*/\* Copyright (c) 2014 Qualcomm Technologies Inc  
  
All rights reserved.  
  
Redistribution and use in source and binary forms, with or without modification,  
are permitted (subject to the limitations in the disclaimer below) provided that  
the following conditions are met:  
  
Redistributions of source code must retain the above copyright notice, this list  
of conditions and the following disclaimer.  
  
Redistributions in binary form must reproduce the above copyright notice, this  
list of conditions and the following disclaimer in the documentation and/or  
other materials provided with the distribution.  
  
Neither the name of Qualcomm Technologies Inc nor the names of its contributors  
may be used to endorse or promote products derived from this software without  
specific prior written permission.  
  
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED BY THIS  
LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,  
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE  
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL  
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR  
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER  
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,  
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. \*/***package** com.qualcomm.ftcrobotcontroller.opmodes;  
  
**import** com.qualcomm.ftccommon.DbgLog;  
**import** com.qualcomm.robotcore.eventloop.opmode.OpMode;  
**import** com.qualcomm.robotcore.hardware.DcMotor;  
**import** com.qualcomm.robotcore.hardware.Servo;  
**import** com.qualcomm.robotcore.util.Range;  
  
*/\*\*  
 \* TeleOp Mode  
 \* <p>  
 \* Enables control of the robot via the gamepad  
 \*/***public class** TeleOp9266 **extends** OpMode {  
  
 */\*  
 \* Note: the configuration of the servos is such that  
 \* as the arm servo approaches 0, the arm position moves up (away from the floor).  
 \* Also, as the claw servo approaches 0, the claw opens up (drops the game element).  
 \*/  
 // TETRIX VALUES.* **final static double *ARM\_MIN\_RANGE*** = 0.20;  
 **final static double *ARM\_MAX\_RANGE*** = 0.90;  
 **final static double *CLAW\_MIN\_RANGE*** = 0.20;  
 **final static double *CLAW\_MAX\_RANGE*** = 0.7;  
  
 *// position of the arm servo.* **double armPosition**;  
  
 *// amount to change the arm servo position.* **double armDelta** = 0.6;  
  
 *// position of the claw servo* **double clawPosition**;  
  
 *// amount to change the claw servo position by* **double clawDelta** = 0.4;  
  
 DcMotor **motorRight**;  
 DcMotor **motorLeft**;  
 Servo **claw**;  
 Servo **arm**;  
 Servo **door**;  
 DcMotor **conveyorbelt**;  
 DcMotor **lift**;  
  
  
 */\*\*  
 \* Constructor  
 \*/* **public** TeleOp9266() {  
  
 }  
  
 */\*  
 \* Code to run when the op mode is first enabled goes here  
 \*   
 \* @see com.qualcomm.robotcore.eventloop.opmode.OpMode#start()  
 \*/* @Override  
 **public void** start() {  
  
 */\*DbgLog.msg(Double.toString(scaleInput(0.2)));  
 DbgLog.error(Double.toString(scaleInput(0.2)));\*/  
  
 /\*  
 \* Use the hardwareMap to get the dc motors and servos by name. Note  
 \* that the names of the devices must match the names used when you  
 \* configured your robot and created the configuration file.  
 \*/  
   
 /\*  
 \* For the demo Tetrix K9 bot we assume the following,  
 \* There are two motors "motor\_1" and "motor\_2"  
 \* "motor\_1" is on the right side of the bot.  
 \* "motor\_2" is on the left side of the bot and reversed.  
 \*   
 \* We also assume that there are two servos "servo\_1" and "servo\_6"  
 \* "servo\_1" controls the arm joint of the manipulator.  
 \* "servo\_6" controls the claw joint of the manipulator.  
 \*/* **motorRight** = **hardwareMap**.**dcMotor**.get(**"motor\_1"**);  
 **motorLeft** = **hardwareMap**.**dcMotor**.get(**"motor\_2"**);  
 **conveyorbelt** = **hardwareMap**.**dcMotor**.get(**"motor\_conveyorbelt"**);  
 **lift** = **hardwareMap**.**dcMotor**.get(**"motor\_lift"**);  
  
 **motorLeft**.setDirection(DcMotor.Direction.***REVERSE***);  
 **conveyorbelt**.setDirection(DcMotor.Direction.***REVERSE***);  
  
  
   
 *//arm = hardwareMap.servo.get("servo\_1");  
 //claw = hardwareMap.servo.get("servo\_6");  
 //door = hardwareMap.servo.get("servo\_3");  
  
 // assign the starting position of the wrist and claw  
 //armPosition = 0.2;  
 //clawPosition = 0.2;* }  
  
 */\*  
 \* This method will be called repeatedly in a loop  
 \*   
 \* @see com.qualcomm.robotcore.eventloop.opmode.OpMode#run()  
 \*/* @Override  
 **public void** loop() {  
  
 */\*  
 \* Gamepad 1  
 \*   
 \* Gamepad 1 controls the motors via the left stick, and it controls the  
 \* wrist/claw via the a,b, x, y buttons  
 \*/  
  
 // throttle: left\_stick\_y ranges from -1 to 1, where -1 is full up, and  
 // 1 is full down  
 // direction: left\_stick\_x ranges from -1 to 1, where -1 is full left  
 // and 1 is full right* **float** right = -**gamepad1**.**right\_stick\_y**;  
 **float** left = **gamepad1**.**left\_stick\_y**;  
  
  
  
  
  
  
 *// clip the right/left values so that the values never exceed +/- 1* right = Range.*clip*(right, -1, 1);  
 left = Range.*clip*(left, -1, 1);  
  
 *// scale the joystick value to make it easier to control  
 // the robot more precisely at slower speeds.* right = (**float**)scaleInput(right);  
 left = (**float**)scaleInput(left);  
  
 *// write the values to the motors* **motorRight**.setPower(right);  
  
 **motorLeft**.setPower(left);  
  
 **if**(**gamepad1**.**a** )  
 {  
 **conveyorbelt**.setPower(-1);  
 }  
  
 **if**(**gamepad1**.**y**)  
 {  
 **conveyorbelt**.setPower(1);  
  
  
 }  
  
 **if**(**gamepad1**.**x**)  
 {  
 **conveyorbelt**.setPower(0);  
  
 }  
  
 **if**(**gamepad1**.**b**)  
 {  
 **lift**.setPower(1);  
 }  
  
 **if**(**gamepad1**.**right\_bumper**)  
 {  
 **lift**.setPower(-1);  
 }  
 **if**(**gamepad1**.**left\_bumper**)  
 {  
 **lift**.setPower(0);  
 }  
  
  
 *// clip the position values so that they never exceed their allowed range.  
 // armPosition = Range.clip(armPosition, ARM\_MIN\_RANGE, ARM\_MAX\_RANGE);  
 // clawPosition = Range.clip(clawPosition, CLAW\_MIN\_RANGE, CLAW\_MAX\_RANGE);  
  
 // write position values to the wrist and claw servo  
 // arm.setPosition(armPosition);  
 // claw.setPosition(clawPosition);  
  
  
  
 /\*  
 \* Send telemetry data back to driver station. Note that if we are using  
 \* a legacy NXT-compatible motor controller, then the getPower() method  
 \* will return a null value. The legacy NXT-compatible motor controllers  
 \* are currently write only.  
 \*/* **telemetry**.addData(**"Text"**, **"\*\*\* Robot Data\*\*\*"**);  
 **telemetry**.addData(**"motorright"**, **"motor right:"** + String.*format*(**"%.2f"**, right));  
 **telemetry**.addData(**"motorleft"**, **"motor left:"** + String.*format*(**"%.2f"**, left));  
  
  
 */\*telemetry.addData("arm", "arm: " + String.format("%.2f", armPosition));  
 telemetry.addData("claw", "claw: " + String.format("%.2f", clawPosition));  
 telemetry.addData("left tgt pwr", "left pwr: " + String.format("%.2f", left));  
 telemetry.addData("right tgt pwr", "right pwr: " + String.format("%.2f", right));  
 \*/* }  
  
 */\*  
 \* Code to run when the op mode is first disabled goes here  
 \*   
 \* @see com.qualcomm.robotcore.eventloop.opmode.OpMode#stop()  
 \*/* @Override  
 **public void** stop() {  
  
 }  
   
 */\*  
 \* This method scales the joystick input so for low joystick values, the   
 \* scaled value is less than linear. This is to make it easier to drive  
 \* the robot more precisely at slower speeds.  
 \*/* **double** scaleInput(**double** dVal) {  
 **double**[] scaleArray = { 0.0, 0.05, 0.09, 0.10, 0.12, 0.15, 0.18, 0.24,  
 0.30, 0.36, 0.43, 0.50, 0.60, 0.72, 0.85, 1.00, 1.00 };  
   
 *// get the corresponding index for the scaleInput array.* **int** index = (**int**) (dVal \* 16.0);  
 **if** (index < 0) {  
 index = -index;  
 } **else if** (index > 16) {  
 index = 16;  
 }  
   
 **double** dScale = 0.0;  
 **if** (dVal < 0) {  
 dScale = -scaleArray[index];  
 } **else** {  
 dScale = scaleArray[index];  
 }  
   
 **return** dScale;  
 }  
  
}