/\*----------------------------------------------------------------------------\*/

/\* Copyright (c) 2017-2018 FIRST. All Rights Reserved. \*/

/\* Open Source Software - may be modified and shared by FRC teams. The code \*/

/\* must be accompanied by the FIRST BSD license file in the root directory of \*/

/\* the project. \*/

/\*----------------------------------------------------------------------------\*/

**package** org.usfirst.frc.team4499.robot;

**import** edu.wpi.first.wpilibj.DoubleSolenoid;

**import** edu.wpi.first.wpilibj.TimedRobot;

**import** edu.wpi.first.wpilibj.command.Command;

**import** edu.wpi.first.wpilibj.command.Scheduler;

**import** edu.wpi.first.wpilibj.smartdashboard.SendableChooser;

**import** edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;

**import** org.usfirst.frc.team4499.robot.commands.DriveTrain;

**import** org.usfirst.frc.team4499.robot.commands.ExampleCommand;

**import** org.usfirst.frc.team4499.robot.commands.Fire;

**import** org.usfirst.frc.team4499.robot.commands.Set\_Piston;

**import** org.usfirst.frc.team4499.robot.subsystems.ExampleSubsystem;

/\*\*

\* The VM is configured to automatically run this class, and to call the

\* functions corresponding to each mode, as described in the TimedRobot

\* documentation. If you change the name of this class or the package after

\* creating this project, you must also update the build.properties file in the

\* project.

\*/

**public** **class** Robot **extends** TimedRobot {

**public** **static** ExampleSubsystem *m\_subsystem* = **new** ExampleSubsystem();

**public** **static** OI *m\_oi*;

**private** RobotConfig config;

**private** DriveTrain drive;

**private** Fire fire;

Command m\_autonomousCommand;

SendableChooser<Command> m\_chooser = **new** SendableChooser<>();

/\*\*

\* This function is run when the robot is first started up and should be

\* used for any initialization code.

\*/

@Override

**public** **void** robotInit() {

*m\_oi* = **new** OI();

m\_chooser.addDefault("Default Auto", **new** ExampleCommand());

config = **new** RobotConfig();

config.setup();

// chooser.addObject("My Auto", new MyAutoCommand());

SmartDashboard.*putData*("Auto mode", m\_chooser);

}

/\*\*

\* This function is called once each time the robot enters Disabled mode.

\* You can use it to reset any subsystem information you want to clear when

\* the robot is disabled.

\*/

@Override

**public** **void** disabledInit() {

}

@Override

**public** **void** disabledPeriodic() {

Scheduler.*getInstance*().run();

}

/\*\*

\* This autonomous (along with the chooser code above) shows how to select

\* between different autonomous modes using the dashboard. The sendable

\* chooser code works with the Java SmartDashboard. If you prefer the

\* LabVIEW Dashboard, remove all of the chooser code and uncomment the

\* getString code to get the auto name from the text box below the Gyro

\*

\* <p>You can add additional auto modes by adding additional commands to the

\* chooser code above (like the commented example) or additional comparisons

\* to the switch structure below with additional strings & commands.

\*/

@Override

**public** **void** autonomousInit() {

m\_autonomousCommand = m\_chooser.getSelected();

/\*

\* String autoSelected = SmartDashboard.getString("Auto Selector",

\* "Default"); switch(autoSelected) { case "My Auto": autonomousCommand

\* = new MyAutoCommand(); break; case "Default Auto": default:

\* autonomousCommand = new ExampleCommand(); break; }

\*/

// schedule the autonomous command (example)

**if** (m\_autonomousCommand != **null**) {

m\_autonomousCommand.start();

}

}

/\*\*

\* This function is called periodically during autonomous.

\*/

@Override

**public** **void** autonomousPeriodic() {

Scheduler.*getInstance*().run();

}

@Override

**public** **void** teleopInit() {

Set\_Piston piston1 = **new** Set\_Piston(RobotMap.*catapult*,RobotMap.*catapultResting*);

piston1.start();

Set\_Piston piston2 = **new** Set\_Piston(RobotMap.*catapultRelease*, RobotMap.*releaseClosed*);

piston2.start();

fire = **new** Fire();

drive = **new** DriveTrain();

drive.start();

// This makes sure that the autonomous stops running when

// teleop starts running. If you want the autonomous to

// continue until interrupted by another command, remove

// this line or comment it out.

**if** (m\_autonomousCommand != **null**) {

m\_autonomousCommand.cancel();

}

}

/\*\*

\* This function is called periodically during operator control.

\*/

@Override

**public** **void** teleopPeriodic() {

**if** (OI.*controllerZero*.getRawAxis(2)>0.5) {

RobotMap.*intake*.set(DoubleSolenoid.Value.***kReverse***); //right = 3 left = 2

}

**if** (OI.*leftshiftButton*.get()) {

RobotMap.*shifters*.set(DoubleSolenoid.Value.***kForward***);

}

**else** **if** (OI.*rightshiftButton*.get()) {

RobotMap.*shifters*.set(DoubleSolenoid.Value.***kReverse***);

}

**if**(!(fire.isRunning())){

**if** (OI.*controllerZero*.getRawAxis(3)> 0.5)

{

Set\_Piston move = **new** Set\_Piston(RobotMap.*intake*, RobotMap.*intakeOut*);

move.start();

}

**else** **if** (OI.*controllerZero*.getRawAxis(3)<0.5)

{

Set\_Piston move = **new** Set\_Piston(RobotMap.*intake*, RobotMap.*intakeIn*);

move.start();

}

}

**if**(OI.*fireButton*.get()) {

fire.start();

}

**if** RobotMap.intakeOut {

RobotMap.intake.set(DoubleSolenoid.Value.kForward);

}

}

Scheduler.getInstance().run();

}

/\*\*

\* This function is called periodically during test mode.

\*/

@Override

**public** **void** testPeriodic() {

}

}