Nomenclature:

Identifier type	Rules for naming	Examples
Major Classes (programs that will be downloaded to the robot and run)	Class names should be nouns in `UpperCamelCase`, with the first letter of every word capitalized. Use whole words — avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML). The tele-op program can simply be called “TeleOp” and the version number. If multiple programs are needed for different drive teams, then “TeleOp” can be followed by the driver name. (i. e. TeleOpSteve2.11.9) Autonomous program names must start with “Auto”, then “Red” or “Blue” (if the field requires mirrored or otherwise different programs for each side), and its purpose (where it’s scoring). If a delay is added, the number of seconds it waits should be added after the alliance color with an “s” after the number (see example). If a menu program is created, the autonomous program can simply be called “AutoMenu” and the version number.	• For an autonomous on the blue side of the field with a 10 second delay that will score in the center goal, version 1.6.3 would be named: AutoBlue10sCenter1.6.3
Minor Classes (classes that are included in Major Classes)	Class names should be nouns in `UpperCamelCase`, with the first letter of every word capitalized. Use whole words — avoid acronyms and abbreviations (unless the abbreviation is much more widely used	• `class DriveFunction;` • `class IRSensing;`

	than the long form, such as URL or HTML). Classes should be named such that their purpose is easily identifiable from the name.	
Methods	Methods should be verbs in `lowerCamelCase` or a multi-word name that begins with a verb in lowercase; that is, with the first letter lowercase and the first letters of subsequent words in uppercase.	• `brake();` • `calibrateGyro();` • `getColor();`
Variables	Local variables, instance variables, and class variables are also written in `lowerCamelCase`. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic— that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.	• `int i;` • `char c;` • `float myWidth;`
Constants	Constants should be written in uppercase characters separated by underscores. Constant names may also contain digits if appropriate, but not as the first character.	• `static final int MAX_DISTANCE = 10;`
Motors	Motor names should be nouns in lowerCamelCase. Drive motors should start with “r” or “l” to denote right or left sides of the robot, then “Drive” to indicate it is a drive motor, and (if there is more than one drive motor per side) a number (1, 2, etc.). Other motors should include their function (Lift, collection, etc.), “Motor”, and a number if there are multiple motors allocated to that	• `Dcmotor rDrive1` • `Dcmotor collectionMotor` • `Dcmotor liftMotor6`

	function.	
Servos	Servo names should be nouns in lowerCamelCase. Names should start with their function, “Servo”, and a number if there are multiple servos allocated to that function. If two servos are for the same function but need to be further identified (i. e. they need to be programmed separately), a letter indication as to its location can be added at the beginning of the name(r=right, l=left, u=up, d=down, f=front, b=back).	• `Servo iceCreamServo` • `Servo rClawServo` • `Servo bClimberServo`
Sensors	Sensor names should be nouns in lowerCamelCase. Names should start with the sensor type, “Sensor” and then a number if there multiple sensors of that type.	• `GyroSensor gyroSensor` • `ColorSensor colorSensor3`

Versioning:

Version	Rules for versioning	Examples
Overview	We will be using semantic versioning 2.0.0. Given a version number MAJOR.MINOR.PATCH, increment the: 1. MAJOR version when you make incompatible API changes (revamp an autonomous program), 2. MINOR version when you add functionality in a backwards-compatible manner, and (make	1. If I add multiple things into the tele-op and change some servo values that would all be one MINOR version. If I only changed some drive distances, then I only need to move up one PATCH version when I push it to GitHub. 2. If I am working on autonomous 1.8.6, and add more functionality, it would bump the MINOR version up by 1, and reset the PATCH version to 0, meaning I will be pushing version 1.9.0 to GitHub.

	the autonomous do more stuff) 3. PATCH version when you make backwards-compatible bug fixes (change values for drive distance, sensor reading, etc.) You should not jump multiple versions if you make multiple edits in one session (see example 1). Once you release a new version, lower version types are reset to 0 (see example 2).	
Pre-release version	Before a program is fully functional, it is in a “pre-release” stage. In this case the MAJOR version will be 0, and the MINOR and PATCH versions will change when you add and edit material.	Before the Tele-op is fully functional it will be in version 0.x.y. Every time you add some functionality (a new motor is programmed in) you go up to a new MINOR version. Every time you change some values (a servo position) you can release a PATCH version.
Release Versions	Once a program is fully functional (it accomplishes everything it is supposed to for the next competition) it will be in MAJOR version 1.x.y. If a MAJOR version is completed but after the competition you want to add more points in autonomous or majorly revamp the tele-op, you will need to go up to a new MAJOR version once all new changes are completed. If you are only going to be tweaking, you can continue with MINOR and PATCH versions.	If I finish an autonomous program for the first competition and then make some tweaks, it could be up to version 1.2.3. Then if after the competition we decide to add more to the autonomous, we would continue upgrading the MINOR and PATCH versions until all the new parts are fully functional. At that point it would be at version 2.0.0, and would continue being improved from there.