Understanding a Subsystem Class

589 Falkon Robotics

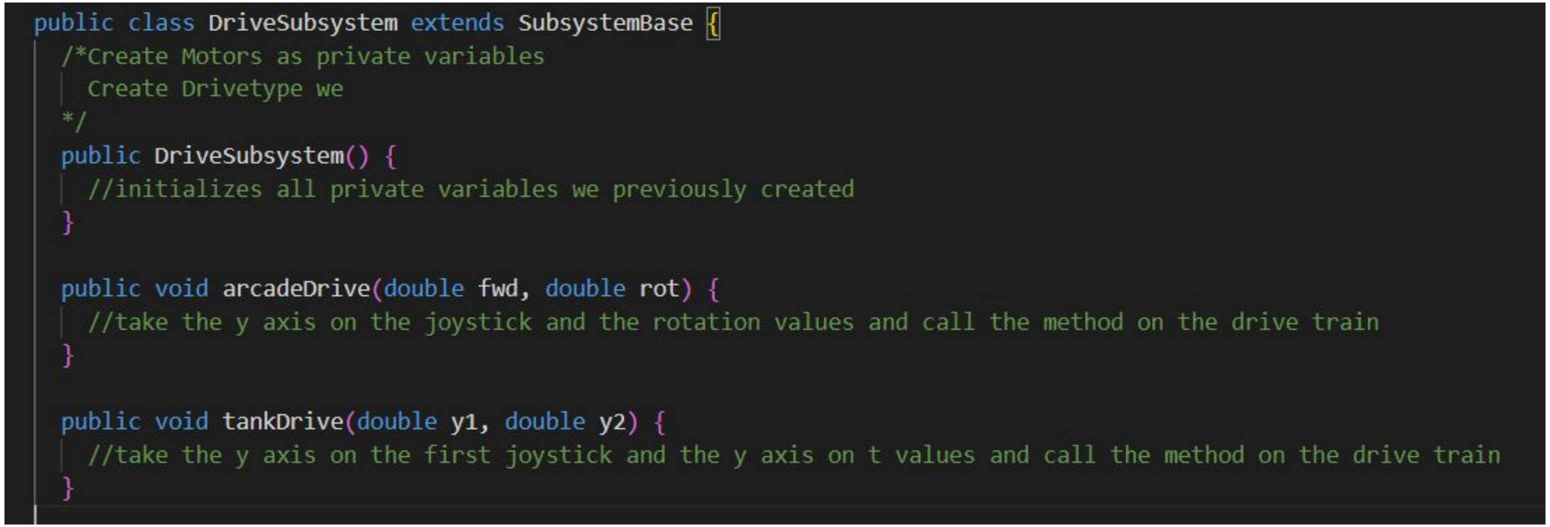
Prerequisite

Before continuing, please read up on “What Is ‘Command-Based’ Programming?” by FRC to further understand how we incorporate subsystems!

What is a subsystem class?

A subsystem class is usually for one function of a robot and constructs/initializes objects that control a specific part on the robot.

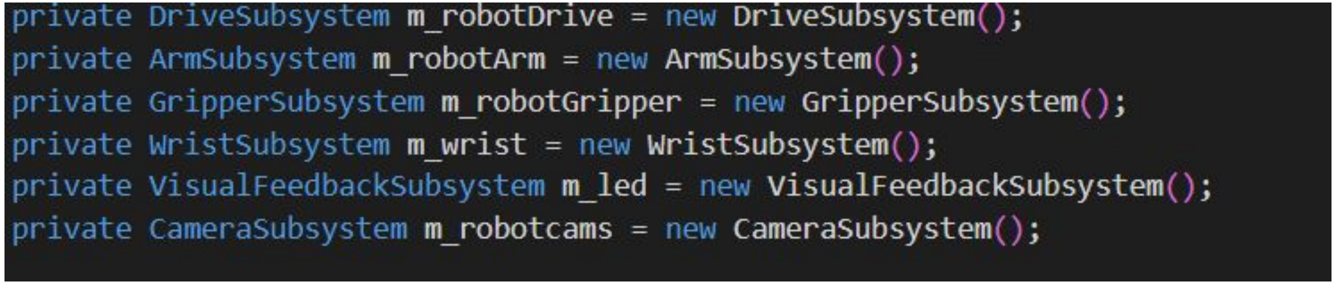
Here is an example of this is a DriveSubsystem with comment code.



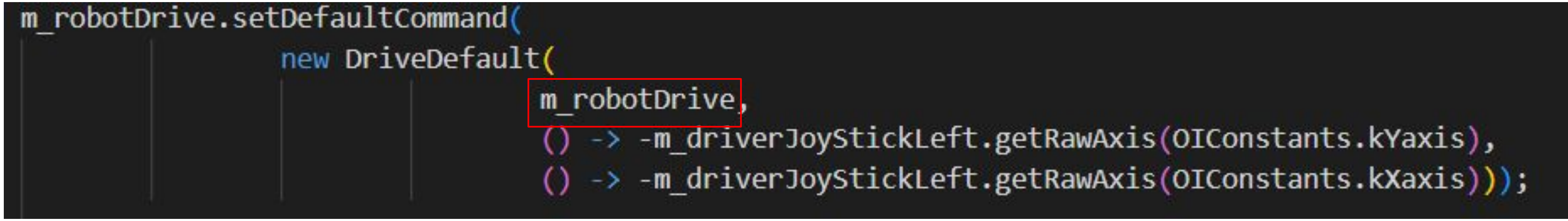
```java
public class DriveSubsystem extends SubsystemBase {
  /*Create Motors as private variables
    Create Drivetype we
  */
  public DriveSubsystem() {
    //initializes all private variables we previously created
  }

  public void arcadeDrive(double fwd, double rot) {
    //take the y axis on the joystick and the rotation values and call the method on the drive train
  }

  public void tankDrive(double y1, double y2) {
    //take the y axis on the first joystick and the y axis on t values and call the method on the drive train
  }
```

In the perspective of RobotContainer

We create one of each subsystem to represent every function of the robot. This can be helpful to use as commands use Subsystems to call methods in the said subsystem.



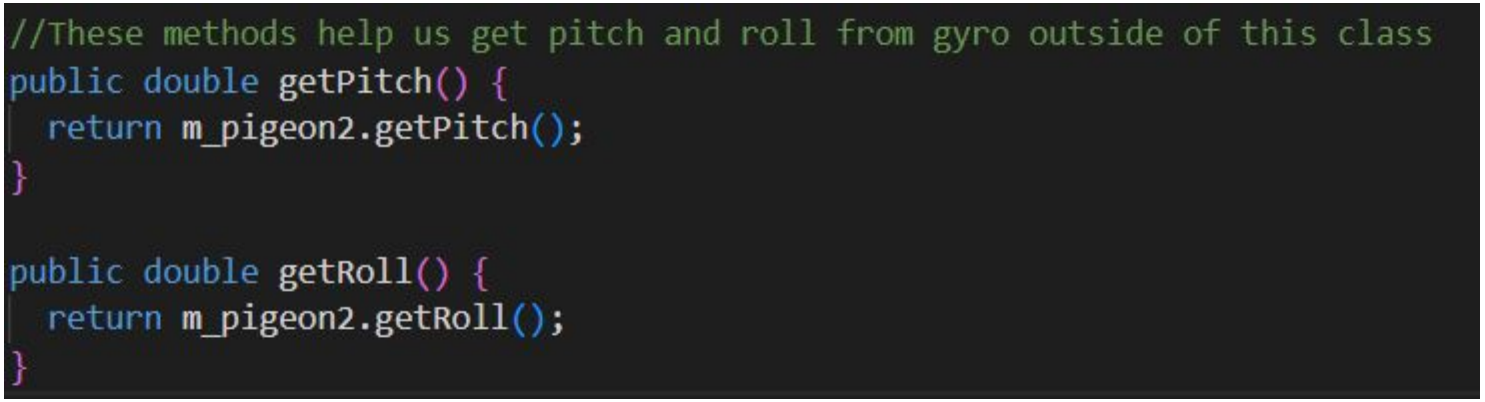
```
private DriveSubsystem m_robotDrive = new DriveSubsystem();
private ArmSubsystem m_robotArm = new ArmSubsystem();
private GripperSubsystem m_robotGripper = new GripperSubsystem();
private WristSubsystem m_wrist = new WristSubsystem();
private VisualFeedbackSubsystem m_led = new VisualFeedbackSubsystem();
private CameraSubsystem m_robotcams = new CameraSubsystem();
```



```
m_robotDrive.setDefaultCommand(
        new DriveDefault(
                m_robotDrive,
                () -> -m_driverJoyStickLeft.getRawAxis(OIConstants.kYaxis),
                () -> -m_driverJoyStickLeft.getRawAxis(OIConstants.kXaxis)));
```

Accessor Methods

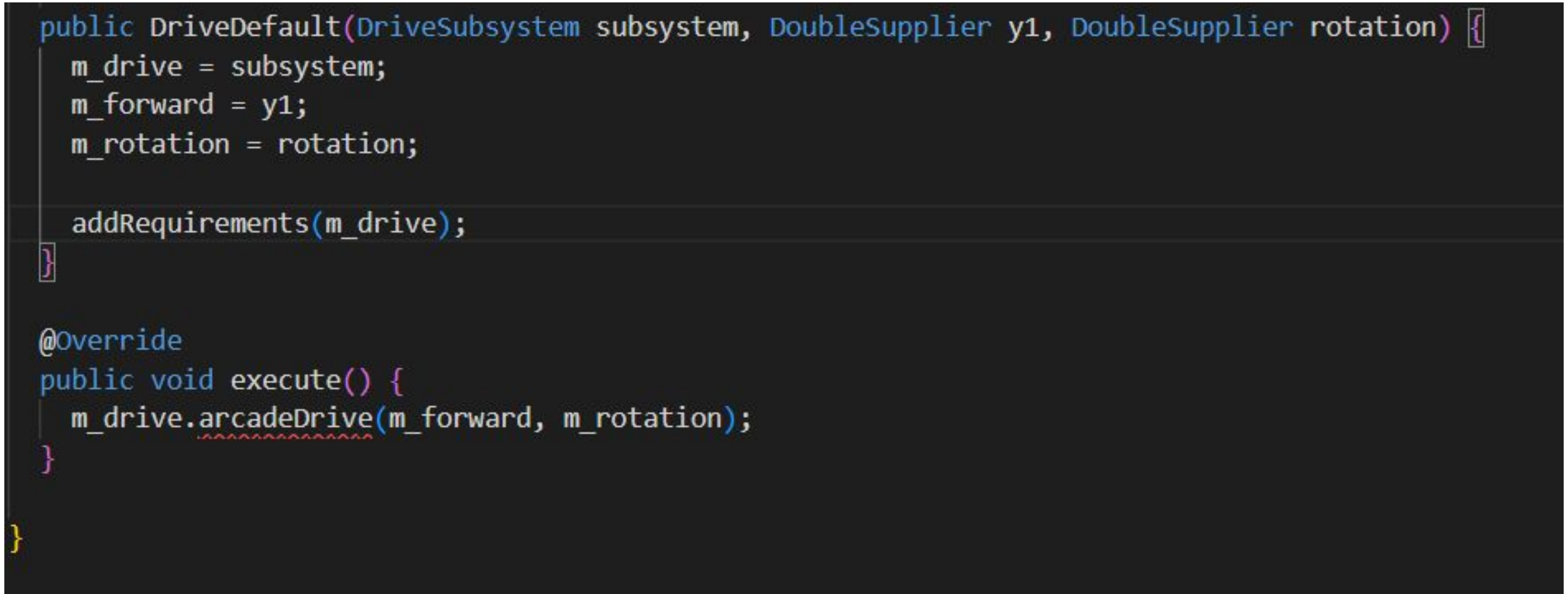
Using accessor methods we can access object values without having to instantiate them more than in more place.



```
//These methods help us get pitch and roll from gyro outside of this class
public double getPitch() {
  return m_pigeon2.getPitch();
}

public double getRoll() {
  return m_pigeon2.getRoll();
}
```

Helper methods and their uses

Helper methods like arcadeDrive() can be used in commands so we can expose these objects without recreating them, here is a default drive command or DriveDefault that is set to periodically loop

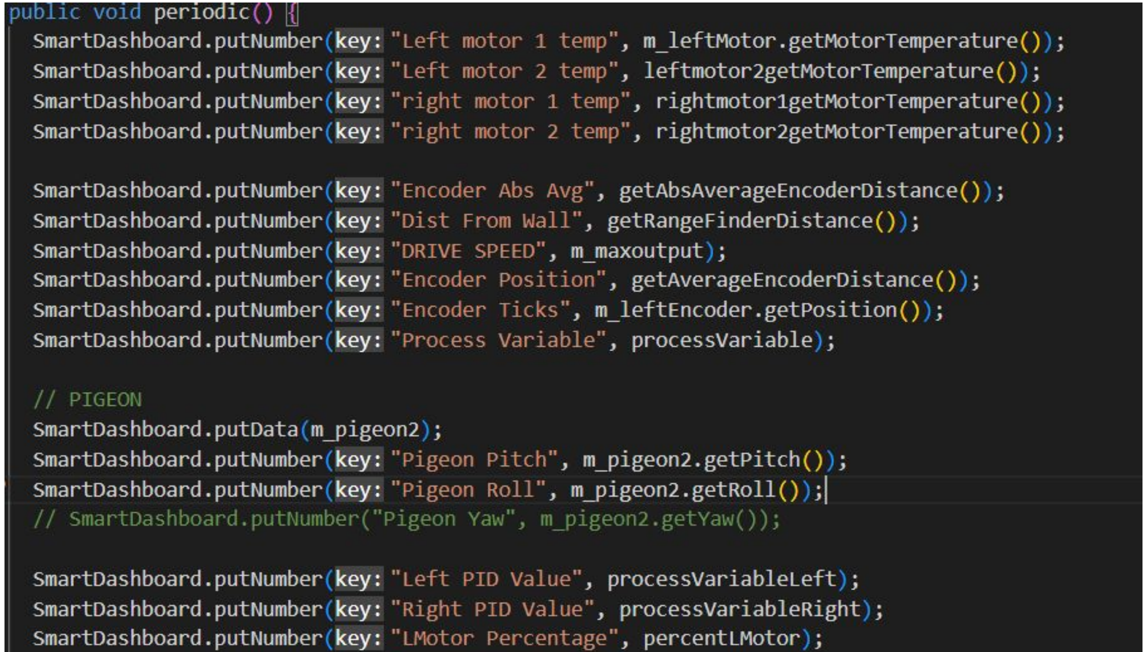


```
  public DriveDefault(DriveSubsystem subsystem, DoubleSupplier y1, DoubleSupplier rotation) {
    m_drive = subsystem;
    m_forward = y1;
    m_rotation = rotation;

    addRequirements(m_drive);
  }

  @Override
  public void execute() {
    m_drive.arcadeDrive(m_forward, m_rotation);
  }


}
```

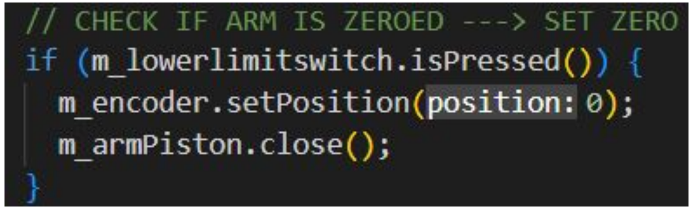
The periodic() method

The periodic method is exactly how it sounds, this method is called in real time repeatedly. This can be used for safety reasons like here:

Or displaying to SmartDashboard values for debugging or drivers like here:



```
public void periodic() {
  SmartDashboard.putNumber(key: "Left motor 1 temp", m_leftMotor.getMotorTemperature());
  SmartDashboard.putNumber(key: "Left motor 2 temp", leftmotor2getMotorTemperature());
  SmartDashboard.putNumber(key: "right motor 1 temp", rightmotor1getMotorTemperature());
  SmartDashboard.putNumber(key: "right motor 2 temp", rightmotor2getMotorTemperature());

  SmartDashboard.putNumber(key: "Encoder Abs Avg", getAbsAverageEncoderDistance());
  SmartDashboard.putNumber(key: "Dist From Wall", getRangeFinderDistance());
  SmartDashboard.putNumber(key: "DRIVE SPEED", m_maxoutput);
  SmartDashboard.putNumber(key: "Encoder Position", getAverageEncoderDistance());
  SmartDashboard.putNumber(key: "Encoder Ticks", m_leftEncoder.getPosition());
  SmartDashboard.putNumber(key: "Process Variable", processVariable);

  // PIGEON
  SmartDashboard.putData(m_pigeon2);
  SmartDashboard.putNumber(key: "Pigeon Pitch", m_pigeon2.getPitch());
  SmartDashboard.putNumber(key: "Pigeon Roll", m_pigeon2.getRoll());
  // SmartDashboard.putNumber("Pigeon Yaw", m_pigeon2.getYaw());

  SmartDashboard.putNumber(key: "Left PID Value", processVariableLeft);
  SmartDashboard.putNumber(key: "Right PID Value", processVariableRight);
  SmartDashboard.putNumber(key: "LMotor Percentage", percentLMotor);
```



```
// CHECK IF ARM IS ZEROED ---> SET ZERO
if (m_lowerlimitswitch.isPressed()) {
  m_encoder.setPosition(position: 0);
  m_armPiston.close();
}
```

Lab

With the information you know, create a simple DriveSubsystem with three public methods, two that use arcade and tank drive, and a third that gets the encoder value and returns it. In the periodic method send the encoder temperature to the SmartDashboard. Refer to the SparkMax Documentation for initializing the Motors and encoders.