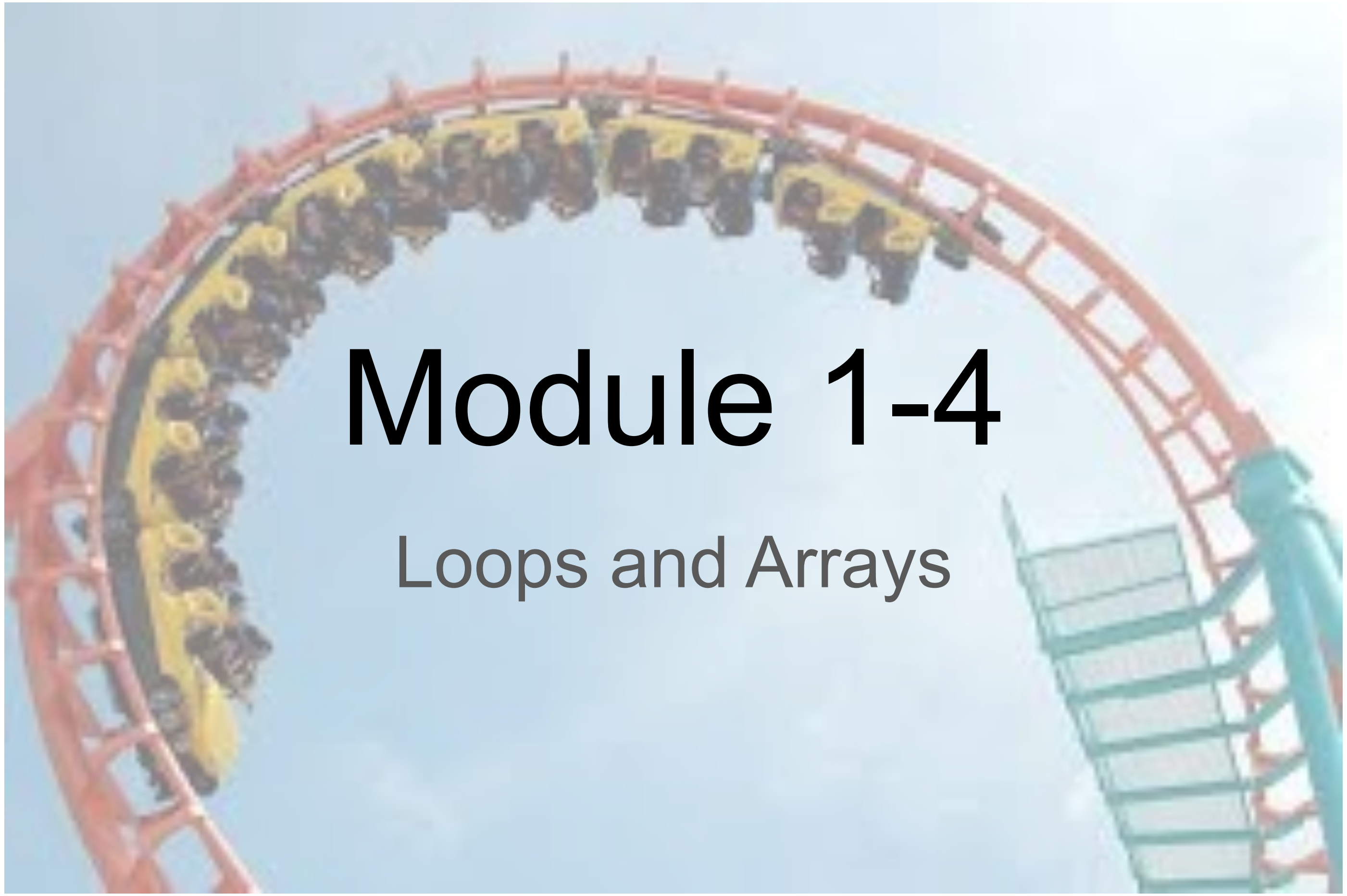


Module 1-4

Loops and Arrays

Module 1 Day 4

Can you?

1. … explain the concepts of arrays
2. … perform the following tasks associated with arrays:
    a. Create an array, Initialize an array, Retrieve values stored in an array
    b. Set/Change values in an array, Find the length of an array
    c. Use a for-loop to iterate over the elements in an array
3. … explain the limitations of arrays
    a. You can't change the length of an existing array
    b. Arrays can only hold values that match the data type of the array**
4. ... describe how to perform the following tasks with arrays:
    a. Get the first element of an array
    b. Get the last element of an array
    c. Change each element of an array
5. … explain the concepts related to variable scope and why it is important
6. … use the increment/decrement short assignments in a program (++ / –)
7. … use the debugger in the IDE to walk through your code

Loops and Arrays

Arrays

Arrays: Life without them ...

Let’s track 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;
        }
}
```

Using variables, we would need to create a variable to track the scores per quarter for each team.

With 4 quarters in a game, and 2 teams, we would need 8 variables.

Arrays

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

- Examples:
  - A roster of names
  - The 10-Day weather report (temperatures)
  - 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: Life with arrays ...

The previous example can also be implemented using 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;}}
```

Instead of 8 variables, we can create 2 arrays of integers, one for each team, as team1Score and team2Score.

We then set the length of each array to 4. That gives the array 4 elements, one for each quarter.

Arrays: Declaration Syntax

An array has the following syntax:

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

4. Give your array a size. **<u>Arrays are of fixed size</u>**.

3. 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 (length)** of the array.

2. Name the array

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

Arrays: Alternative Declarations

If you know the values you want to place in an array ahead of time, you can also use ***inline declaration and assignment*** format to declare them:

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

Arrays: Assigning Items

We can assign values to array elements using their index:

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

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

index	0	1	2	3
value	20	14	18	23

This array has 4 “slots”.

These “slots” are called ***elements***.

Values can be assigned to each element using the index.

Elements can also be accessed by specifying their element ***index***.

<u>Indexes start at 0.</u>

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]);
```

… but this approach has merely transferred to the problem of having multiple variables for each score to the new problem of having to write a *println* for each *Element* in each array.

Loops and Arrays

Loops

Loops

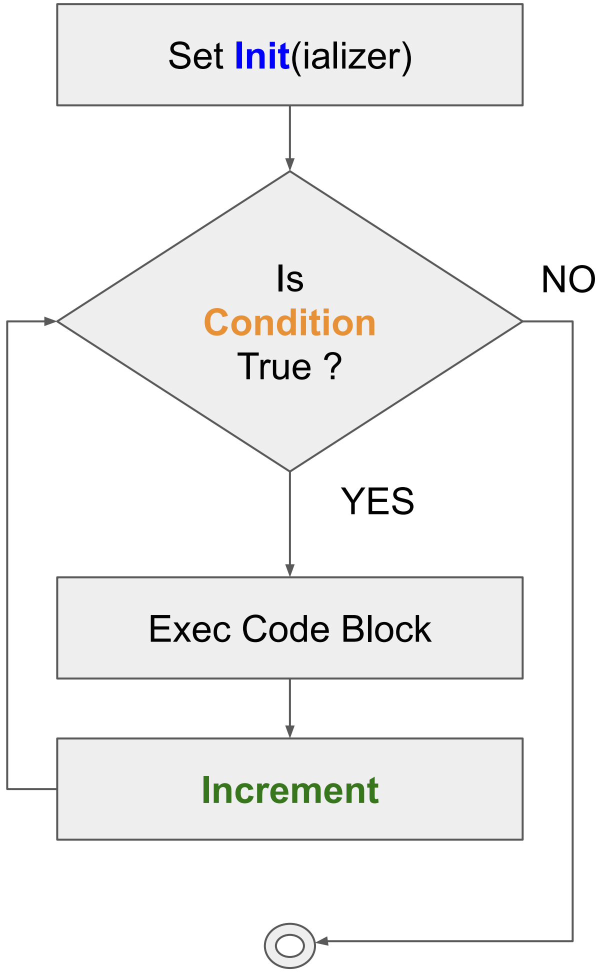
Loops are designed to perform a task until a condition is met.

- 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

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



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) {

    // code to execute FOR EACH iteration of the loop

}
```

Loops: For Loop Examples

Here are two examples:

```
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

While and Do-While loops do not have a fixed number of iterations. Instead, they ***run indefinitely while*** their ***check condition*** is ***true.***

```
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 loop, you must increase or decrease the loop's index if used ***or set the end condition***, within the loop**

- A do-while loop will always execute its code block ***<u>at least once</u>***.

Loops and Arrays

Lecture Review

The Increment/Decrement Operator

The increment (++) and decrement operator (--) increases or decreases a number by 1, respectively. We have seen this in the context of a for loop, here the increment of variable *x* is used as a statement:

```
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 **evaluates** a variable **first**, *then* increments/decrements.
- If the operator is in front a variable it is a **prefix operator** (i.e. ++x).
  - A prefix operator **increments/decrements** the variable **first**, *then* the variable is evaluated.

The Increment/Decrement Operator: Example

Using Increment and Decrement as either a prefix or postfix operator will changes the way in which they are evaluated.

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

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

- In the first example, we have a prefix operator, ***a*** is incremented first to 4, then it is evaluated (3 -> 4 -> Evaluation).
- In the second example we have a postfix operator, ***b*** is evaluated first, then it is incremented (3 -> Evaluation -> 4).

Arrays & Loops: 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 four times, once for i=0, once for i=1, once for i=2, and once for i=3.
- After the fourth iteration, i will increment to 4 and will 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 & Loops: Iterating (in slow motion)

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

Iteration	**i**	**teamScore[i]**	**Is i < team1Score.length ?**
first	0	20	i has increased to 1, 1 < 4 so yes
second	1	14	i has increased to 2, 2 < 4 so yes
third	2	18	i has increased to 3, 3 < 4 so yes
fourth	3	23	i has increased to 4, 4 < 4 so no. No more iterations, loop ends.

Order of Operations…

++ or -- **(Don’t forget about Pre-Fix and Post-Fix use differences)**
PEMDAS (Arithmetic Rules)
Equality Operators (**==** and **!=**)
AND / OR (&&, ||)

Items at the top of the list take higher priority.