Overloading Methods in Java

1. Constructor Overloading (companion code found here).

Let's say that you are making a Button class, and you want to provide an API to your user that is as flexible as possible.

One way you could achieve this is to provide lots of different options for how they can instantiate a Button object. Looking at the code that accompanies this example, you will see that the Button class contains the following member variables and constructor:

```
class Button
{

  PVector position;
  float buttonWidth;
  float buttonHeight;

  String label;

  Button(PVector newPosition, float newWidth, float newHeight, String newLabel)
  {
    position = newPosition;
    buttonWidth = newWidth;
    buttonHeight = newHeight;
    label = newLabel;
  }
```

This is fine, but we can provide a lot more flexibility than this for very little effort. What happens if the user doesn't know how PVectors work, and would be more comfortable with floats for the x and y coordinates, or what about if they want to make a square button and don't want to have to type in the same value for both the width and the height?

We can easily achieve this by adding in more versions of our constructor that take different argument lists, this is called *method overloading*.

Consider the following method signatures:

```
Button(PVector newPosition, float newWidth, float newHeight, String newLabel)
Button(PVector newPosition, float newWidthAndHeight, String newLabel)
Button(float newX, float newY, float newWidth, float newHeight, String newLabel)
Button(float newX, float newY, float newWidthAndHeight, String newLabel)
```

We are offering the user four different versions of the constructor that they can call, depending on what data types they would prefer to use and what shape they want their button to be (rectangle or square).

If you take a look at the `setup()` function then you will see each of these in action:

```
fileButton = new Button(new PVector(5, 5),   100, 40, "File");
saveButton = new Button(new PVector(105, 5), 100, 40, "Save");
loadButton = new Button(205, 5, 100, 40, "Load");
closeButton = new Button(new PVector(width - 45, 5), 40, “X");
```

If you inspect the definition for each of these overloaded constructor methods, you'll see that there really ins't any functional difference between them. So why bother doing it? Well, you'll find this sort of overloading *everywhere* when you start to use libraries and APIs. It's nice to be nice, so library developers want to provide as flexible an interface as possible.

For some more detailed examples, check out the definitions for the different versions of fill() on the Processing GitHub page.