Introduction to Java Programming

Part 3: Loops, Case Switches, More Program Flow

REVIEW FROM LAST TIME

Imports, Packages, and Java Files

```
import java.util.Scanner;
```

This tells the Java compiler that there is additional code (not included by default) needed to run this class.

These two identifiers tell the compiler **where** to look to find this additional code. “java” and “util” are **packages**, which are essentially folders, in the JDK, and this points the compiler into these folders. While they are included with the JDK, they are not compiled by default.

This tells the compiler **which file** it is specifically looking for. In this case, it is a class contained in a file called Scanner.java

Reference Variables, Objects, and Data Streams

```
Scanner inputStream = new Scanner(System.in);
```

Here, we are declaring a variable called inputStream.

Remember, this isn't one of the types we had originally talked about being built in to Java!

The equals sign should immediately tell you that this is an assignment operation!

This syntax shows that we are **constructing** an **instance** of the Scanner **class** with an **argument**, System.in

Scanner is a **class**, and inputStream is an **instance** of that class. It is an **object**.

Accessing Object Methods

```
String inString = inputStream.nextLine();
```

The left side should be familiar now! We are declaring a variable called “inString” that is of type String.

Again, a single equals sign should scream assignment!

InputStream is an **instance** of the Scanner **class**. The period indicates we are accessing some **field or method**, or value, of the object. In this case, the parentheses indicate we are calling a **method** named “nextLine”

How can we add variables of different types?

```
System.out.println(a + " * " + b + " = " + (a * b));
```

- In this case, the “+” operator is no longer performing addition. It is only addition when both of the operands are numbers.
- Here, because one of the operands is a string, the “+” is instead performing concatenation. Concatenation is an operation that basically pushes two String variables together to form one string.
- a and b can be implicitly converted, or **cast**, to Strings, because every number has a defined representation in text.

The Equality Operator

We hadn't seen this before: `(a==b)`

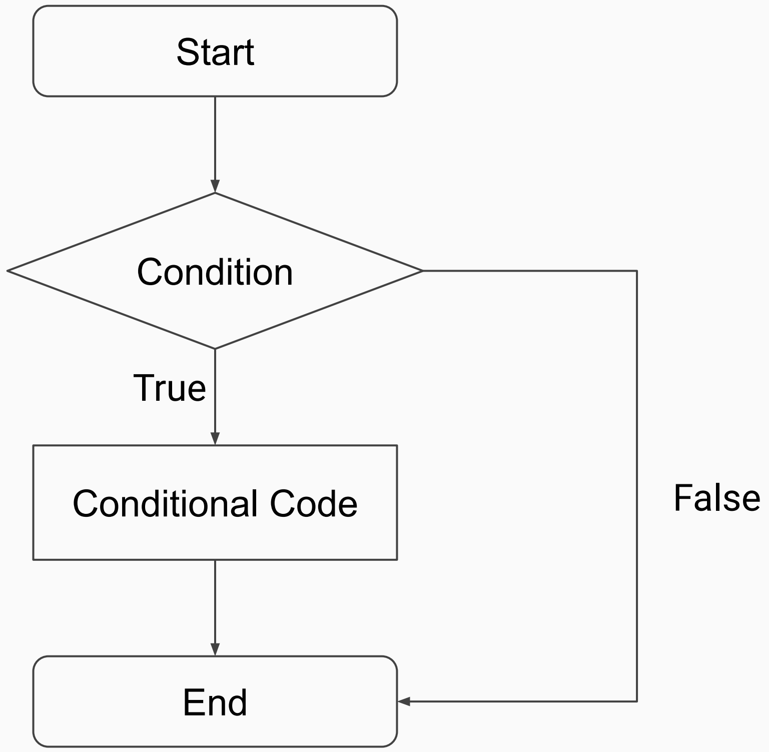
- It is different from the single equals sign, which represents variable assignment.
- The double equals sign serves to test whether the value on the left is equal to the one on the right.
- Remember the boolean operator? The expression (a==b) evaluates to a boolean value. Either a is equal to b - **true**, or a is not equal to b - **false**.
- An important thing to reiterate is that this will only work as expected with **primitive variables.** With reference variables like Strings and other objects, the equality operator's behavior will be unexpected.

END OF REVIEW

Structure of an If Statement

```java
if(a == 1) {
    System.out.println("a is 1");
} else if(a == 2) {
    System.out.println("a is 2");
} else {
    System.out.println("a is something else");
}
```

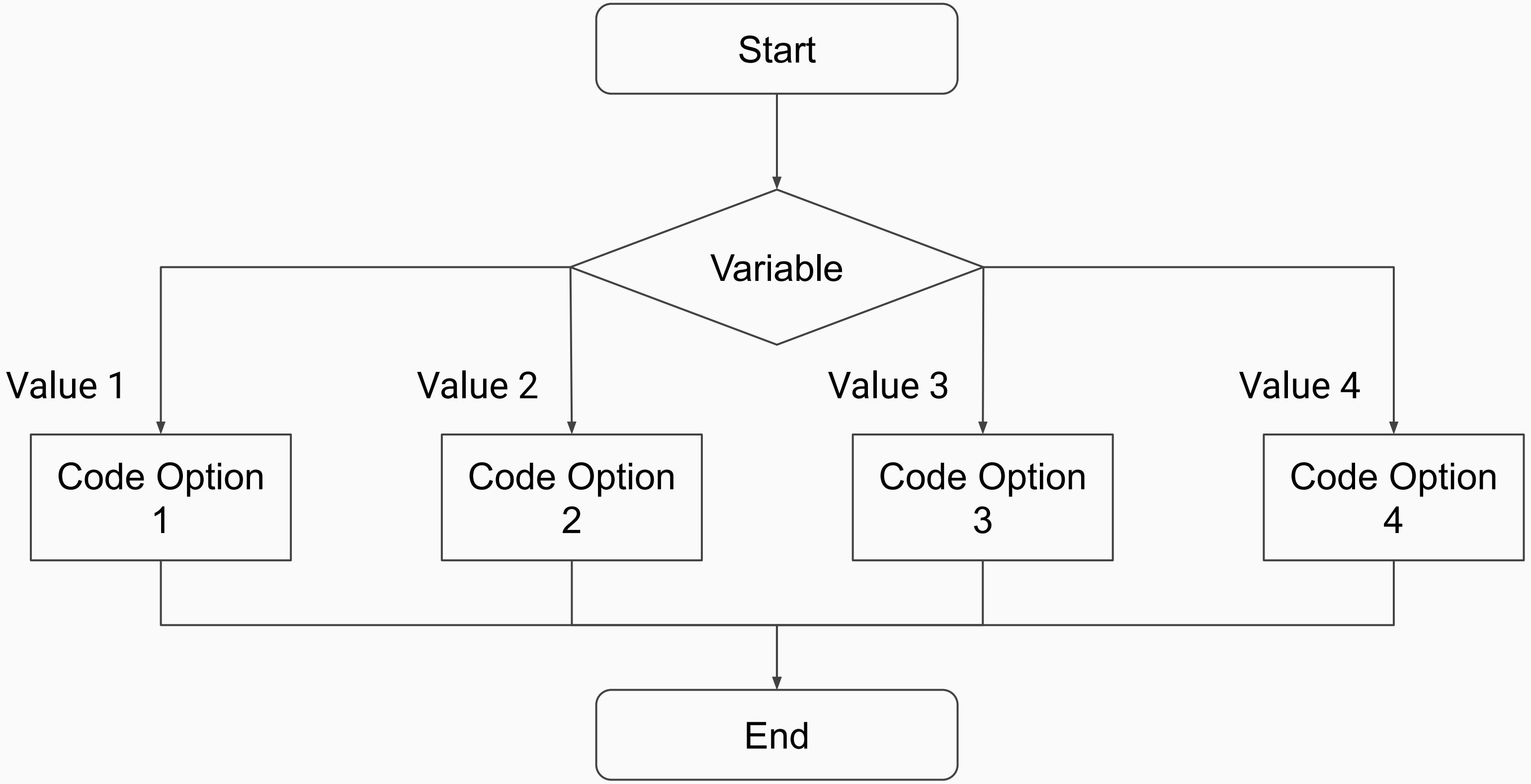
If Statement Logic



Structure of a Case Switch

```
switch(a) {
    case 1: System.out.println("a is 1");
        break;
    case 2: System.out.println("a is 2");
        break;
    default: System.out.println("a is something else");
        break;
}
```

If Statement Logic



Case Switches vs. If/Else

- Both can be used to change the behavior of a program based on certain conditions
- However, case switches only work with ints, bytes, chars, shorts, Strings, and enums (we will talk about enums later). They only work with specific values - not ranges.
- Case switches are only useful when you have a range of behaviors corresponding to specific values of a variable
- On the other hand, if statements are much more broadly useful. They execute based on whether their condition evaluates to true or false, and therefore can be used to change program behavior based on a range of values
- **Let's try to rewrite our calculator program from last time with case switches instead of if/else!**

An Aside: Logical and Comparison Operators

```
boolean a,b;
        a&&b;
        a||b;
        !a;
int x,y;
        x>y;
        x>=y;
        x!=y;
```

These operators act on booleans. They are:
&& (and) - evaluates to true if a AND b are true
|| (or) - to true if a OR b are true
! (not) - negates the value of the boolean it is in front of

These operators act on other types. They are:
> (greater than) - evaluates to true if the first variable is numerically greater than the second
>= (greater than or equal to)
!= (not equal)
Of course, less than and less than or equal to are also available and are omitted for space.

Another Aside: More Arithmetic Operators

```
int x = 5, y = 3;

        x+y;  8
        x-y;  2
        x*y;  15
        x/y;  1
        x%y;  2
        x++;  x = x + 1;
        x--;  x = x - 1;
        x+=y; x = x + y;
```

You might notice that the last three do not evaluate to simple numbers. This is because they are actually assignment statements. x++ is equivalent to x = x + 1, and as such it increments the value of the variable x by 1. Important to note is that the += assignment operator has equivalents for -, *, /, and %.

Finally, Where Are the Exponents?

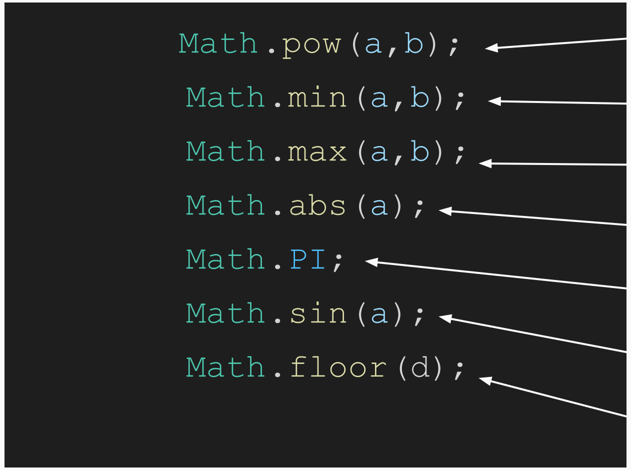
So far, we haven’t seen any exponent support in Java! Where is it?

The usual “^” carat symbol is used for something called bitwise XOR, which we will probably never need to use. But this leaves us with no clear way of doing exponents, so what is the answer?

As it turns out, the Java language provides us a set of static methods that can achieve other common mathematical functions within the Math class.

Note that because they all can be called without creating an instance of Math, they are all static methods!

Java Math Functions



```
Math.pow(a,b);
Math.min(a,b);
Math.max(a,b);
Math.abs(a);
Math.PI;
Math.sin(a);
Math.floor(d);
```

a^b

Returns whichever is greater

Returns whichever is lesser

Returns absolute value

Pi as a double

Returns sine of value (other trig functions are available)

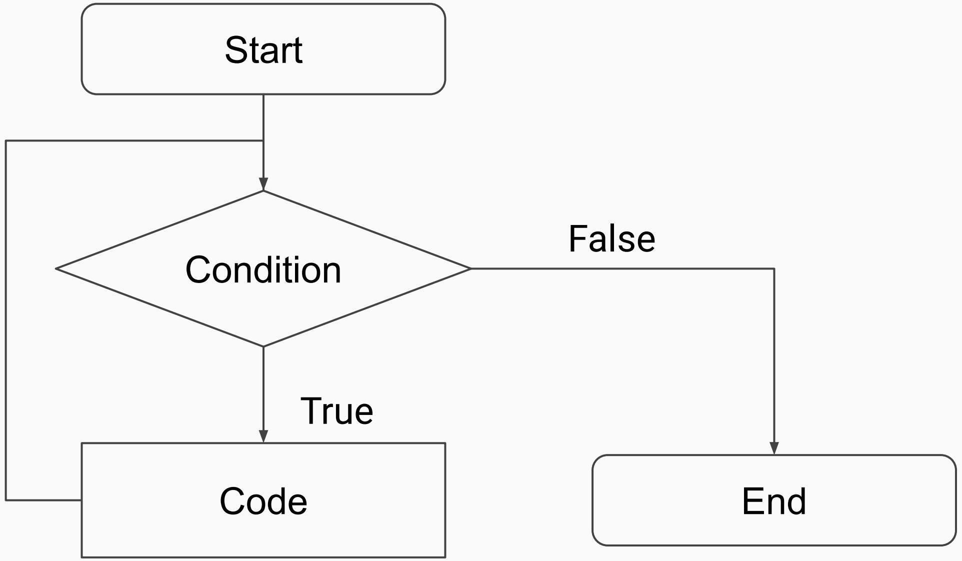
Returns the input rounded down to lower integer (5.8 -> 5)

Project: Learning about Loops

Sometimes, you want a piece of code to repeat a certain number of times without you having to retype it.
While loops, for loops, and do-while loops allow you to accomplish this easily.
They are some of the most important concepts to understand in Java and most programming languages.

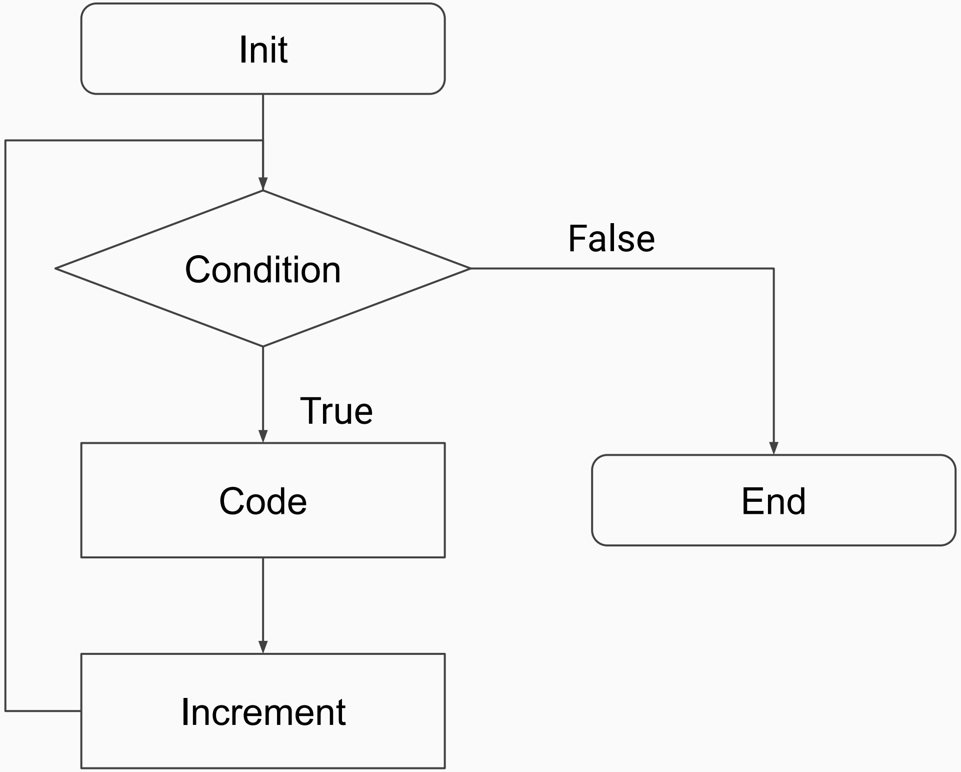
While Loop Syntax and Logic

```
while(condition) {
    //code
}
```



For Loop Syntax and Logic

```
for(init; condition; increment)
{
      //Code
}
```



Project: Learning about Classes

At this point, from different analogies, snippets of code, and explanations, you should have a basic idea of what a class is, conceptually. However, it becomes hard to see how that is useful if you haven't made full use of them.
Classes in Java are the fundamental unit of the language. As an Object Oriented Language, working with classes is beneficial in all but the most simple applications.

IF THERE’S TIME: Arrays, Lists, Data Structures