

# Logic in Java

CPSC 2150

# `if` Statements

- Java has the same syntax for `if` statements as C++
- The only thing that is different is that 0 and 1 do not evaluate to Boolean values in Java like they will in C++

# Loops

- `for` and `while` loops in Java have the same syntax as they do in C++
- There is an additional loop syntax in Java, which can be helpful for arrays (or something that can evaluate to an array)
  - There are two parts:
  - **Declaration**
    - Declare a new variable that will only be used in this loop. It should be the data type of the items in the array you are trying to loop through
  - **Expression**
    - This is the expression that evaluates as an array. It can be an array, or a function call that returns an array
- **Syntax:** `for (<Declaration of variable> : <Expression/array>) {}`
  - It is helpful to think of it as “for-each variable in array”

# Extra for Loop Example

```
int[] numbers = {10, 20, 56, 67};  
for (int x : numbers) {  
    System.out.println(x);  
}  
// prints every number x in the array numbers
```

# Functions (Methods)

- Functions (or methods) in Java and C++ are almost exactly the same
- There are two important differences:
  1. In Java, every function belongs to a class (because everything belongs to a class)
    - This means that every function must have a visibility (`private`, `public`, etc.) defined for it
    - It also means that declaring a `static` function is much more common (we'll discuss `static` functions in detail later in-class)
  2. We must carefully consider how we pass parameters to a function in Java

# Passing Parameters to Functions

- In C++, we could choose to pass parameters to either by value or by reference
  - **By value** means that a copy of the parameter was made, and changes would not be reflected outside of the function
  - **By reference** means that we passed in the reference (pointer/memory location) to the variable, so changes made inside the function would affect the variable outside of the function
- In Java, we don't have control over the pointers, referencing and dereferencing
- Everything in Java is passed by value, however reference type variables have a value of a memory location
- This means that all primitive types are passed by value, and everything else is essentially passed by reference
- We can have aliasing issues here, since any change to a reference type variable made inside the function will be made outside the function as well

# The End

- Continue to the video on **Classes in Java**