```
/*
Copyright (c) 2016 Robert Atkinson

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 Robert Atkinson nor the names of his 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 FITNESSFOR 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 org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.util.ElapsedTime;

/**
 * This file illustrates the concept of driving a path based on encoder counts.
 * It uses the Babybot hardware class to define the drive on the robot.
 * The code is structured as a LinearOpMode
 *
 *  This code ALSO requires that the drive Motors have been configured such that a positive
 *  power command moves them forwards, and causes the encoders to count UP.
 *
 *   The desired path in this example is:
 *   - Drive forward for 48 inches
 *   - Spin right for 12 Inches
 *   - Drive Backwards for 24 inches
 *   - During all driving Sense for color and move claw or arm by color type
 *
 *  The code is written using a method called: encoderDrive(speed, leftInches, rightInches, timeoutS)
 *  that performs the actual movement.
 *  This methods assumes that each movement is relative to the last stopping place.
 *  There are other ways to perform encoder based moves, but this method is probably the simplest.
 *  This code uses the RUN_TO_POSITION mode to enable the Motor controllers to generate the run profile
 *
 * Use Android Studios to Copy this Class, and Paste it into your team's code folder with a new name.
 * Remove or comment out the @Disabled line to add this opmode to the Driver Station OpMode list
 */

@Autonomous(name="Babybot: EncoderBase", group="Abdul")
//@Disabled
public class BabybotAutoDriveByEncoder_Base extends LinearOpMode {

    /* Declare OpMode members. */
    HardwareBabybot         robot   = new HardwareBabybot();   // Use a Babybot's hardware
    private ElapsedTime     runtime = new ElapsedTime();

    static final double     COUNTS_PER_MOTOR_REV    = 1120 ;    // eg: AndyMark Motor Encoder
    static final double     DRIVE_GEAR_REDUCTION    = 1.0 ;     // This is < 1.0 if geared UP
    static final double     WHEEL_DIAMETER_INCHES   = 3.0 ;     // For figuring circumference
    static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV * DRIVE_GEAR_REDUCTION) /
                                                      (WHEEL_DIAMETER_INCHES * 3.1415);
    static final double     DRIVE_SPEED             = 0.6;
```

```
static final double     TURN_SPEED              = 0.5;


@Override
public void runOpMode() throws InterruptedException {

    /*
     * Initialize the drive system variables.
     * The init() method of the hardware class does all the work here
     */
    robot.init(hardwareMap);

    // Send telemetry message to signify robot waiting;
    telemetry.addData("Status", "Resetting Encoders");    //
    telemetry.update();

    robot.leftMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    robot.rightMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    idle();

    robot.leftMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    robot.rightMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

    // Send telemetry message to indicate successful Encoder reset
    telemetry.addData("Path0",  "Starting at %7d :%7d",
                      robot.leftMotor.getCurrentPosition(),
                      robot.rightMotor.getCurrentPosition());
    telemetry.update();


    // Wait for the game to start (driver presses PLAY)
    waitForStart();

    // Step through each leg of the path,
    // Note: Reverse movement is obtained by setting a negative distance (not speed)
    encoderDrive(DRIVE_SPEED,  48,  48);  // S1: Forward 48 Inches with 5 Sec timeout

    //////////////////////////
    robot.leftMotor.setPower(0);
    robot.rightMotor.setPower(0);
    Thread.sleep(200);
    //////////////////////////

    encoderDrive(TURN_SPEED,   12, -12);  // S2: Turn Right 12 Inches with 4 Sec timeout
    encoderDrive(DRIVE_SPEED, -24, -24);  // S3: Reverse 24 Inches with 4 Sec timeout

    ///////////////////////
    sleep(1000);     // pause for servos to move

    telemetry.addData("Path", "Complete");
    telemetry.update();
}

/*
 *  Method to perfmorm a relative move, based on encoder counts.
 *  Encoders are not reset as the move is based on the current position.
 *  Move will stop if any of three conditions occur:
 *  1) Move gets to the desired position
 *  2) Move runs out of time
 *  3) Driver stops the opmode running.
 */
public void encoderDrive(double speed,
                         double leftInches, double rightInches) throws InterruptedException {
    int newLeftTarget;
    int newRightTarget;

    // Ensure that the opmode is still active
    if (opModeIsActive()) {

        // Determine new target position, and pass to motor controller
        newLeftTarget = robot.leftMotor.getCurrentPosition() + (int)(leftInches * COUNTS_PER_INCH);
        newRightTarget = robot.rightMotor.getCurrentPosition() + (int)(rightInches * COUNTS_PER_INCH);
        robot.leftMotor.setTargetPosition(newLeftTarget);
        robot.rightMotor.setTargetPosition(newRightTarget);

        // Turn On RUN_TO_POSITION
        robot.leftMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        robot.rightMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
```

```
            //  start motion.
            robot.leftMotor.setPower(Math.abs(speed));
            robot.rightMotor.setPower(Math.abs(speed));

            // keep looping while we are still active, and there is time left, and both motors are running.
            while (opModeIsActive() && (robot.leftMotor.isBusy() && robot.rightMotor.isBusy())) {

                // Display it for the driver.
                telemetry.addData("Path1",  "Running to %7d :%7d", newLeftTarget,  newRightTarget);
                telemetry.addData("Path2",  "Running at %7d :%7d",
                                            robot.leftMotor.getCurrentPosition(),
                                            robot.rightMotor.getCurrentPosition());
                telemetry.update();

                // Allow time for other processes to run.
                idle();
            }

            // Stop all motion;
            robot.leftMotor.setPower(0);
            robot.rightMotor.setPower(0);

            // Turn off RUN_TO_POSITION
            robot.leftMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            robot.rightMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

            sleep(250);   // optional pause after each move
            idle();

        }
    }

}
```