

Module 1-4

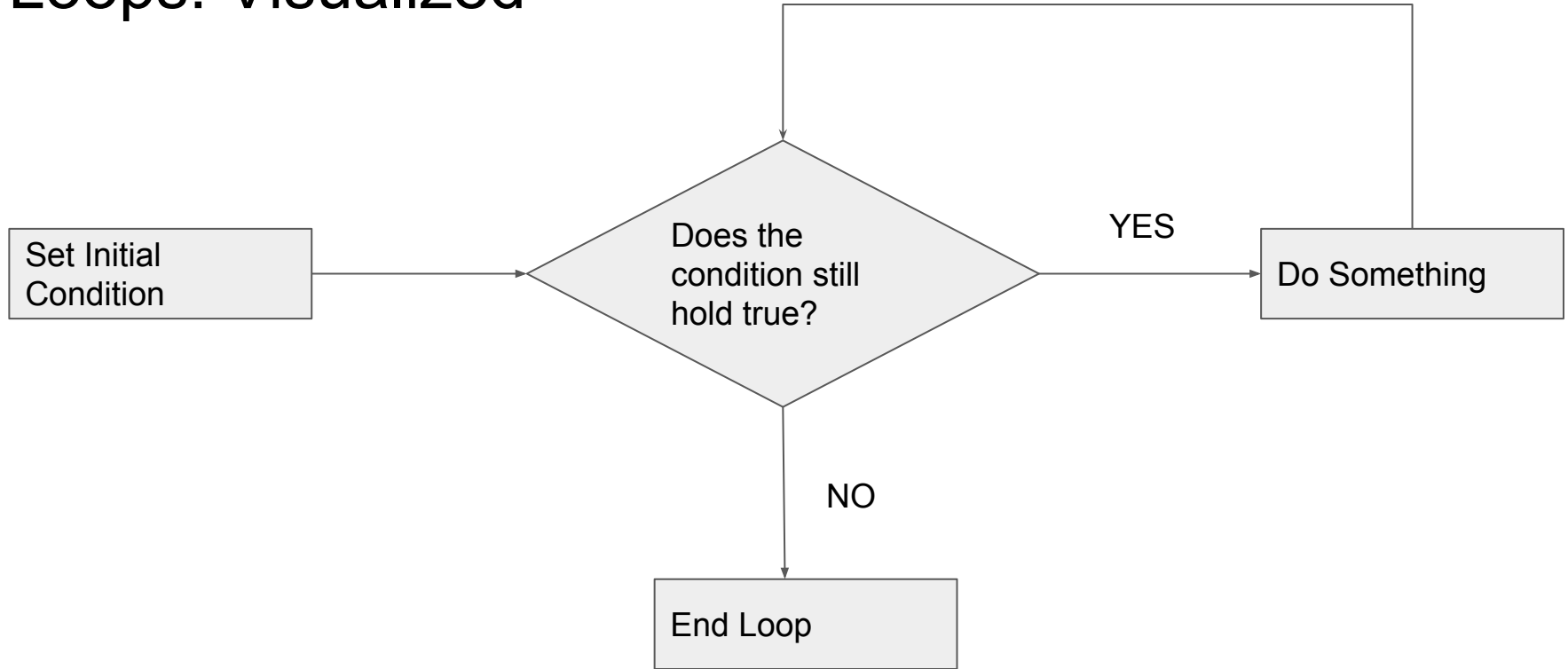
Loops and Arrays

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)
 - 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:

```
public class MyClass {  
    public static void main(String[] args) {  
        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**.

Arrays

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

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

Arrays: What if we didn't use them

Let's define the points scored per quarter in a basketball game.

```
public class Basketball {  
  
    public static void main(String[] args) {  
        int homeTeamQ1Score = 20;  
        int homeTeamQ2Score = 14;  
        int homeTeamQ3Score = 18;  
        int homeTeamQ4Score = 23;  
  
        int awayTeamQ1Score = 20;  
        int awayTeamQ2Score = 26;  
        int awayTeamQ3Score = 10;  
        int awayTeamQ4Score = 27;  
    }  
}
```

Suppose we needed to create variables that tracked the scores per quarter. There are 4 quarters in a basketball game, and there are 2 teams...so we would need 8 variables!

Arrays: What if we did use them

The previous example can be implemented with an array.

```
public class Basketball {  
  
    public static void main(String[] args) {  
  
        int [] team1Score = new int [4];  
        int [] team2Score = new int [4];  
  
        team1Score[0]= 20;  
        team1Score[1]= 14;  
        team1Score[2]= 18;  
        team1Score[3]= 23;  
  
        team2Score[0]= 20;  
        team2Score[1]= 26;  
        team2Score[2]= 10;  
        team2Score[3]= 27;}}
```

We created 2 arrays of integers, team1Score and team2Score. We have set the length of each one of these arrays to 4 elements.

Arrays: Declaration Syntax

An array has the following syntax:

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

Give your array a size. **Arrays are of fixed size.**

Give you 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.

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

Arrays: Assigning Items

We can assign items to individual elements in an array:

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

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

Note that this array has 4 slots.

These slots are called “elements.” We can access elements by specifying a number starting from 0. This number that designates the element 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];  
  
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 must be a better way! (Spoiler Alert: there is)

Arrays: Iterating

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

```
public class Basketball {  
  
    public static void main(String[] args) {  
  
        int[] team1Score = new int[4];  
  
        team1Score[0] = 20;  
        team1Score[1] = 14;  
        team1Score[2] = 18;  
        team1Score[3] = 23;  
  
        for (int i = 0; i < team1Score.length; i++) {  
            System.out.println(team1Score[i]);  
        }  
    }  
}
```

- The value of team1Score.length will be 4.
- The loop will iterate three times, once for i=0, once for i=1, and once for i=2. After that, i is no longer less than team1Score.length.
- Note that i is also used to specify the position of the array so that it knows which element to print.

Arrays: Iterating (in slow motion)

Given the previous example, an array containing {20, 14, 18, 23}

Iteration #	i	teamScore[i]	Is i < team1Score.length ?
1	0	20	i has increased to 1, 1 < 4 so yes
2	1	14	i has increased to 2, 2 < 4 so yes
3	2	18	i has increased to 3, 3 < 4 so yes
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:

```
int[] team1Score = {4, 3, 2};  
String[] lastNames = { "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 in behind a variable it is a **postfix operator** (i.e. x++).
 - A postfix operator is evaluated first, then incremented.
- If the operator is in front a variable it is a **prefix operator** (i.e. ++x).
 - A prefix operator is incremented first, then evaluated.

The Increment/Decrement Operator: Example

A choice of having a prefix or a postfix in certain calculations 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, it is used in the operation (4+4).
- In the second example we have a postfix operator, a is used in the operation first (3+4), then increased.

Order of Operations... Given what we know

++ or --
PEMDAS (Arithmetic Rules)
Equality Operators (== and !=)
AND / OR (&&,)

Items at the top of the list take higher priority.