Code basics

INTRODUCTION

Starter Question: How do you make a peanut butter sandwich?

You might say something like:

- Spread peanut butter on bread
- Spread jelly on bread
- Put slices of bread together

But if you were trying to get a robot to do that you left out some important information. What does ‘spread’ mean? Spread with what? Where did the bread come from? What about the peanut butter and jelly? When you are giving a computer or robot instructions you will likely need to break things down

into smaller steps than you are used to. If you think of things that might seem basic, like shooting a ball or even driving forward, those are made up of much more complicated steps like turning on different motors to specific speeds and monitoring them to make sure that they get to the right position and then turning them off.

Computers don't speak English. Siri and Alexa might understand you when you speak, but in order to get a computer to do what you want you need to write code in a language it understands. We are going to use Java. When you write that code, and go to run it on a computer (or robot) it goes through what is called **compilation**. When it compiles, it gets turned from the more human readable version of Java into the more machine readable version of Java, so then you can run your code.

So roughly you will go through the process, write code, compile, run code. Now, it may end up going something more like write code, compile, fix, compile, run, write more code, compile, run code. You do have to write code before compiling and compile code before running it, but you won't be doing each one just once.

BASIC CODE

Output

The most basic piece of code you can write is to write some words on the screen. Here's the code:

```
System.out.println(“Hello, World”);
```

You can put pretty much anything you want in between the parentheses, but if you want to output text it will have to be in between quotation marks.

Skeleton code

This isn't going to be the only code in your file, though. For now your files will look something like this:

```
import java.util.*;
import java.lang.*;
import java.io.*;


public static class Main {
    public static void main(String[] args) {
```

```
        System.out.println(“Hello, World”);
    }
}
```

Any of the code that we’re talking about for now is going to go inside the curly braces { } where the System.out.println() code is listed.

File names

All your file names will end in .java . The name of your file will need to match the word that follows `public static class`. So if, like the code above you have `public static class Main`, your file will be Main.java. If you have `public static class Robot`, it will be Robot.java.

Comments

Comments are an important way to add extra information into your code. They do absolutely nothing on the computer or robot but they help a lot for the person writing the code. They can explain why or how you did something to the next person reading the code. Please write them. You won’t know what that 7 does the next time you look at your code, and if you don’t know, the next person to look at it definitely won’t.

Single line comments look like this:

```
// All the text after the slashes is a comment
```

Multi-line comments look like this

```
/* All text in between
the first slash star and the
ending star slash is considered a comment */
```

Semicolons

Note that most lines in Java end in a semi-colon. Rows that end in a curly brace don’t need one, but generally everything else.

RUNNING CODE

Where can I run code

Eventually you will need to run Eclipse on a Windows laptop for robot code. You may want to start coding in that now to learn it.

BlueJ is another good option for something you can run easily on a laptop.

There are some things that you can also run online so that you can work on an ipad, but they may be limited in scope. http://www.ideone.com may be a good start.

Compilation errors

If compiling doesn't work, you will small red underlines and/or a small red x next to lines of code that have errors. Typically hovering over that error icon will give you more information about the error.

Console

Anything you output will show up in an area called the **console**. When you start using a new IDE (like Eclipse, BlueJ, or ideone.com) know where that is if you want to print things out. You may have to enable it in order to see it.

VARIABLES

What are they

A **variable** is a storage bucket in code. It is a place specifically set aside to store a value, so that it can be used later on in code.

Variables have a few different things:

- A name: in order to refer to them in code we're going to have to refer to them by some name.
- A type: variables can only store a particular type of values
- A value: since it is a bucket for storing something we can actually put something in it

Types

Here are some of the basic types that are available in Java:

- `int` : used for numbers, short for integer so, whole numbers only
- `float/double` : also used for numbers, but allows for decimal numbers. double allows for more digits, so more decimal places or larger numbers.
- `String` (this one has a capital letter at the beginning) : This is used for text.

Syntax

Declaration

To create a variable it first has to be declared. A variable declaration looks like this:

```
int x = 5;

float y = 4.5;

double z = 30.2;
```

You can see the three parts of the variable in the declaration. First the type, then a name, an equals sign, a value then a semicolon.

String variables

String variables look a bit different, because you need quotation marks around the value, but otherwise look pretty much the same.

```
String s = “value”;
```

Using variables

After you declare a variable, you can simply use its name to refer to the value. So examine this snippet of code:

```
int a = 2;

System.out.println(a); // This prints out 2.

// Yes you can use println for variables!

a = 5;

System.out.println(a); // This prints out 5.
```

Note that after the first line, you no longer have to put the part declaring that it is an int variable, that is already known by the compiler. I can change the value, but I can't change the type. So if I tried to put a line calling a a double or String or trying to set it to the value 4.5 or "Chad", it wouldn't compile.

Rules for variable names

There are some rules on what you can (and can't) name your variables:

- Variables may only contain letters, numbers, and underscores (_)
- Variables may not start with a number
- Every variable must have a unique name
- Variable names cannot be words that have other meanings in Java (you've already seen things like int, float, public, class, but there are more)

There are also things you should do when naming variables:

- Make your variable names clear. x y or z are not as clear as things like robotSpeed.
- Variable names can be any length but keep them shorter
- Java variables are often camelCase, which means that it starts with a lowercase letter and then if there is another word in the variable name it is capitalized.

Input

Sometimes you may want to input something from the console. That snippet of code looks like this:

```
import java.util.*;
import java.lang.*;
import java.io.*;

public static class Main {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int a = s.nextInt(); // This will read a int from the console
        String str = s.next(); // This will read a string
```

```
    }

}
```

TRY IT OUT!

1. Read a variable (or more than one) from the console and print it out.
2. Change the value of a variable in your code.