Introduction to Java Programming

Part 6: OOP Finished (for now) - GitHub, Electronics

REVIEW FROM LAST TIME

Important Access Modifiers

Modifiers are keywords used to define where a field or method can be accessed. They are useful when defining expected and desired behavior for users of a library or set of code. It is good practice to always use the most restrictive modifier possible.

Modifier	**Class**	**Package**	**Subclass**	**General**
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
Default (no modifier)	Yes	Yes	No	No
private	Yes	No	No	No

Important Non-Access Modifiers

Modifier	**Function**
static	Used to declare fields and methods independent of any instance of a class. Used to create **class variables**.
final	Used to indicate a variable can be initialized only once. Effectively, this makes a variable a constant.
abstract	We will use this later - indicates that a class might not provide implementations of its methods and cannot be instantiated.
synchronized, volatile	Used in conjunction with multithreading.

Method Declaration

```java
public void doMethod(int in1, String in2) {
```

This is an access modifier controlling which classes can run (call) the method.

This is the return type. “void” means the method doesn’t return any values, but return types can be any class or primitive type.

This is the method name, named according to general convention.

Within the parentheses are the method’s arguments, which function as local variables within the method’s code.

Programming Tricks: Recursion

- Recursion is a programming trick in which a method calls itself during its own execution.
- This may seem weird at first, but if you modify some sort of variable in a similar way as you would a loop, you can get a single method to behave in a similar way as a for loop or a while loop.
- Let’s take a look at how this works!

```
public void doThing(int i) {
    doThing(i + 1);
}
```

Encapsulation

```
    //Define a setter method
    public void setMessage(String str)
{
        this.message = str;
    }

    //Define a getter method
    public String getMessage() {
        return this.message;
    }
```

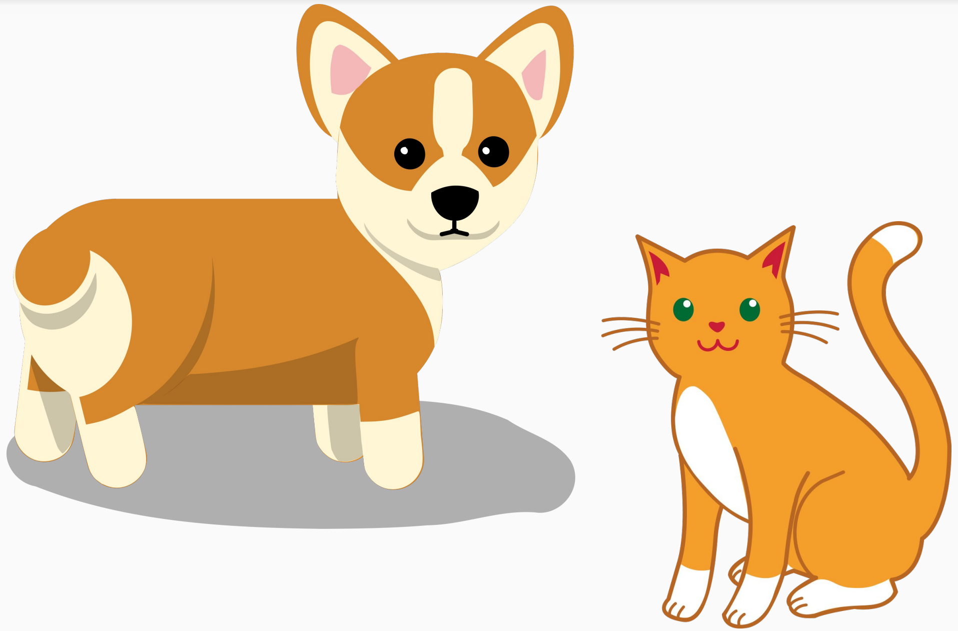
- Encapsulation is an object oriented programming concept that essentially dictates that all instance variables should be declared private, and getter and setter methods should be used to control access to them.
- This makes it easier for others to use your code - there is no ambiguity about whether they are misusing your variables.

Variable Scope

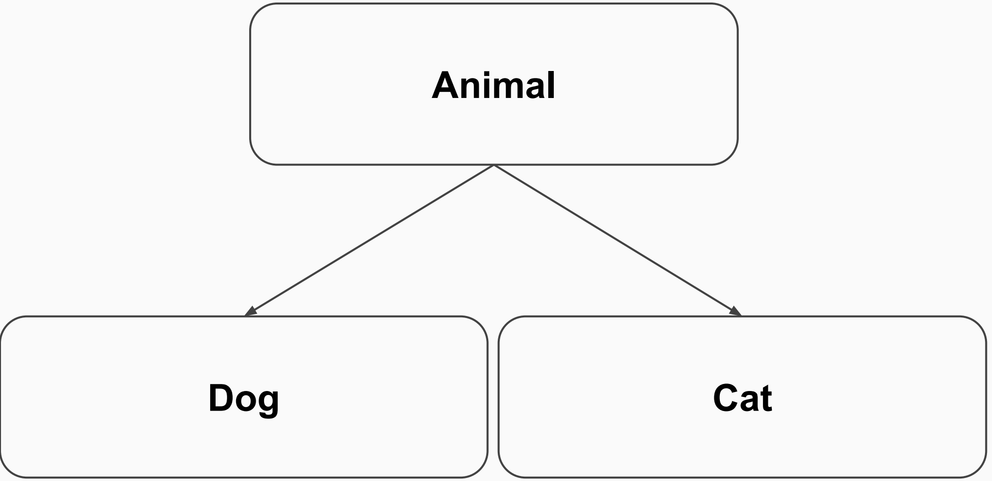
- We have talked about access modifiers, which define which classes can access another class’s fields or methods.
- What about local variables and method arguments? Where can we access variables?
- The **scope** of a variable refers to where the variable can be accessed, and is determined by where in your code it was declared. The rules are:
- A variable declared within a class but outside of any method is known as a member variable (including both instance and class variables), and can be accessed from any method within the class. It can be accessed from outside the class according to its access modifier.
- A variable declared within a method is local to that method only, and ceases to exist when the method’s execution is over.
- A variable declared within a block (such as an if statement or while loop) exists only within that block.

An Introduction to Inheritance

- Suppose we have to program a representation of cats and dogs that can walk, eat, and make noise. We discussed two ways of doing this.
- At first, it seemed like the best way to do this would be to create two separate classes. Cats and dogs have different states and behaviors, so it would make sense for them to be represented by different classes.



An Introduction to Inheritance



- Then, we noticed that cats and dogs share a lot of behaviors and states in common - they both eat, walk, have four legs and a tail - and these similarities translate to code repetition.
- In general, code repetition is inefficient and undesirable. It makes code harder to read.
- **We introduced a third class called Animal that handled all the behaviors cats and dogs had in common**

An Introduction to Inheritance

- Java has a feature that lets us do this! It is called **inheritance**. One class can inherit from one other class. The class that inherits the other is called the subclass, while the class the subclass inherits from is called the superclass.
- A subclass has the same fields and methods as its superclass, but can only access them if their access level is protected or public.
- This allows us to share behaviors between similar classes. Interesting note - every single class in Java inherits from the Object class.
- Every subclass shares its own instance with an instance of its superclass, which can be accessed with the **super** keyword. This functions similarly to the **this** keyword.

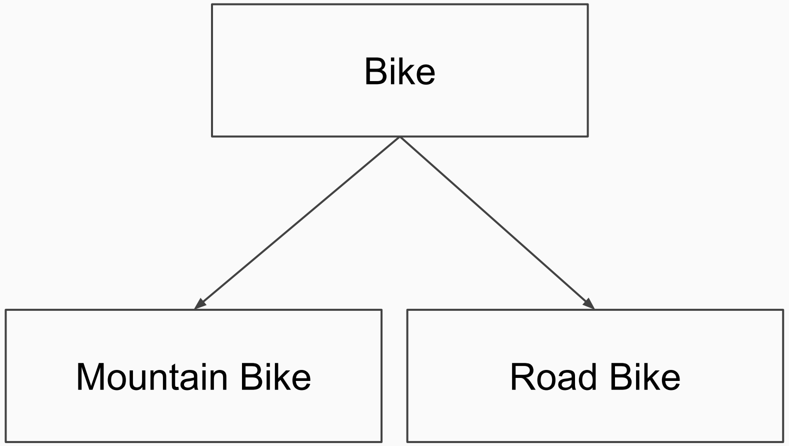
END OF REVIEW

Has-A vs. Is-A Relationships

- All of the Object-Oriented Programming concepts we have discussed so far can be grouped into two main relationships: Has-A and Is-A.
- A class **has a** method or a member variable (class or instance) - in essence, a behavior or a state.
- A class **is a** superclass, interface, or abstract class that it inherits from or implements
- In our example, Dog **is an** animal, like Cat **is an** animal.
- Dog **has a** name, Dog **has a** behavior called bark.
- This concept is useful to remember when discussing inheritance and interfaces, especially when dealing with **polymorphism**

Inheritance Overview

- Inheritance allows classes that share code to “inherit” from one common superclass, as we have discussed.
- One superclass can be inherited by many different subclasses, but a subclass can only inherit one superclass.
- Nothing special has to be done to a class to allow it to be a superclass
- Private or protected modified fields or methods are not accessed by the subclass.



Abstract Classes

- An abstract class is a class that may or may not contain abstract methods - that is, methods without any body.
- An abstract class cannot be instantiated, but can be inherited from.
- It allows you to fully define some parts of the class that will be inherited, but leave the rest up to the subclasses to implement in a specific way.
- We turned our Animal class into an abstract class at the end of last time.

Interfaces

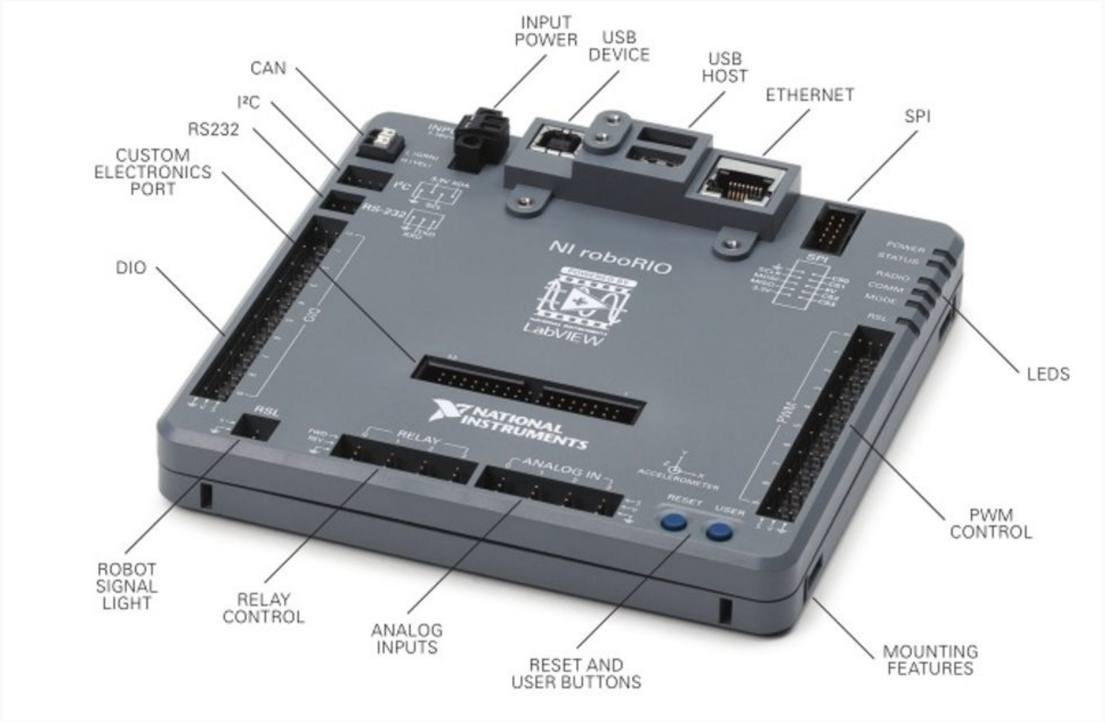
- Interfaces are like abstract classes, but take it a step further. They define a set of methods without implementations and constants.
- Interfaces are not inherited, but implemented.
- Interfaces can be thought of as a contract between the class that implements them and the compiler, promising that the class will provide an implementation for the specified methods.
- A class can implement any number of interfaces.

Introduction to Git and GitHub

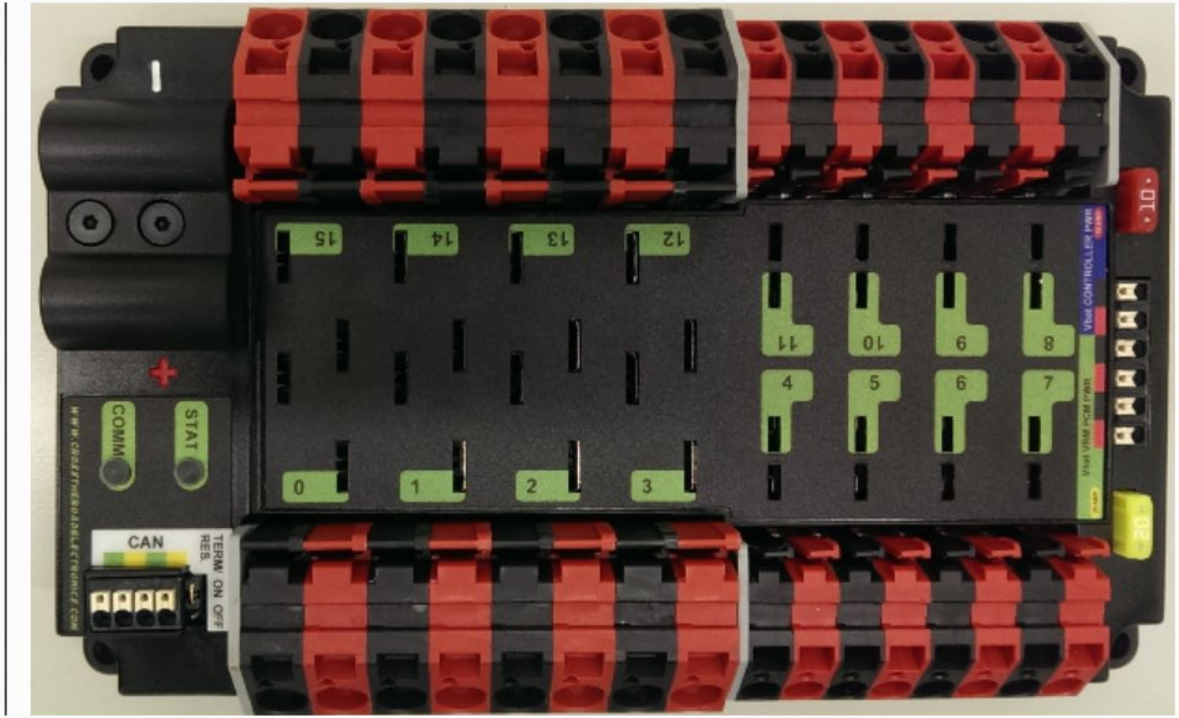
Introduction to FRC Electronics

(Referencing Team 4994's Electronics Bible + Official Materials)

RoboRio - End 1 of CAN bus

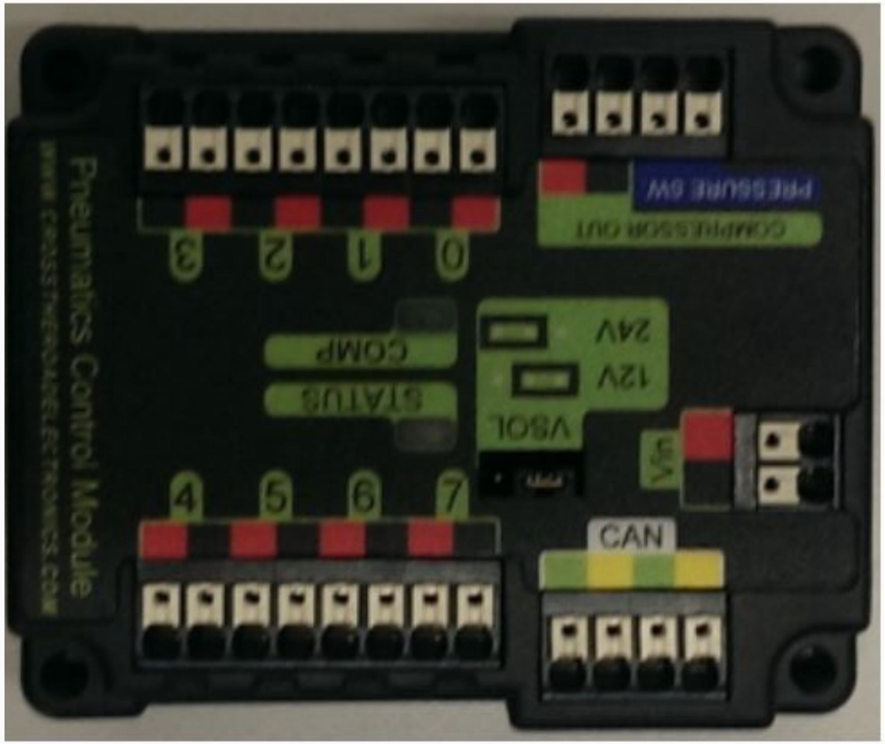


PDP - End 2 of CAN Bus

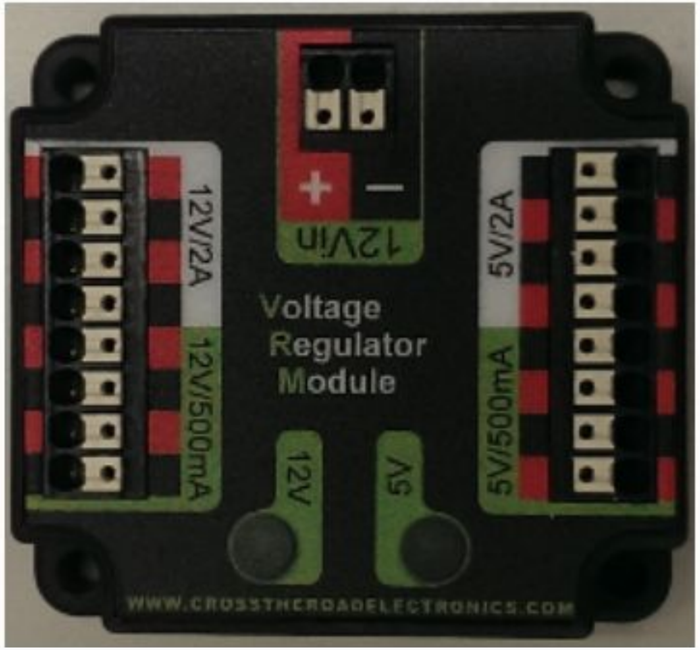


Talon SRX

Pneumatics Control



VRM


12V/2A
12V/500mA
+
—
12Vin
Voltage Regulator Module
12V
5V
5V/2A
5V/500mA
WWW.CROSSTHEROADELECTRONICS.COM

Robot Radio

Camera