Module 1-4

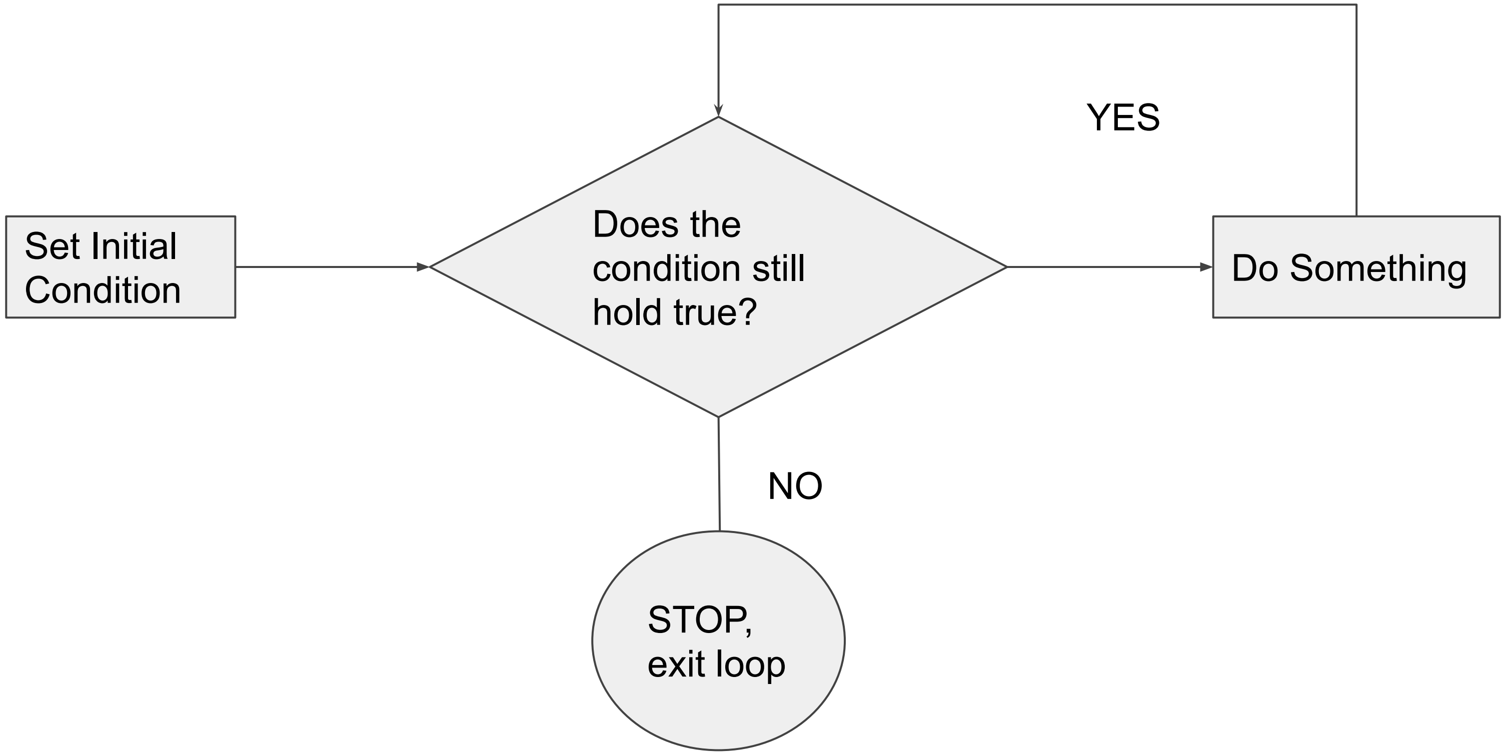
Loops and Arrays

Loops

Loops are designed to perform a task until a condition is met.

- Examples:
  - Print a list of all numbers between 0 and 10.
  - Print all the items in a grocery list.
- There are several types of loops in Java:
  - For Loop (by far the most common)
  - For Each Loop
  - While Loop
  - Do-While Loop

Loops: Visualized



Loops: For Loops

For Loops are the most common types of loops. They follow this pattern:

```
for (initialize index ; check condition ; increment or decrement index) {

    // action to repeat

}
```

Loops: For Loop Examples

Here is an example:

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

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

This code will print “Developers!” five times, followed by the numbers 0 to 4.

Loops: For Loop Visualized

Let’s consider this example again:

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

Each run of the loop is called an iteration. You can generally tell how many iterations the loop will run for just by looking at the code. In this example, we expect 5 iterations.

Iteration #	Value of i at beginning	Action	Value of i at end	Is i less than 5?
1	0	Prints 0	1	Yes
2	1	Prints 1	2	Yes
3	2	Prints 2	3	Yes
4	3	Prints 3	4	Yes
5	4	Prints 4	5	No

Loops: While & Do-While Loops

Here is an example:

```
int i = 0;

while (i < 5) {
        System.out.println(i);
        i++;
}
```

```
int i = 0;

do {
        System.out.println(i);
        i++;
} while (i < 5);
```

- For both the while and do-while you must increase or decrease the index manually.
- The do-while is guaranteed to execute **at least once**.

Loops and Scope

Be mindful of variable scoping when using loops, these two examples will not compile on the indicated line.

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

```
int i = 0;

while (i < 5) {
        System.out.println(i);
        int k = 9;
        i++;
}
k = 15; //problem
```

Let’s practice writing loops

Arrays

Arrays are a collection of elements having the same data type.

- Examples:
  - A roster of names
  - The weather report
  - In sports, the points earned per inning / quarter / half
- In Java, this would mean:
  - An array of Strings
  - Also an array of doubles
  - An array of int’s.

Arrays: Declaration Syntax

An array has the following syntax:

```
int [ ] team1Score = new int [4];
```

Start by defining a data type followed by an empty set of square brackets.

Give your array a name

On the right of the equal sign, you need to type the keyword new followed by the data type and another pair of square brackets. Inside the brackets you need to specify the size of the array.

Arrays: Assigning Items

We can assign items to individual slots in an array:

```
int [] team1Score = new int [4];

team1Score[0]= 20;
team1Score[1]= 14;
team1Score[2]= 18;
team1Score[3]= 23;
```

This array has 4 slots.

These slots are called “elements.” We can access elements by specifying a slot number, i.e. 0,1,2,3.

This slot number is called an index.

index	0	1	2	3
value	20	14	18	23

Arrays: Iterating

Going back to our basketball example, let’s say we want to print the score for each quarter. This might be the first thing that comes to mind:

```
int [] team1Score = new int [4];
team1Score[0] = 20;
team1Score[1] = 14;
team1Score[2] = 20;
team1Score[3] = 23;

System.out.println( team1Score[0] );
System.out.println( team1Score[1] );
System.out.println( team1Score[2] );
System.out.println( team1Score[3] );
```

This seems like a very inefficient way to accomplish this task. There is an easier way to do this.

Arrays: Iterating

We can use a loop to sequentially iterate through each element of the array.

```
int[ ] team1Score = new int[4];
team1Score[0] = 20;
team1Score[1] = 14;
team1Score[2] = 20;
team1Score[3] = 23;

for (int i = 0; i < team1Score.length; i++) {
        System.out.println(team1Score[i]);
}
```

Arrays: Iterating (in slow motion)

Given the previous example, an array containing {20, 14, 20, 23}

Iteration #	i	teamScore[i]	Is i < team1Score.length ?
1	0	20	i has increased to 1, 1 < 4 so yes, iterate again
2	1	14	i has increased to 2, 2 < 4 so yes , iterage again
3	2	20	i has increased to 3, 3 < 4 so yes, iterate again
4	3	23	i has increased to 4, 4 < 4 so no. No more iterations, loop ends.

Arrays: Alternative Declaration

If you know the values you want to place in an array ahead of time, consider using this condensed format to declare an array:

```
int [ ] team1Score = {4, 3, 2};
String[ ] captains = { "April", "Pike", “Kirk”};
```

The Increment/Decrement Operator

The increment (++) and decrement operator (--) increases or decreases a number by 1 respectively. You have seen this in the context of a for loop. Here is a more general example (the output is 94):

```
int x = 93;
x++;
System.out.println(x);
```

- If the operator is behind a variable it is a **postfix operator** (i.e. x++).
  - For postfix, we use the variable first, then increment
- If the operator is in front a variable it is a **prefix operator** (i.e. ++x).
  - For prefix, we increment first, then use the variable

The Increment/Decrement Operator: Example

Using the prefix vs a postfix can have unexpected consequences:

```
int a = 3;
System.out.println(++a + 4); // prints 8

int b = 3;
System.out.println(b++ + 4); // prints 7
```

- In the first example, we have a prefix operator, a is increased first and becomes 4, then it is used in the operation (4+4).
- In the second example we have a postfix operator, b is used in the operation first (3+4), then increased.

Let’s work with some arrays

Summation Shortcut

Suppose you have some sort of running total, in this case the sum of all the even numbers:

```
int totalSumEvens = 0;
for (int i=0; i < 13; i++) {

        if (i%2 == 0) {
                totalSumEvens = totalSumEvens + i;
        }
}
System.out.println(totalSumEvens);
```

We can rewrite *totalSumEvens = totalSumEvens + i* as *totalSumEvens += i;*

Summation Shortcut

Here is the alternate version:

```
int totalSumEvens = 0;
for (int i=0; i < 13; i++) {

        if (i%2 == 0) {
                totalSumEvens +=  i;
        }
}

System.out.println(totalSumEvens);
```

You can also implement the following for subtraction and multiplication, and division:: -=, *=, /= .