Java Objects

Academic Resource Center

THE ARC.

What is a Java Class?

- It may contain variables
- It may contain method definitions
- It may contain class definitions

THE ARC.

Object vs Class

- A Class is like the idea, or the definition of an actual thing. Car can be a Class, but "A Car" would be what is called an instance of that class or, an Object.

- To be more specific, we might say that every car has a make and model, that would be the Class. From there we could make Blue Corvets, Red Mustangs, and Black Porsche, all Objects.

THE ARC.

How? Let's do an Example:

```
class Car{
 String make;
 String model;
}
```

THE ARC.

Implementing our Car

```
class Example{
  public static void main(String[] args){
   Car car1 = new Car();
   car1.make = "Red";
   car1.model = "Sedan";

   Car car2 = new Car();
   car2.make = "Blue";
   car2.model = "Mustang";
  }
}
```

THE ARC.

Constructors

- We can add our first method to the car class, a constructor, to make it easier to create Car objects:

```
public Car(String make, String model){
    this.make = make;
    this.model = model;
}
```

- Note that `this.make` is the Car's `make` String, and `make` is the variable passed in. The same is true for `this.model` and `model`.

THE ARC.

Implementing our Car (Again)

```
class Example{
  public static void main(String[] args){
   Car car1 = new Car("Red", "Sedan");
   Car car2 = new Car("Blue", "Mustang");
  }
}
```

- Much easier.

Hiding our Variables

- It is quite often the case that you do not want other using your object to be able to mess with it's variables any way they choose, in this case you need to make them private. Then what we accessor and mutator methods can be used to interact with them in specific ways.

Hiding our Variables (Cont.)

- So we'll make our variables private:

- `private String make;`
- `private String model;`

- This means trying to access them like we did the first time will no longer work. Java won't allow it to happen.

Hiding our Variables (Mutators)

- Adding a mutator method, will allow us to change the value of a variable from outside the object, like in our main class.

```
public void setMake(String make){
    this.make = make;
}
```

- Now we can change the value of make, even though it's `private`, because the method `setMake`, is not `private`, and we can access that.

THE ARC.

Hiding our Variables (Mutators)

- We can even add a rule now, making sure that the user is at least changing the make to something at all.

```
public void setMake(String make){
    if (make.length() > 0){
        this.make = make;
    }
}
```

- Now setMake has a small rule.

THE ARC.

Hiding our Variables (Accessors)

- Adding an accessor method, will allow us to see the value of a variable from outside the object.

```
public String getMake(){
    return make;
}
```

- Now we can see the value of make, in the same way `getMake` worked.

Implementing our Car (Again)

```
class Example{
  public static void main(String[] args){
   Car car1 = new Car("Red", "Sedan");
   System.out.print(car1.getMake());
   car1.setMake("Yellow");
   car1.setMake("");
   System.out.print(car1.getMake());
  }
}
```

- This will print `Red`, then `Yellow`.

THE ARC.

toString() and Why

- Every class in Java has several methods by default, `toString()` is one of them. `toString()` is special, because if a method is expecting a String (such as `println()`) but it recieves an Object, the `toString()` method of that Object will be called to fill in the value.
- `toString()` returns a String variable that represents the variables of the object in a clear and concise way. Let's make one for Car.

Implementing toString()

```
public String toString(){
    return make + " " + model;
}
```

- Done, now we can do this:

```
Car car =  new Car("Black", "SUV");
System.out.println(car.toString());
System.out.println(car);
```

- Both of the prints will be the same: `Black SUV`

THE ARC.

Reference

- Compiled by Corey Sarsfield

THE ARC.