Java
John Marchand – jhmarchand@gmail.com
Team 4381 – Twisted Devils
https://github.com/jhmarchand/java4381

Last Week Review

I am not going to be printing as much code this time, so please pull up the files from Github Week6

Person.java.

Notice I have 2 Constructors (a function with the same name as the class, and no return type that gets called when a variable of this class is created.) This allows users of my class flexibility and makes both of the following possible.

```
Person me = new Person("John","Marchand", 47);
Person luke = new Person();

luke.age = 13;
luke.firstName = "Luke";
luke.lastName = "Marchand";
```

If you ever create a class and don't add a constructor, the compiler will create a default constructor for you. It takes no parameters and doesn't do anything.

Notice GetFullName() and GetFullName2(), they both do the same thing, but if I want, I can write it as one line.

PersonMain.java

When you new up a class, you are really creating space in memory to hold that class. Your variable points at that memory. If I do the following code, I have two variables that point at the same spot in memory.

```
Person mySon = luke;
luke.haveABirthday();
```

If mySon.age was 13, after the luke.haveABirthday() call, it will be 14.

Notice we can have arrays of our classes and even functions that return types of our class.

```
static Person FindOldest(Person[] personArray)
```

We can also use functions in functions, just a reminder

```
System.out.println(person.GetFullName() + " is " + person.age);
```

StringCalculator.Java

Classes can represent real world objects, but they can also be utility type classes that do things for us. Imagine a class that takes a equation in the constructor and gives us the answer. Putting the functionality in a class makes it portable so others can use it.

```
                    StringCalculator calculator = new StringCalculator(equation);
                    System.out.println(calculator.getFirstNumber() + " " + calculator.getOpertor()
+ " " +  calculator.getSecondNumber() + " = " + calculator.getAnswer() );
```

Speaking of utility classes, remember Java.Util.Scanner that we use to get inputs? That's a utility class also. It has a really long name doesn't it? Most classes available to us like that will have long names. It's so that if 2 classes have the same name, we can differentiate them. In this case Scanner is the class name, and Java.Util is the package name. Sometimes the long name clutters up code, so we can use an import statement. If I put the following import statement at the top of file, above the class declaration, I just need to use the class name.

```
import java.util.Scanner;
```

No we can declare a scanner like this.

```
            Scanner input = new Scanner(System.in);
```

This gives us a much cleaner look.

Looking at StringCalculator, we see a new modifier called private. For each class field (member) or function you can define them as public or private. By default they will be public. Public means they are usable outside of the class. For private items, only the class can use them. Sometimes you want to make fields private so that users of your class can't change them. Consider my StringCalculator class. If someone changed firstNumber, my answer would be wrong. Same for many of fields. So all the user can do is call a get function. For the equation, I also have a set function, and when that is called, it processes the equation so that all of my variables are correct. Notice the same function is called in either the constructor or the setEquation function.

Class Hierarchy

Sometimes many classes will have lots of similar data or functionality. Imagine a class called Pet. It probably has a name and an owner(our Person class). Now what if I wanted a class called Cat, or Dog? Wouldn't they have the same fields? Instead of duplicating all of those fields, Java allows us to inherit the functionality. So if we say Class Dog Extends Pet it means our Dog class automatically has all the members and functions of our Pet class without having to type them.

We can also change the behavior. Imagine the Pet class has a function called speak() that just Prints Hi to the screen. We can override that function and for the dog class it would print woof to the screen. For a cat, meow.

We can also add class specific functions like getAgeInDogYears() that is only on the dog class.

We can have a Pet variable point at the memory of a Cat object. Below will print out Meow even though it's a Pet variable because its pointing at a cat.

```
Pet myPet = new Cat("Gruffy",7,someOne);
myPet.speak();
```

Even the printPet function takes a pet, but its fine if you pass in a Cat or Dog.

SubClass declaration and constructor. Notice the syntax in the class declaration where we say Cat Extends Dog. Also notice, the Cat constructor calls a function called super. Remember, Cat is the subclass. Pet is the super class. So here it says call the super classes constructor, and really that's all we need to do here. We could do more if we had some cat specific parameter passed into our constructor.

```
public class Cat extends Pet {

	public Cat(String aName, int aAge, Person aOwner) {
		super(aName, aAge,aOwner);

	}

	void speak()
	{
		System.out.println("Meow");
	}

}
```

Do you know anyone who has a pet pig? I don't, but just in case, can you make a Pig class and use it in your PetMain?

ROBOTS!

Lets create our own Robot classes! Design in class.