

Coding in Java vs. RIS

To some extent programming is programming, no matter which language you use. Java is a text-based, object-oriented language while RIS is a visual, procedural language, which makes the two very different. But they do have many similarities. This section displays RIS code, which you have learned in previous chapters, and Java code. Some of the Java code will be complete programs while other code will be code fragments. You can cut and paste the Java code into your editor, compile it and download it to your robot.



At left is a complete RIS program. At right is the equivalent Java program. Both programs can be downloaded to the RCX and both do nothing.

```
class DoNothing {
    static void main(String[] args) {
    }
}
```



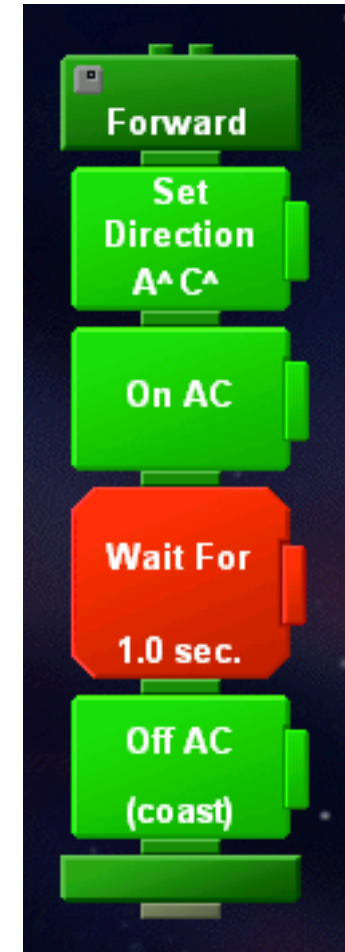
At left is a complete RIS program. Its name is Forward and it has two statements, a small block that sets the direction of the motors and a second block that turns on the A & C motors. The Java seems more complex. It starts with an import statement that makes RCX objects available to our program. The class statement defines a Forward object, whose “objectness” we ignore in this simple program. We define one method, main, which is our entry point for our program. main has a parameter, an array of Strings, which we also do not use.

```
import josx.platform.rcx.*;

class Forward {

    static void main(String[] args) {
        Motor.A.forward();
        Motor.C.forward();
    }
}
```

Forward, the main program, at right, has one statement, a big block Forward. Forward starts both motors in the forward direction for one second. Big blocks are subroutines, the equivalent to a Java method. The Forward big block is expanded at far right. The algorithm for Forward has four steps: set the motors' direction, turn the motors on, wait for one second and then turn the motors off. The Java code, below, does the same thing. The entry point is `main` and there is a single statement, `goForward(1000)`. Java times are in milliseconds so 1000 milliseconds is one second. The method `goForward` has the same statements as the Forward big block. The `Motor.A.forward` method both sets the direction and turns the motor on. `Thread.sleep` suspends execution for the specified time just as the Wait For block does in the RIS code. Finally, the `Motor.A.flt` statement turns off the motors. Again, `goForward` is static; it can be used without an associated object.



```
import josx.platform.rcx.*;

class Forward {

    static void goForward(int time) {
        Motor.A.forward();
        Motor.C.forward();
        try {
            Thread.sleep(time);
        } catch (InterruptedException e) {}
        Motor.A.flt();
        Motor.C.flt();
    }

    static void main(String[] args) {
        goForward(1000);
    }
}
```

This is a full program that you can cut and paste into your editor.

The next section compares RIS code fragments to their equivalent Java code fragments. You can experiment with these fragments by creating the called method, e.g. `turnLeft`, in the `Forward` class file and replacing the `goForward(1000)` method call with the respective code fragment.

These fragments are flow-control statements. If-then, conditionals, allow the execution of a program to proceed along different paths. Loops are the other flow-control statements and

