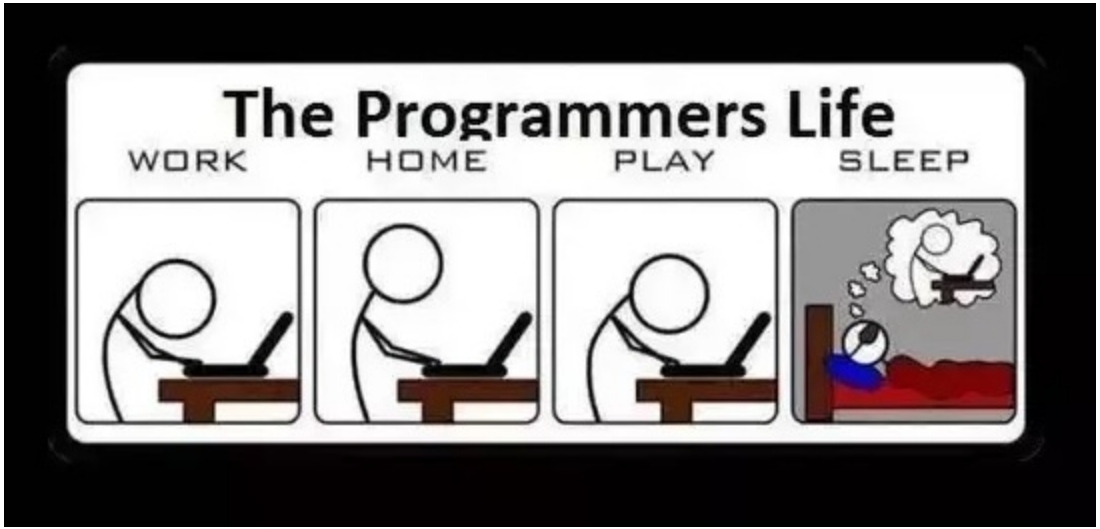
The Programmers Life
WORK
HOME
PLAY
SLEEP

Module 1-4

Loops and Arrays

Objectives

- Understand and explain the concepts of arrays
- Perform tasks associated with arrays:
  - Create an array
  - Initialize an array
  - Retrieve values stored in an array
  - Set/Change values in an array
  - Find the length of an array
  - Use a for-loop to "walk-through" the elements in an array
- Explain the limitations when using arrays
  - Can't change the length of an existing array
  - Arrays can only hold the data types it was declared with

Objectives

- Describe how to perform the following manipulations on arrays:
  - Get the first element of an array
  - Get the last element of an array
  - Change each element of an array
- Explain the concepts related to variable scope and why it is important
- Use the increment/decrement short assignments properly in a program
- Use the debugger in their IDE to walk through the code

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 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.

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;
```

index	0	1	2	3
value	20	14	18	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.

Variable Scope

- Code can be grouped together in **blocks**
  - **Block** is a group of zero or more statements between braces
  - Can be used anywhere a single statement is allowed.
- **Scope** defines where the variable can be referenced
  - **Referenced** means exists, or can be accessed
- **Rules of Scope:**

1. Variables declared inside of a function or block **{..}** are local variables and only available within that block. This includes loops.
2. Blocks can be nested within other blocks and therefore if a variable is declared outside of a block, it is accessible within the inner block.

Variable Scope Example

```
public static void main(String[] args){
        {
            // The variable x has scope within
            // brackets
            int x = 10;
            System.out.println(x);
        }

        // Uncommenting below line would produce
        // error since variable x is out of scope.

         System.out.println(x);
}
```

Variable Scope Example

```
public static void main(String args[]){
        int outside = 5;
        { // inner block
            int inside = 7;
            outside += 15; // outside = outside + 15;
            System.out.println(inside); // will print 7
        }

        // will print 20
        System.out.println(outside);
    }
```

Variable Scope Example

```
public static void main(String args[]) {
       int x = 25;
       if (x >= 20) {
           x /= 5;
           System.out.println(x);
       }

      System.out.println(x);
   }
```

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 loop order of execution:

1. Initialization
2. Condition
3. Block
4. Altering

```
for (initialization; condition; altering ) {
    statements
}
```

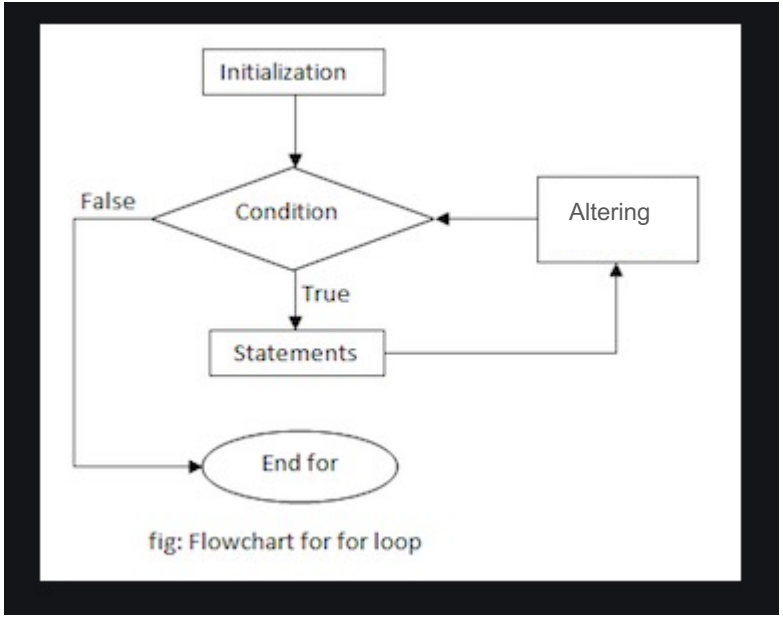

fig: Flowchart for for loop

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(i);
              }
           }
}
```

This code will print 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 **<u>at least once</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]);
```

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 Java Operations…. Given what we know

Precedence	Operator	Type	Associativity
12	() [] .	Parentheses Array subscript Member selection	Left to Right
10	++ -- + - ! (*type*)	Unary increment Unary decrement Unary plus Unary minus Unary logical negation Unary type cast	Right to left
9	* / %	Multiplication Division Modulus	Left to right
8	+ -	Addition Subtraction	Left to right
7	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
6	== !=	Relational is equal to Relational is not equal to	Left to right
5	^	Exclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Larger number means higher precedence.

Objectives

- Understand and explain the concepts of arrays

Index	Value
0	55
1	65
2	42
3	29

Objectives

- Understand and explain the concepts of arrays
- Perform tasks associated with arrays

Create an array -- int [] myScores = new int [5];

Initialize an array –
```
myScores[0] = 90;
myScores[1] = 95;
myScores[2] = 80;
myScores[3] = 75;
myScores[4] = 100;
```

Find the length of an array – myScores.length

Retrieve values stored in an array –
```
System.out.println("The first value is " + myScores[0]);
```

Set/Change values in an array –
```
myScores[3] = 65;
```

Use a for-loop to "walk-through" the elements in an array –
```
for (int i = 0; i < myScores.length; i++) {
      sum = sum + myScores[i];
}
```

Objectives

- Understand and explain the concepts of arrays
- Perform tasks associated with arrays
- Explain the limitations when using arrays

Arrays are fixed length!

Arrays can only hold the data type it was declared with

Objectives

- Understand and explain the concepts of arrays
- Perform tasks associated with arrays
- Explain the limitations when using arrays
- Describe how to perform the following manipulations on arrays

Get the first element of an array – return myGrades[0]

Get the last element of an array –
return myGrades[myGrades.length – 1]

Change each element of an array –

```
for (int i = 0; i < myGrades.length; i++) {
    myGrades[i] += 5;
}
```

Objectives

- Understand and explain the concepts of arrays
- Perform tasks associated with arrays
- Explain the limitations when using arrays
- Describe how to perform the following manipulations on arrays
- Explain the concepts related to variable scope and why it is important

A variable is only alive, or in scope or valid (or available for use) inside the block in which it is declared

Objectives

- Understand and explain the concepts of arrays
- Perform tasks associated with arrays:
- Explain the limitations when using arrays
- Describe how to perform the following manipulations on arrays:
- Explain the concepts related to variable scope and why it is important
- Use the increment/decrement short assignments properly in a program

```
i = i + 1;
    is equal to i +=1;
        is equal to i++; or ++I;
```

```
i = i - 1;
    is equal to i -=1;
        is equal to i--; or –i;
```

Objectives

- Understand and explain the concepts of arrays
- Perform tasks associated with arrays:
- Explain the limitations when using arrays
- Describe how to perform the following manipulations on arrays:
- Explain the concepts related to variable scope and why it is important
- Use the increment/decrement short assignments properly in a program
- Use the debugger in their IDE to walk through the code

Debugging and Breakpoints

- Debugging is process of locating and fixing errors in code.
- Breakpoint is a marker in the IDE that indicates program will pause execution

- Debugging Best Practices
  - Insert breakpoint at start of code to validate
  - Make single change, then test. Repeat
  - Read any error messages that show up
  - **Explain your code to another person**