**-----Glossary-----**

**Class** - a specific type of object that can contain other variable types and methods

**Declaration and Initialization-** A statement in which a variable is declared and immediately given its initial value

**Extends –** Means you are creating a subclass of the class you are extending

**Method** - a group of code that has the ability to take in, process, or return data

**Object -** an instance of a particular class. The class to which an object belongs defines the general characteristics of all instances of that class

**Opmode** - A user generated operational mode, or a program run by the robot

 **Normal** - an OpMode archetype that employs 4 main method during its run cycle: init, start, loop, and stop

 **Linear** - an OpMode archetype that employs only 1 main method, the runOpMode method, in its run cycle. Linear OpModes are used for autonomous programming because the looping nature of Normal OpModes is unneeded

 **Autonomous** - robot-controlled OpModes that are run during the first 30 seconds of a match

 **Teleoperated** - driver-controlled OpModes that are controlled during the last 2 minutes of a match

**Primitive Data Types -** Java's standard, non-class data types. They include:

 **int -** a positive or negative whole number

 **float -** a number with an integer and decimal component

 **double -** a number with an integer and decimal component

 **long -** a positive or negative whole number. Can represent larger numbers than an int

 **short -** a positive or negative whole number. Can represent smaller numbers than int, but consumes less space

 **byte -** a single byte of data

 **char -** a two-byte character from the ISO Unicode character set

 **boolean -** a single bit value. Represents on (1) or off (0)

**Run Cycle –** The order of operations of a program from its start to closing

**Throws –** A clause in a *method header* that indicates an exception will be *propagated* by that method

**Variable** - a piece of data associated with an object

**While Loop -** One of Java's three *control structures* used for looping. The other two are the *do loop* and *for loop*. A while loop contains both a condition and a loop body. As long as the condition holds true, the loop body will continue to execute.

**-----FTC Specific Code-----**

**ElapsedTime -** a user controlled timer that can count in units as small as milliseconds. After initializing a timer (via *ElapsedTime [nameOfTimerVariable] = new ElapsedTime();*)*,* the user can reset the timer with *[nameOfTimerVariable].reset();* and get the current time elapsed since the last called reset with *[nameOfTimerVariable].time();*

**DcMotor -** a variable that represents one DcMotor on the robot

**Servo -** a variable that represents any type of servo that is connected to the robot. This includes 180 degree, continuous rotation, and quarter-scale

**Motor and Servo Initialization -** motor and servo variables need to be given motors and servos on the physical robot to reference; otherwise, the variables will not represent actual motors or servos. To initialize a motor, use the line *[nameOfDcMotorVariable].hardwareMap.DcMotor.get("nameOfMotorInConfigFile");,* and to initialize a servo use the line *[nameOfServoVariable].hardwareMap.servo.get("nameOfServoInConfigFile");*

**Motor Movement -** a motor is set to run at a specific power via a call to *[motorVariable'sName].setPower(double power);* . The double *power* can be a value anywhere from -1.0 to 1.0 with -1.0 representing full power backwards, 0 representing motionless, and 1.0 representing full power forwards.

**Servo Movement -** a servo is moved to a specific position via a call to *[servoVariable'sName].setPosition(double position);* . Although the double *position* can only be a value from 0 to 1.0, those values represent the servo's position in degrees from 0 to 180. The decimal input value can be determined by dividing the desired position in degrees by 180

**WaitForStart -** a method included in the FTC API that tells the program to wait for the press of the start button to continue the program. Only useful in Linear OpModes, because Normal OpModes have separate methods for initialization, starting, and stopping. Called with *waitForStart();*

Some of the definitions on this list are borrowed from:

https://www.cs.kent.ac.uk/people/staff/djb/oop/glossary.html

Additional coding terminology can be found at this website.