```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.HardwareMap;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.util.ElapsedTime;

import com.qualcomm.robotcore.hardware.ColorSensor;

/**
 * This is NOT an opmode.
 *
 * This class can be used to define all the specific hardware for a single robot.
 * In this case that robot is a Babybot.
 *
 * This hardware class assumes the following device names have been configured on the robot:
 * Note:  All names are lower case and some have single spaces between words.
 *
 * Motor channel:  Left  drive motor:        "left_drive"  Motor 2
 * Motor channel:  Right drive motor:        "right_drive" Motor 1 With encoder Wire
 * Servo channel:  Servo to open  claw:      "claw"
 * Servo channel:  Servo to open  arm:       "arm"
 */
public class HardwareBabybot
{
    /* Public OpMode members. */
    public DcMotor  leftMotor   = null;
    public DcMotor  rightMotor  = null;

    public Servo    claw    = null;
    public Servo    arm   = null;

    public ColorSensor colorSensor  = null;    // Color Sensor Device Object

    public static final double STD_SERVO       =  0.1 ;
    public static final double FIN_SERVO       =  0.9 ;

    // bLedOn represents the state of the LED.
    public boolean bLedOn = false;

    /* local OpMode members. */
    HardwareMap hwMap           =  null;
    private ElapsedTime period  = new ElapsedTime();

    /* Constructor */
    public HardwareBabybot(){

    }

    /* Initialize standard Hardware interfaces */
    public void init(HardwareMap ahwMap) {
        // Save reference to Hardware map
        hwMap = ahwMap;

        // Define and Initialize Motors
        leftMotor   = hwMap.dcMotor.get("left_drive");
        rightMotor  = hwMap.dcMotor.get("right_drive");


        leftMotor.setDirection(DcMotor.Direction.FORWARD); // Set to FORWARD if using AndyMark motors
        rightMotor.setDirection(DcMotor.Direction.REVERSE);// Set to REVERSE if using AndyMark motors

        // Set all motors to zero power
        leftMotor.setPower(0);
        rightMotor.setPower(0);


        // Set all motors to run with encoders.
        leftMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rightMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        // get a reference to our ColorSensor object.
```

```
        colorSensor = hwMap.colorSensor.get("color sensor");

        // Set the LED in the beginning
        colorSensor.enableLed(bLedOn);


        // Define and initialize ALL installed servos.
        claw = hwMap.servo.get("claw");
        arm = hwMap.servo.get("arm");
        claw.setPosition(STD_SERVO);
        arm.setPosition(FIN_SERVO);
    }

    /***
     *
     * waitForTick implements a periodic delay. However, this acts like a metronome with a regular
     * periodic tick.  This is used to compensate for varying processing times for each cycle.
     * The function looks at the elapsed cycle time, and sleeps for the remaining time interval.
     *
     * @param periodMs  Length of wait cycle in mSec.
     * @throws InterruptedException
     */
    public void waitForTick(long periodMs) throws InterruptedException {

        long  remaining = periodMs - (long)period.milliseconds();

        // sleep for the remaining portion of the regular cycle period.
        if (remaining > 0)
            Thread.sleep(remaining);

        // Reset the cycle clock for the next pass.
        period.reset();
    }
}
```