Encapsulation

What is it?

Encapsulations provide a public interface for interacting with your class. It will hide all the implementation details. It can also be refered to as *information hiding*. Kind of like driving a car, you know the gist of what it does, but you'd gotta be a nerd (No I'm kidding!) to know what's going on involving the mechanisms of the car.

String is a encapsulated class, which means that we know what a string does and how to use the methods it provides us, but it wouldn't concern us how exactly Strings is doing what it does.

Leave the stage effects to the backstage, focus on your show.

Using Encapsulation

For encapsulation, we should make **private instance variables**, because users of the class will use all the interactions using public methods. Any methods dealing with things the user doesn't need to know about should be private. Keep the secret recipe a secret :)

Privacy

When creating a class to implementation, be mindful of what you should make private. **Private methods or variables are only accessable or callable by the class the method is in**. Therefore, we should make methods that aren't meant for the user to call private.

Imagine one day you woke up, you wanted your robot servant to bake a cake. `BakeCake()` is something you can do because its a public method. You probably wouldn't need to ever tell him to `MixBatter()` because mixing a batter without a purpose is useless. In this case, `BakeCake()` is public. `MixBatter()` is private because you don't need the method unless you were baking a cake. Leave it to the robot to handle the details.

Private Variables

Private variables are used by the class and only by the class, as that class is the only one that can access them. This is good for cases where you dont need the variable and is only used in manipulation within that class. But what if there are variables we want to change? What if we want to tell the robot to use 3 eggs instead of 2 in it's cake baking method/recipe? You had better hope the robot's manufacture coded in a way to change the private variable `eggs`.

Setters for Private Variables

If we wanted to change a private varible, we would have to use a **setter**. A setter is a function that changes the value of a variable.

Why would we use a setter? Why not just use the normal `eggs = 3` command? This is because the program accessing the class (In this case, us trying to command the robot) cannot access the variable. It is invisible to us. This is to done to make classes simplier, avoid conflict with program variables, and to protect the variables from being set to something that shouldn't work (Imagine if we were allowed to set the eggs to one million. The robot would probably tell us that we don't have enough eggs... Or go to every store and buy out their eggs and put it on our credit cards).

Here's an example function for a setter:

```
1  //The variable is private, so we gotta use a setter.
2  private int eggs = 2;
3
4  //This is a public function, meaning anyone can call this function.
5  public void setEggs(int egg) {
6
7          //This will set the new egg amount if it is a reasonable amount.
8          if(0 >= egg >= 10) {
9                  throw new IllegalArgumentException("Unreasonable amount");
10         }
11         else {
12                 this.eggs = egg;
13         }
14
15 } //setEggs()
```

Getters for Private Variables

Since a variable is private, it won't be visible to us. We'll have to ask the robot to tell us what the variable is. If we are allowed to see it, the manufacturer would have made us a public function to ask the robot to tell us the value. Since the number of eggs used is a private variable, we'll have to ask the robot how much eggs he uses in his recipe by asking it nicely with a `getEggs()` function:

```
public int getEggs() {

        //Nothing too fancy here, just telling the value of the variable eggs
        return eggs;

} //getEggs()
```