Java Basics

Assignment

When programming, we often work with variables to help us achieve whatever task we are to complete. We declare variables in Java with the type of the variable followed by the variable name.

```
int age;
```

This bit of code **declares** into existence a variable with name *age* that holds ints (integers). We can then assign an int value to that *age* variable like so and conduct operations using *age*, just like in math.

```
age = 5;
System.out.println(age + 5);
```

This will print to the console the value 10, since age has the value of 5 and $5 + 5 = 10$. We can also declare a variable and assign a value to that in one line. We call that **initialization**.

```
int age = 5;
```

An important thing to note with the assignment statements is that the right hand side of the assignment always gets executed first and then that final value is assigned to the variable.

```
age = age + 5;
```

After this line of code, *age* will have the value of 10, because *age* was previously 5, $5 + 5 = 10$, and this final value is reassigned to *age*.

When it comes to variables, there are two main types, primitives and objects.

Primitives

Primitives behave just like variables in math. When you assign a variable a value, that variable is that value. You can think of it as a box with the value inside of it. The box is the variable and whatever is inside the box is what the variable is equal to. There are 8 primitive data types: boolean, byte, char, short, int, long, float, double.

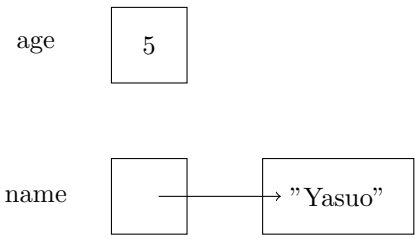
Objects

Objects are a little different. Variables assigned to objects are actually **pointers** to the object. This means if the box is the variable, then inside the box is an arrow that points to some blob, which is the object. Understanding and being able to draw these diagrams will be crucial skill to have moving forward.

Java Code:

```
int age = 5;
String name = "Yasuo";
```

Diagrams:



Conditionals

If Statement

We may want to only execute a portion of code if a specific condition is met. To do this we use the *if statement*.

```
if (boolean condition) {
    conditionally executed code
}
```

The code inside the two curly braces will only be executed if the *boolean condition* is a true value. Note that the *boolean condition* could simply be a single boolean variable.

```
boolean isFalse = true;
if (isFalse) {
    System.out.println("It is false.");
}
```

Else Statement
We can use an *else statement* if we want to execute two different pieces of code based on a single condition.

```
if (boolean condition) {
    Code 1
} else {
    Code 2
}
```

If the *boolean condition* is true, then **Code 1** will be executed and **Code 2** will not, while if the *boolean condition* is false, then **Code 2** is executed and **Code 1** will not.

Else If Statement
If we want more conditional checks in the same block, we can add *else if statements*.

```
if (condition 1) {
    Code 1
} else if (condition 2) {
    Code 2
}
...
else {
    Code n
}
```

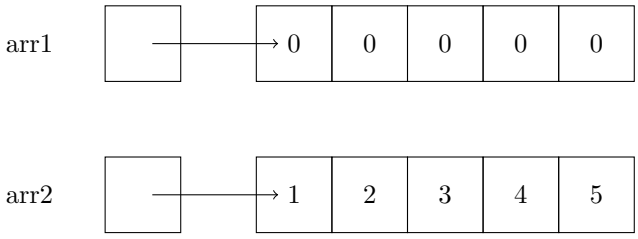
You can add as many *else if statements* as you want in between the *if* and *else*. Each *else if* will execute if the previous conditionals are false.

Arrays

If we want to store multiple values in a single object, we use arrays. We can initialize an array in multiple ways.

```
// creates an array of length 5
int[] arr1 = new int[5];
// creates an array of length 5 with values 1, 2, 3, 4, 5
int[] arr2 = new int[] {1, 2, 3, 4, 5};
```

The general syntax for arrays is the type of value the array stores followed by square brackets as you see above. In Java, arrays are fixed length, so if you make an array of a specific length, you cannot change that length. For arrays that do not specify what each value is upon initialization have default values. For *ints* the default value is 0 and for *objects* the default value is *null*. Here is the box and pointer diagram for the code snippet above.



Arrays are indexed started at 0, so for a 5 item array, the specific indices are 0, 1, 2, 3, 4. You can access elements of an array based on index with square brackets. You can also change the array with indices and square brackets. You can retrieve the length of an array with the *length* field.

```
// val will have the value of 3 since 3 is the value at the 2nd index of arr2
int val = arr2[2];
// this changes the first value of arr1 to be 1 instead of 0
arr1[0] = 1;
// len will have the value of 5 since arr2 is of length 5
int len = arr2.length;
```

Loops

If we want to execute the same code or a similar code multiple times, we can take advantage of loops

While Loop

A *while loop* will check a condition each iteration and execute the code snippet as long as that condition is true.

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

The code snippet will usually contain a line of code that modifies a variable in the condition, so the loop will eventually terminate. Consider the following.

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i = i + 1;
}
```

For Loop

Another form of loop is the *for loop*. It is exactly the same as the while loop with different syntax.

```
for (variable initialization; condition; update) {
    code snippet
}
```

Within the parenthesis of the *for loop* there are three main parts, variable initialization, condition, and update. The previous *while loop* can be written as.

```
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

Note: $i++$, $i+=1$, and $i = i + 1$ all increment i by 1.

Enhanced For Loop

The final loop is the *enhanced for loop* or aka the "for each loop". This is only used when we want to iterate through arrays or collections (will learn later). It has the following syntax.

```
for (item : array/collection) {
    code snippet
}
```

This loop can be read as "for each item in the array/collection, do the code snippet". Here is a more concrete example using *arr2* from above.

```
for (int i : arr2) {
    System.out.println(i);
}
```

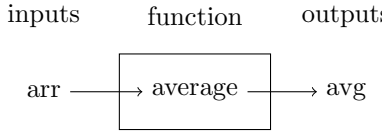
This loops means, for each int value in arr2, print that value to the console.

Functions

For now you can think of functions code snippets that serve specific purposes when called. Let's break down this function.

```
public static double average(double[] arr) {
    double sum = 0;
    for (double val : arr) {
        sum += val;
    }
    double avg = val / arr.length
    return avg;
}
```

The first line tells us about the function itself. The keyword *double* means that this function will return a value of type *double*. The "average" tells us that this is the name of the function and *double[] arr* is what we call parameters or arguments, you can think of them as inputs. The *public* and *static* are not important to understand as of right now. The rest is code to be executed and the function ends with a *return* signalling what the function will spit out after it's done. You can think of functions as simply inputs and outputs.



Calling/using a function can be done by using the function name and supplying the appropriate paramenters.

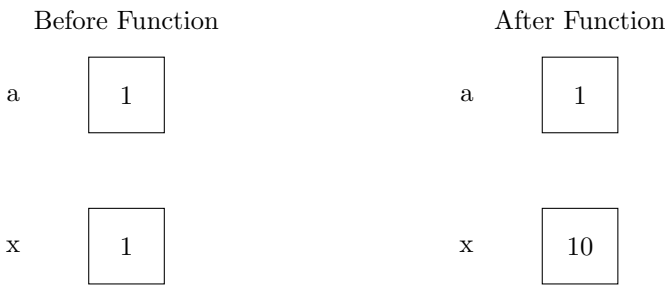
```
double[] lst = new double[]{1.0, 2.0, 3.0};
// use function name and supply parameters to use function
double avg = average(lst);
```

Pass By Value
When it comes to functions, primitive types follow what we call "pass by value". This means the value of the variable gets passed in to the parameter. Consider the following function.

```
public void reassign(int x) {
    x = 10;
}

int a = 1;
reassign(a);
```

You would think this function changes the value of a to be 10, but it simply just changes the value of x and not a. Here's the diagram for the code snippet.



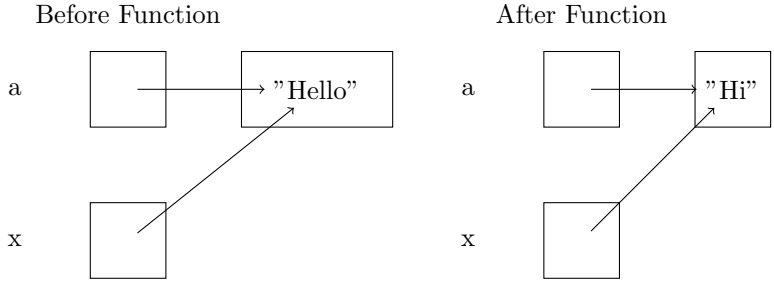
Note: passed in primitive values stay the same after a function call.

Pass By Reference
Objects are a little different when it comes to functions because they are "pass by reference". This means when an object is passed into a function, it's reference is passed in. Consider the same function, except with Strings, which are objects.

```
public void reassign(String x) {
    x = "Hi";
}

String a = "Hello";
reassign(a);
```

The value of a actually get changed because a and x both point to the same String instance, so when x changes the value, a also changes because of it. Here's the diagram.



Pass by Value and Pass by Reference are some of the most important things to remember when it comes to functions, so don't forget about them! Drawing the diagrams really help, so make sure to draw them.

TL;DR

Fully understand pass by value and pass by reference...that is all