Introduction to Commands

Why Commands?

Commands represent actions the robot can take. Commands enforce structure into how we tell the robot to make it more predictable. All commands share the same structure so once you learn that structure you will be able to write complex actions with the simple building blocks of Commands!

The 4 Parts of a Command

Basic commands have 4 components that are related to the action desired.

Initialize

The Initialize method is called only once at the beginning of the command

Execute

the execute method is called every periodic cycle (every 20ms) code you put in here is run over and over again until the command is finished.

isFinished

the isFinished method is where we determine if the command should be finished it is also run every periodic cycle (every 20ms). Some commands we want to run until another command needs the subsystem and therefore we will return false. Others run until a condition is met and that logic is put within the isFinished method to return true when the command is supposed to end

End

This method is called at the End of the command once isFinished is True or another command needs the subsystem. Here we often set things back to the way they were before the command ran.

Writing Commands in English

First write out what you want the robot do so using template

First _____, then do _____, Until _____, Finally do _____

For example if we want the robot to score a game piece from the claw it could be

First **Set claw motor to spit**, then do _____, Until **Claw no longer sees game piece**, Finally do **set claw motor to zero**

> Notice how we don't always have to fill out every box

Or another example, we want the robot to drive around

First _____, then do **command drivetrain based of current joystick values**, Until _____, Finally do **set drivetrain motors to zero**

Writing Command in Code (class)

We can create a new file and class to hold our command in that will look like. Command names are always Capatalized because they are the name of a java class

```
public class CommandName extends CommandBase {

  Subsystem subsystem = Subsystem.getInstance()

  public CommandName() {
    addRequirements(subsystem);
  }

  @Override
  public void initialize() {

  }

  @Override
  public void execute() {

  }

  @Override
  public boolean isFinished() {

  }
```

```
  @Override
  public void end(boolean interrupted) {
  }
}
```

Notice first how the class extends and inherits from CommandBase. From there we have the 4 basic parts of a command for our code to be executed.

inside the constructor we had the line addRequirments(). What if I asked you to type an essay and juggle? You cant because you would need your hands to only be doing one. The same is true for the robot we cannot ask the *same* hardware to do 2 things at once. The same is true for commands however you must define what subsystems the command uses so that no two commands that share a subsystem can run at the same time

Writing Commands in Code (Composition)

often we don't want to have to define a whole file for simple command so we use a process wpilib calls composition. WPILIB has many options for writing commands depending on which of the 4 components we need we can write commands very similarly to as if we were writing them out in plain English.

runOnce

Within a subsystem if only need to call a function once or use the initialize of a command you can do the following

```
public Command doMyThing() {
  return this.runOnce(this::methodName)
}
```

Here we call the define a function doMyThing() which will return the command for use. The command is created by the runOnce() method. This method takes in the name of a function in a special format called a Lambda. This way lets the code call that function within the command instead of running the function right now.

> Lambda expressions can look scary but are just the name of a function that the computer can use to call a function later. They can look two different ways one being the object::method or () -> object.method() we can even write small lines of code inside a lambda like () -> { awesome code line1; great code line 2}

runEnd

if instead inside a subsystem we need the initialize and the end we can instead use the runEnd() function

```
public Command doMyThing() {
  return this.runEnd(this::initialMethod,this::endMethod)
}
```

This method is super useful for running commands where we want to at the beginning set a motor to a value and at the end zero the motor out. For example if I want when I press a button for a motor to spin, runEnd is how we would accomplish that

Modifying Commands

With just runOnce and runEnd we can accomplish a lot but sometimes we would like to change the isFinished or modify how the command is to be run. WPILIB Commands have a many methods that modify the command to do what we want called decorators.

until

the until decorator takes a command and allows us to set a condition for when the command should end. We pass in a Lambda for a function or code that will return a boolean that is what we want for isFinished

```
private boolean isMySensor() {
  return inputs.mySensorValue
}

public Command doMyThing() {
  return this.runOnce(this::initialMethod).until(this::isMySensor)
}
```

withTimeout

A special case is we often want a command to run for a specified amount of time or even exit even if it it not done after the time. We can accomplish this behavior with the withTimeout

```
public Command doMyThing() {
  return this.runEnd(this::initialMethod,this::endMethod).withTimeout(numSeconds)
}
```

References

- WPILIB Commands Introduction
- WPILIB Command Composition

Revision #5
Created 12 July 2023 19:51:21 by jheidegg
Updated 12 July 2023 20:57:16 by jheidegg