2019 Team 61 Robot Code Revisions

```java
public abstract class MoveWithLimitSwitches extends Command {
    protected LSLevelSubsystem m_lslevelsubsystem;
    private int destination;
    protected boolean reachedDestination;
    private final double speed = 0.5;

    public MoveWithLimitSwitches(int destination) { this.destination = destination; }

    @Override
    protected void initialize() { reachedDestination = false; }

    @Override
    protected void execute() {
        m_lslevelsubsystem.checkLocation();
        if (m_lslevelsubsystem.getLocation() < destination) {
            m_lslevelsubsystem.move(speed);
        } else if (m_lslevelsubsystem.getLocation() > destination) {
            m_lslevelsubsystem.move(-speed);
        } else {
            reachedDestination = true;
        }
    }
```

By Kevin Downing

11/4/2019-11/8/2019

Objectives:

- Condense the code for the arm, front lift, and rear lift, which all use the limit switches for movement and are near-identical versions of each other, to just one file for a subsystem and one file for a command.
- Add a feature in the code that will move the arm/robot away from the end limit switches when they are pressed.

Prerequisites:

- Understand advanced Java syntax including concepts such as polymorphism and abstract inheritance.
- Be familiar with WPILib API's classes, method, and command-based programming structure.

Procedure:

1. Create an abstract class called LSLevelSubsystem and have it extend from the Subsystem class. This will act as a template for the subsystems that contain limit switches (arm, front lift, and rear lift).
2. Make the following global variables in the LSLevelSubsystem class:
    a. A LimitSwitch array to contain the limitswitches in the subsystem.
    b. An integer variable called location to store the location of the arm/robot.
    c. A double variable called awayFromEdgesSpeed and declare it as 0.2. This is the speed for a later method that will move the arm/robot away from the edge limit switches.
3. Write the following code for the constructor:

```
public LSLevelSubsystem(String name, int[] levels) {
    super(name);
    LSLevels = new LimitSwitch[levels.length];
    for (int i = 0; i < LSLevels.length; i++) {
        LSLevels[i] = new LimitSwitch(levels[i]);
    }
    location = 0;
}
```

Figure 1-1

In Figure 1-1, String name and int[] levels are found in the parameters. The name String is sent to the Subsystem class through super() and the int[] levels stores the port numbers of the limit switches of any given subsystem. The port numbers are used to create a LimitSwitch instance by having a for loop copy down the port numbers into the LimitSwitch.

4. Create an abstract method called setSpeed(). Every subsystem with limit switches will have to have this method and it will be used to set the speed of the motors.
5. Write two boolean methods that return whether the bottom or top end limit switches have been switched.
6. Create a method called move() that has if statements that will run setSpeed only if the end limit switches aren't pressed. If the end limit switches are pressed, run setSpeed() to a fixed value that will run the arm/robot away from the end limit switches.
7. Write a setLocation() and getLocation() method to set and get the value of the location of the arm/robot.
8. Create a method called checkLocation() that has a for loop that has an if statement to see if each limit switch has been pressed, if one did then the location of the arm/robot is set to the location of the limit switch that has been pressed.
9. Create an abstract class called MoveWithLimitSwitches that extends from Command. This will act as a template for all commands that use the limit switch subsystems for movement.
10. In the class create the following variables:
    a. An integer variable named destination that will store the desired destination of the arm/robot.
    b. A boolean variable called reachedDestination that determines if the location of the arm/robot matches the desired destination.
    c. A double variable called speed that just stores the speed that the robot/arm will move at.
11. In the execute method, a method from the Command class, use if statements to set the speed positive if the destination is higher than the current location, and negative when the destination is lower than the current location. Another if statement for when the two are equal which will just end the command.
12. Create two subsystems called FrontLift and RearLift that extends from the LSLevelSubsystem then create instances of them in TorqueLift. This give the TorqueLift subsystem contain all the capabilities of the FrontLift and RearLift since it is made of them.
13. In the Arm class, have it extend from the LSLevelSubsystem to share its capabilities.
14. Create three Command Classes called moveArm, moveFront and moveRear that extend from the MoveWithLimitSwitches class. Set the subsystem to the name of the subsystem of that the command uses.

Discussion:

At first, the major issue was that the LSLevelSubsystem variable in MoveWithLimitSwitches was used for requires(), a method to declare subsystem dependencies. This created caused the driver station to read "No Robot Code" despite code being deployed. To check that the issue was caused with the code itself, another already test code was deployed and worked fine so that problem was linked to the new code that was put in. In the console log, a message read that in the moveArm, moveFront, and moveRear the subsystem dependencies read null. This meant that the subsystem placed in requires which was an abstract class LSLevelSystem which made sense for it to read null since it was abstract. The issue was resolved by removing the requires method and instead was moved to the move Commands so that they would reference their specific subsystem.

The second issue was how the safety feature worked. First, the speed was turned off when it hit the limit switches but that just caused the arm to get stuck. Instead the speed was set to move away from the edge until the limit switch is no longer pressed. During a couple of the tests for the arm safety feature, the wire kept getting tangled and undone due it failing and going past the limit switch. The wire get tangled into a shaft instead of its set location. The wire would be too tight to be moved back to its original position so that arm had to be lifted to cause it to be more loose. After the wire became loose it was wrapped around its original position and the arm was lowered.

During Tuesday, there were driving tryout and after one driver (Mr. Norton) rammed into a wall, the loose nut for the torque lift went undone. The nut was attempted to be placed back however the wire connected to the left side was too short while the one on the right side was too loose. The issue was resolved by turning the torque lift gears so that each side would be balanced. After that a ½ wrench was used to keep the nut screwed in tightly to prevent potential issues in the future.

When code was uploaded into the robot, the drivetrain would oscillate. This means that two commands are fighting over the same motors. The subsystems were checked and the drivetrain and torque lift subsystem referenced the same motors. Usually the requires() method would allow a subsystem to be used for only one task at a time but since there were two subsystems with the same components the drivetrain motors were able to be called on by multiple commands. The drivetrain and torque lift subsystems were combined into a subsystem called RobotBase so that the drivetrain motors wouldn't be called in two different subsystems.

Conclusion:

An abstract class can be used as a template for subsystems and commands. The LSLevelSubsystem abilities to check the location of the arm and declare the limit switches based on simply the port numbers. By extending classes that use limit switches from the LSLevelSubsystem, that class will already have all the methods for tracking location and a safety feature for movement. This safety feature moves the arm/robot away from the edges and since it

is in the LSLevelSubsystem class, it doesn't have to be repeated in every class that uses limit switches. The MoveWithLimitSwitches class is a command template that allows moves the arm/robot to a certain limit switch. The moveArm, moveFront, and moveRear extend from this class so that the only thing that has to be declared in those classes is the subsystem that is used.