used to store data, as described in the blueprint

instance – see object

Note: sometimes, there are nuanced differences between the term instance and object, depending on the language. For most people they are the same.

21.3 **Accessing Instance Variables inside Objects**

Now that we have created the three objects, we need to learn how to access the object's variables (called *instance variables* since there is one set per instance of the object). You do this by putting a dot after the variable name, and then following with the variable name.

```
object.instanceVariable
```

for example

```
b1.x
```

What happens here, is that Processing looks inside `b1`,

this now acts just like any other variable. You can add to it, subtract from it, use it in a formula, etc.

For example, we can setup one of our bad guys in the setup block:

```
void setup()
{
  size(500,500);
  b1.x = 10;
  b1.y = 10;
  b1.w = 100;
  b1.h = 50;
  b1.clr = 255;
  b1.moveSpeed = 10;
```



```
}
```

Remember: here we don't use the `new` keyword because the object has already been created (*instantiated*) globally. Just like with arrays, objects always need to be instantiated.

We can use this to move, and then draw, the bad guy, in the draw block. The same as above, we use these like any other variable:

```
void draw()
{
  background(0);

  // move bad guy
  int move = (int)random(b1.moveSpeed*2)-b1.moveSpeed;
  b1.x += move;
  move = (int)random(b1.moveSpeed*2)-b1.moveSpeed;
  b1.y += move;

  // draw bad guy
  rect(b1.x, b1.y, b1.w, b1.h);
}
```

So this should work, but so far we are not really saving much work. I could do the above example without objects, and just make my own variables; I would even save typing while at it!

21.4 **Objects and functions**

The power of Objects starts to become obvious when we use them with functions. Classes are just another type, so we can toss objects back and forth from functions no problem. Further, because objects work like arrays – we toss the memory address around and not all the data – functions can actually modify the object.

Let's make a new function, that generates a new `BadGuy` and sets it to some random values. It doesn't take any parameters, but will return the new bad guy

- Create a new object instance of `BadGuy`
- Set the instance variables to reasonable but random values
- Return a reference to the object (the memory address). Make sure to set the return value of your function to the Class type as below

Try it out. This is what I came up with

```
BadGuy newRandomBadGuy()
{
  BadGuy b = new BadGuy();
  b.x = (int)(random(width));
  b.y = (int)(random(height));
  b.w = (int)(random(MAX_WIDTH)); // 100
  b.h = (int)(random(MAX_HEIGHT)); // 100
  b.clr = (int)(random(255));
  b.moveSpeed = (int)(random(MAX_SPEED)); // 10
  return b;
}
```

Now we can use this to create as many bad guys as we want. Since we instantiate the object in this function, we don't need to do it at the top of the program any longer. Also, update your setup to call this function and set your `b1`, `b2`, `b3` to their own random bad guys.

```
void setup()
{
  size(500,500);
  b1 = newRandomBadGuy();
  b2 = newRandomBadGuy();
  b3 = newRandomBadGuy();
}
```

We now have three random bad guys! Let's make two more functions. First, let's draw a bad guy:

```
void drawBadGuy(BadGuy b)
{
  stroke(b.clr);
  fill(b.clr);
  rect(b.x, b.y, b.w, b.h);
}
```

This takes in a `BadGuy` variable, and uses the instance variables to draw as needed. Next, let's make a function to move a `BadGuy` around. This is new! This was not possible before, without objects. It should take a `BadGuy` variable, modify it in place,

and not return anything.

```
void moveBadGuy(BadGuy b)
{
  int move = (int)random(b.moveSpeed*2)-b.moveSpeed;
  b.x += move;
  move = (int)random(b.moveSpeed*2)-b.moveSpeed;
  b.y += move;
}
```

This function takes a reference to a bad guy (a memory address), and uses that to get the instance variable values, AND, change them. Changes made here are permanent inside the object that was passed to us.

All we need now is to update our draw block:

```
void draw()
{
  background(0);
  moveBadGuy(b1);
  moveBadGuy(b2);
  moveBadGuy(b3);
  drawBadGuy(b1);
  drawBadGuy(b2);
  drawBadGuy(b3);
}
```

This is very powerful. Combine this with arrays, and – wow! We can do a lot. Keep at CS and you will learn even more powerful tools.

