Team 4611

Code Standards

A Broad Guideline to Ozone's Code

"Commit Early, Commit Often"

Robot

We use what is called an iterative robot. It follows the structure as follows...

robotInit() -- provide for initialization at robot power-on

init() functions -- each of the following functions is called once when the appropriate mode is entered:

- DisabledInit() -- called only when first disabled
- AutonomousInit() -- called each and every time autonomous is entered from another mode
- TeleopInit() -- called each and every time teleop is entered from another mode
- TestInit() -- called each and every time test mode is entered from another mode

Periodic() functions -- each of these functions is called iteratively at the appropriate periodic rate (aka the "slow loop"). The period of the iterative robot is synced to the driver station control packets, giving a periodic frequency of about 50Hz (50 times per second).

- disabledPeriodic()
- autonomousPeriodic()
- teleopPeriodic()
- testPeriodoc()

OI

OI is where we set up our player input for the robot. This can include Xbox controllers, joysticks, button boxes, and more.

It is also were commands are bound to buttons.

Command

A command is something you actually want the robot to do. Something like "SpinUpShooter" or "CloseGearGadget. " The command flow goes as follows. (Figure 1.)

Commands can be put into different types of groups including parallelCommandGroups and sequentialCommandGroups. These groups can contain any number of commands. In a parallelCommandGroup, all of the commands will run at once, in a sequential one, they run one after another. There are other types of commands too.

A command group can be treated just like a normal command, it can be scheduled, and even nested into other command groups.

Subsystem

A subsystem is an actual mechanism. This is where you reference the objects you want to use and tell them what to do with your own methods. Some Subsystems that we've had in the past include, "Drivetrain", "BallDelivery", and "intake".

Git

GitHub is like the cloud, for sharing code. It's Google Docs, but for code. GitHub allows multiple people to collaborate on one project. When we want to work on a new feature for the robot, we pull out a branch of the code to work on. This means that we are working on separate code than everybody else. This helps prevent us from breaking other people's code. Once the feature is complete and tested, it can get merged back together with the rest of the code.

Working Tree/Workspace: On your local computer stored on your text editor
Staging Area: A temporary storage place for your changes
Local Repo: More permanent storage place for your local changes. These are things that you know work and are committed to keeping.
Remote Repo: This is where you place your files you are absolutely committed to and want to be available to others. These are not local changes.
Add: Moves files from workspace to staging area
Commit: Moves files from staging to local repo
Commit -am: Moves files from workspace straight to local repo. Adds a message to this commit. Should be descriptive.
Push: Moves files from local repo to remote repo
Pull: Pulls files from remote repo to local repo
Master: The cleanest branch of files. All code in here must work and is the main source for code
Branch: You "branch" from master creating a second remote repo that lies next to master but pushing to this branch will not affect master
Merge: When you move your changes from one branch to another branch (or master)

A quick how to with getting code into Eclipse:

1) To import programs: file, import
2) Clone URI
3) Paste repository URL (found on github)
4) Import

5) **Put program file, for example Stronghold2016 into workspace for easy access

https://education.github.com/git-cheat-sheet-education.pdf

Figure 1. Command Flow

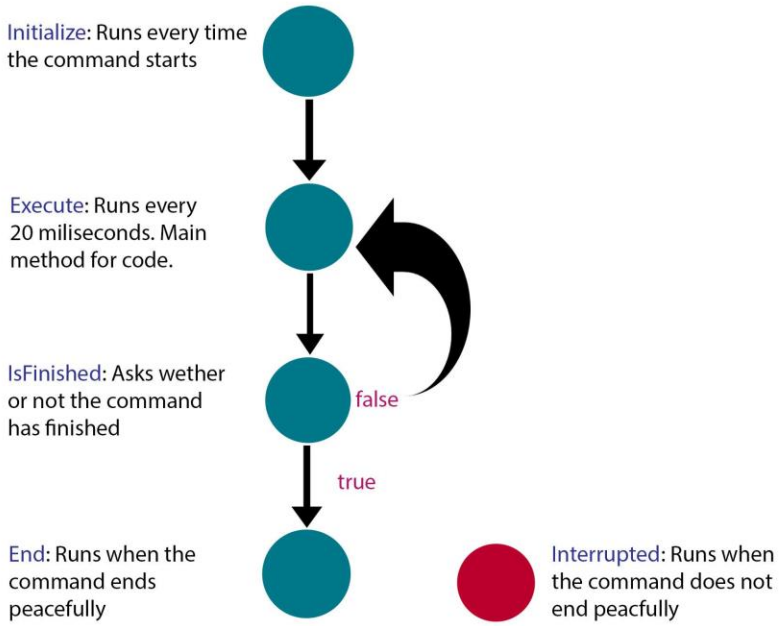

Figure 2. Example Command

```
7  public class TankDrive extends Command{
8
9      public TankDrive(){
10         this.requires(Robot.tankDrive); //This command uses this subsystem
11     }
12
13     protected void execute() { //execute is called every 20 miliseconds
14         double rightJoyVal = Robot.oi.filter(Robot.oi.rightJoy.getY()); //Grab the Y value of the joystick and pass
15         double leftJoyVal = Robot.oi.filter(Robot.oi.leftJoy.getY());; //it through the filter
16         Robot.tankDrive.move(leftJoyVal, rightJoyVal); //Then pass that double to the method "move" in tankDrive
17       }
18
19     @Override
20     protected boolean isFinished() {
21         return false; //Don't stop running this command
22     }
23
24 }
25
```

Says this class is going to be a command

Asks what subsystem will be running this command

Where you place your command methods

Figure 3. Example Subsystem

DriveTrain.java

```
1  package org.usfirst.frc.team4611.robot.subsystems;
2
3  import org.usfirst.frc.team4611.robot.RobotMap;
7
8  public class DriveTrain extends Subsystem {
9
10     public void move(double left, double right) { //Grabs the left and right values that get passed by "TankDrive"
11         RobotMap.driveTrain.tankDrive(-left, -right); //Use those values for the method "tankDrive" which calls for joystick values
12     }
13
14     @Override
15     protected void initDefaultCommand() {
16         setDefaultCommand(new TankDrive()); //This subsystem will automatically run this command
17     }
18 }
19
```

Says this class is going to be a subsystem

Create your own methods you want the subsystem to run

If you want the subsystem to "boot up" with a command already running do that here

Figure 4. Git Commands and Flow