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

Week 7
Lets consider the following class that represents someone in my family.

```
public class Marchand implements Person {

	String firstName;
	final int birthYear;
	static String homeTown;
	final static String LastName = "Marchand";

	Marchand(String aFirstName, int yearOfBirth) {
		firstName = aFirstName;
		birthYear = yearOfBirth;
	}

	void changeFirstName(String firstName ) {
		this.firstName = firstName;
	}

	public String getFullName() {
		return firstName + " " + LastName;
	}
}
```

Final

Final is a keyword that allows us to set something once in the constructor, but after that we can never change it. This makes sure that we don't accidentally write code later that does it, or that users of our class(its public) can change it. Each Marchand has a birthYear, and once you create a Marchand, you cant change their birthYear.

Static

The static key word is a member of the class that is stored with the class itself as opposed to an instance of the class. For a static class variable, it doesn't matter now many Marchand's you new up, they all share one and only one homeTown. Also, you can access Marchand.hometown before you even new up the first Marchand.

This and Scope

Look at the changeFirstName. Notice that it has a parameter called firstName which is also the name of a class variable. So if I wrote firstName = firstName, I might know what I mean, but the compiler doesn't automatically know. There are rules that define which firstName I am talking about. The scope of firstName inside that constructor is the parameter firstName and not the member Variable firstName. So I need a way to tell the compile I want the classes member variable. To do that, I use the this keyword. It means this instance of the class. In the following code the function replaces this with john since john is the instance calling the function.

```
john.changeFirstName("Johnny");
```

Final Static

This allows us to set something 1 place, but use it any where. In this case, I could type "Marchand" everywhere I used it, but its simpler and safer to assign it to final static like this. This insures we don't misspell it.

For FTC, we will use this for channel Mappings. You might use final static rightMotorChannel = 1; This makes your code readable, and also, when the build team says, oh we reversed the channels, you can quickly change it to a zero. Otherwise you would have to search your code for 1s and know which ones to change. Some 1s might need to be changed, but others wouldn't. Knowing which ones can be hard, so its best to use final statics (also know as constants).

Interfaces

Interfaces are definitions that say a given class will have these functions. So I created a Person Interface.

```
public interface Person {
	String getFullName();
}
```

Interfaces have no code, just a definition for some functions. This only has one function, but you could have several. Notice the Marchand class implements Person. This means that if I did not have a getFullName() function in that class, the code would not build. I can have other functions and members as well, but I have to have the getFullName one. I have another class that implements Person called student.

```
public class Student implements Person {
	String name;
	int grade;

	public String getFullName() {
		return name;
	}
}
```

That allows me to also create a function like below that will work with the Marchand class or the Student class, even though they have no relation. All the function cares is that they implement the Person interface.

```
	static void printPerson(Person person){
		System.out.println(person.getFullName());
	}
```

Please note that you do need to use the public keyword when implementing the functions for a interface.

Static Classes.

Remember the StringCalculator class? We had to new it up, but we didn't really ever need more than one did we? Lots of times we will have utility classes and they will be static classes so that we don't have to new them up, we can just use them. I changed all functions and variables to static and now I can just do this.

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

So I am just using the class name as opposed to newing up an instance of the class.

If you write a static class function, you can only use static members in that function.

Robot Ball Game

Rules of the game. 2 robots on a field. Each robot starts under the basket on opposite sides of a court. A ball is tossed out a random distance from 1 to 10 feet. The robot must go get the ball, shoot. The next ball will not be thrown out until the robot returns to underneath the basket.

The robot to make the most baskets wins.

The game goes until both robots are out of power.

Create a new project called RobotGame. From GitHub, add Battery.java, Motor.java, BallShooter.java, IRobot.java, Robot.java, RobotGameMain.java to the project.

Get the project compiling and play the game a few times.

Read through the code get a good understanding of the classes and the RobotGameMain.playGame() function.

Robot Name and points should probably be a part of the robot class, can you modify the robot class and robot Game Main to do that?

This block of code is repeated twice, once we have moved name and points into the robot class, can you moved this code block into a function called PlayARound(Robot robot)?

```
if ( robotB.hasPower() )
{
	distanceToBall = GetBallThrow();
	System.out.println("Robot B ball thrown " + distanceToBall + " feet.");
	if (robotB.getAndShootBallandReturn(distanceToBall))
	{
		bPoints++;
		System.out.println("Robot B made a basket!");
	}
	else
	{
		System.out.println("Robot B missed!");
	}
}
else
{
	System.out.println("Robot B out of power!");
}
```

Look at BetterBattery.java – we have a new keyword between public and private called protected. Protected means private except for subclasses.

Add BetterBattery.java to the project. In RobotGameMain, modify the line where robotA is created, pass in a BetterBattery. Does that help robotA win?

Can you create a more efficient motor and give it to one of the robots? Does that help it win?

Can you create a better ballShooter and give it to the other? Who wins now?

Can you create a subclass of Robot called SmartRobot? A smart robot will go get the ball, drive back to the basket and take a easy shot instead of shooting from far outside.

Can you create a SmartBattery class that reports how much battery it has left, and create a SmarterRobot class that only drives closer if its going to have enough power to both drive and shoot? Otherwise it takes its last shot with the last bit of power?

Notice the interface doesn't define that a robot has a motor. What if instead of a motor it has net on the end of a rope that it throws out to grab the ball and bring it back? Maybe it uses less power, but doesn't always grab the ball. Could you create a brand new robot class not derived from robot that takes a net in its constructor?