**Computer Program**

A “computer program” is a sequence of instructions to tell robot what to do:

* **written** by humans in a **programming language**, using an **Editor**
* **translated** into machine language by the **Compiler**
* **uploaded** to robot's processor by an **Uploader**
* **executed** step by step by **Robot’s Computer**

**Java Ground Rules**

Java is one of many computer programming languages. As with every language, you must learn its **grammar** and **vocabulary** to use it effectively. Here are a few ground rules.

1. Java is case-sensitive, i.e. the word “Sound” and “sound” are considered to be different
2. Java is written as a sequence of statements, one per line
3. Java uses three sets of parenthesis/brackets:
	1. curly brackets “{ … }” to group statements together
	2. regular parenthesis “(…)” to denote inputs to functions and for math expressions
	3. square brackets “[ … ]” to denote what’s called arrays
4. A valid Java statement must end with a semi-colon ; unless it starts a group
5. Java names can not contain any spaces or strange symbols, and cannot start with numbers
6. The computer program executes your code (usually) one line after the other in sequence. It will do no more or no less than what your sequence of instructions specifies. You can ‘wish’ that a program will perform an action in a particular way, you must explicitly specify in writing every details of what is supposed to happen.
7. Java uses the term “class” instead of program, “method” instead of function, and “field” instead of state variables.
8. Java uses two types of comments you can use to explain what your program is doing. Comments can be written in any language and are ignored by the computer. Single-line comments start with // (a double-slash), multi-line comments are enclosed in /\* … \*/
9. Just because your program is spelled correctly (grammar and vocabulary) does not mean it will also work correctly – all programs indeed require extensive testing
10. Every (almost) program (i.e. class) has a unique name and looks like this:

**public class ProgramName**

**{**

 **// Here is where static “state” variables go**

 **public static void main(String[] args)**

 **{**

 **/\***

 **This is where the actual program goes**

 **\*/**

 **}**

**}**

**Programming fixed NXT Components**

Any robot built with the NXT brick includes many named components such as **LCD**, **Sound**, **Button**, **Motor**, etc. Some have fixed properties; others must be initialized with various parameters. Programming the fixed components is easy: use them by name and call on their built-in functions using a dot notation. Here is an overview.

**The LCD screen**

To use the LCD screen, you can use commands such as:

* LCD.clear()
* LCD.refresh()
* LCD.drawString("text", col, row), where row is between 0 and 7 and col between 0 and 15

**Delay or Timed Pause**

Many times you need a delay, or a timed break, in your program. For example, you might want to display some text for 2 seconds, then some additional text. Or you might want to engage the motor(s) for 3 seconds, then stop them. To do that you use the command

* Delay.msDelay(time) , where time is the desired delay in milliseconds

Remember that 1000 milliseconds are 1 second.

**The Buttons**

The brick contains four buttons: LEFT, RIGHT, ENTER and ESCAPE. Use the appropriate button name together with the command isDown() or waitForPressAndRelease(), as in:

* Button.LEFT.isDown() or Button.LEFT.waitForPressAndRelease()
* Button.RIGHT.isDown() or Button.RIGHT.waitForPressAndRelease()
* Button.ENTER.isDown() or Button.ENTER.waitForPressAndRelease()
* Button.ESCAPE.isDown() or Button.ESCAPE.waitForPressAndRelease()

**Sound**

You can program the robot to play some music, using the Sound component:

* Sound.playTone(freq, duration in ms)

where freq is the frequency in hertz and duration is the length of the note in milliseconds. Other static methods supported by the Sound component are:

* Sound.beep()
* Sound.beepSequence()
* Sound.beepSequenceUp()
* Sound.buzz();
* Sound.pause(millisecs)
* Sound.playTone(freq, duration in ms)

Note that the method Sound.playTune(440, 500) plays an A for 0.5 seconds, but does not wait until the note has finished playing. If you do want to make sure you can listen to the entire note, use the combination of

Sound.playTune(440, 500);
Delay.msDelay(500);

which causes the program to wait 0.5 seconds as the A plays for 0.5 seconds.