

Last Class

- While loops
- Infinite loops
- Loop counters
- Iterations

```
public class January31 {  
    public static void main(String[] args) {  
        while (true) {  
            forLoops();  
            if (checkClassUnderstands() ) {  
                break;  
            }  
            teachArrays();  
        }  
    }  
}
```

What does this display?

```
public class AnotherNestedLoop {  
    public static void main(String[] args) {  
        final int MAX = 4;  
        int outerCount, innerCount;  
  
        outerCount = 1;  
        while(outerCount <= MAX) {  
            innerCount = 1;  
            while (innerCount <= outerCount) {  
                System.out.print("(" + outerCount + "," +  
innerCount +  
                    ") ");  
                innerCount++;  
            }  
            System.out.println();  
            outerCount++;  
        }  
        System.out.println("Done.");  
    }  
}
```

$(1, 1)$

$(2, 1)$ $(2, 2)$

$(3, 1)$ $(3, 2)$ $(3, 3)$

$(4, 1)$ $(4, 2)$ $(4, 3)$ $(4, 4)$

Done.

Part 2: The `for` Statement

Example: a “for” loop

```
for(int x = 0; x < 4; x++) {  
    System.out.println("I have to write "  
        + "this over and over.");  
}
```

I have to write this over and over.

I have to write this over and over.

I have to write this over and over.

I have to write this over and over.

“for” loops are a little different, but still quite similar

- | | |
|--|---|
| <ul style="list-style-type: none">•• <i>for</i> (initialization; condition; follow-up action)• One statement••••• | <ul style="list-style-type: none">•• <i>for</i> (initialization; condition; follow-up action)• One block of statements•• |
|--|---|

Components of a “for” loop

```
for(int x = 0; x < 4; x++)  
{  
    System.out.println("I have to write "  
        + "this over and over.");  
}
```



```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int count;
```

```
    for (count = 1; count <= 500; count++)
```

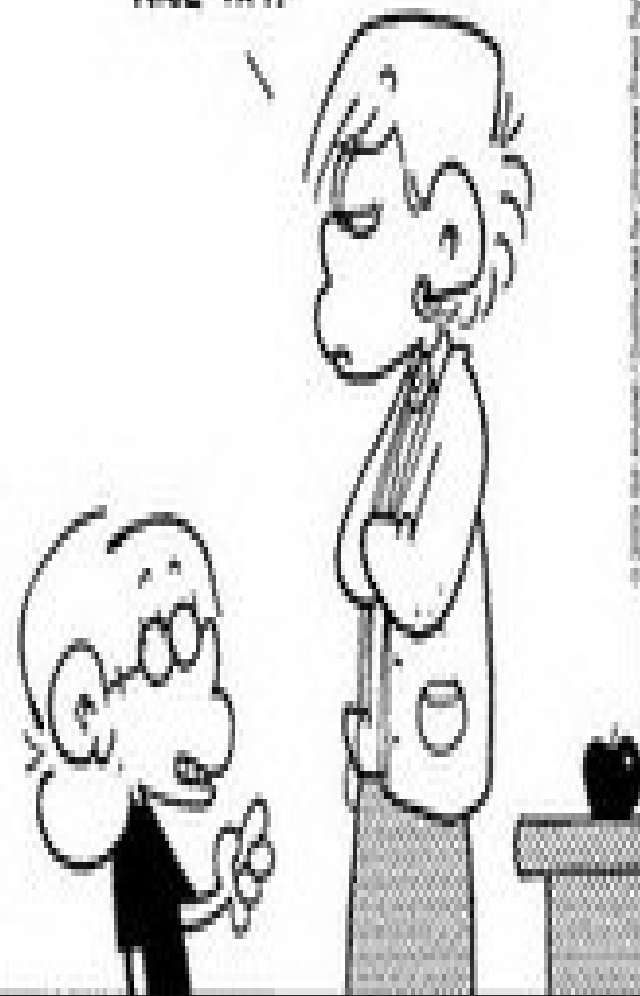
```
        printf("I will not throw paper airplanes in class.");
```

```
    return 0;
```

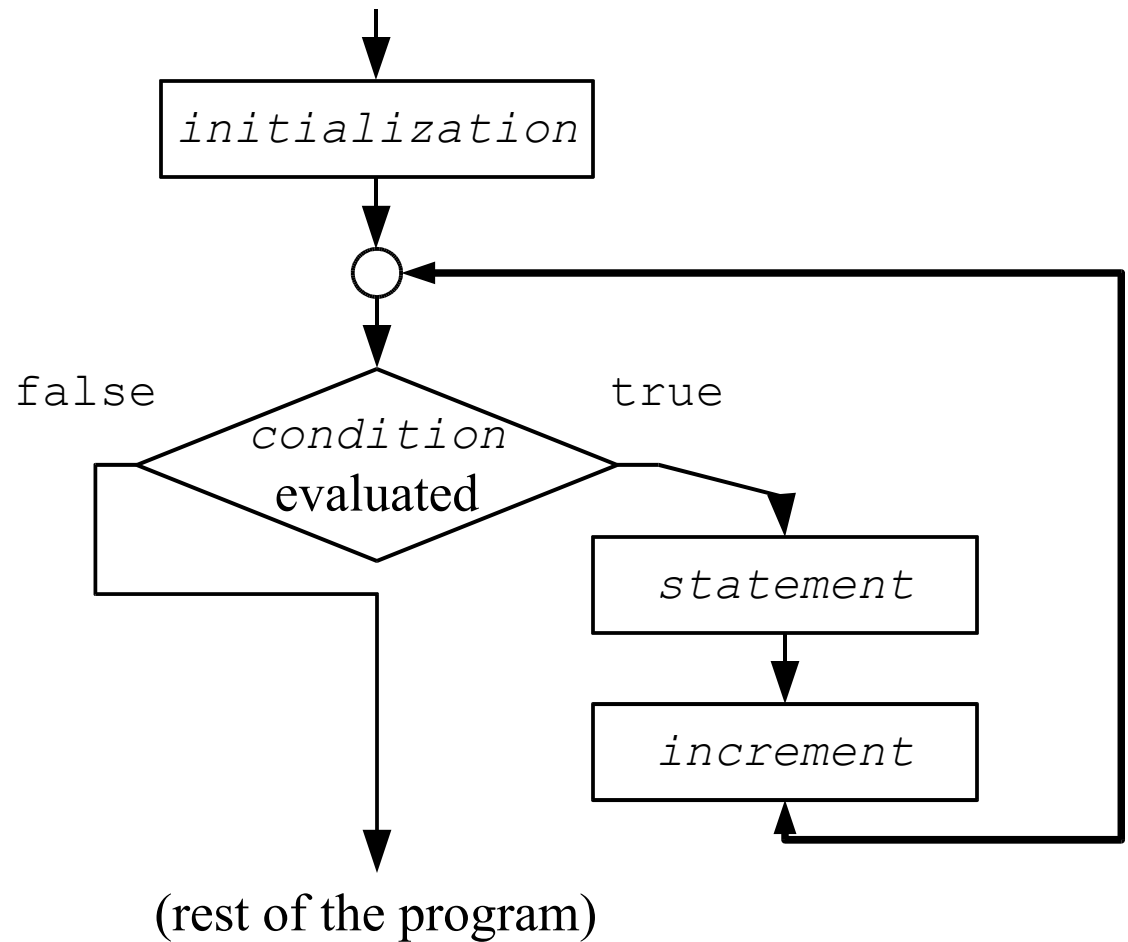
```
}
```

WEEK 11-1

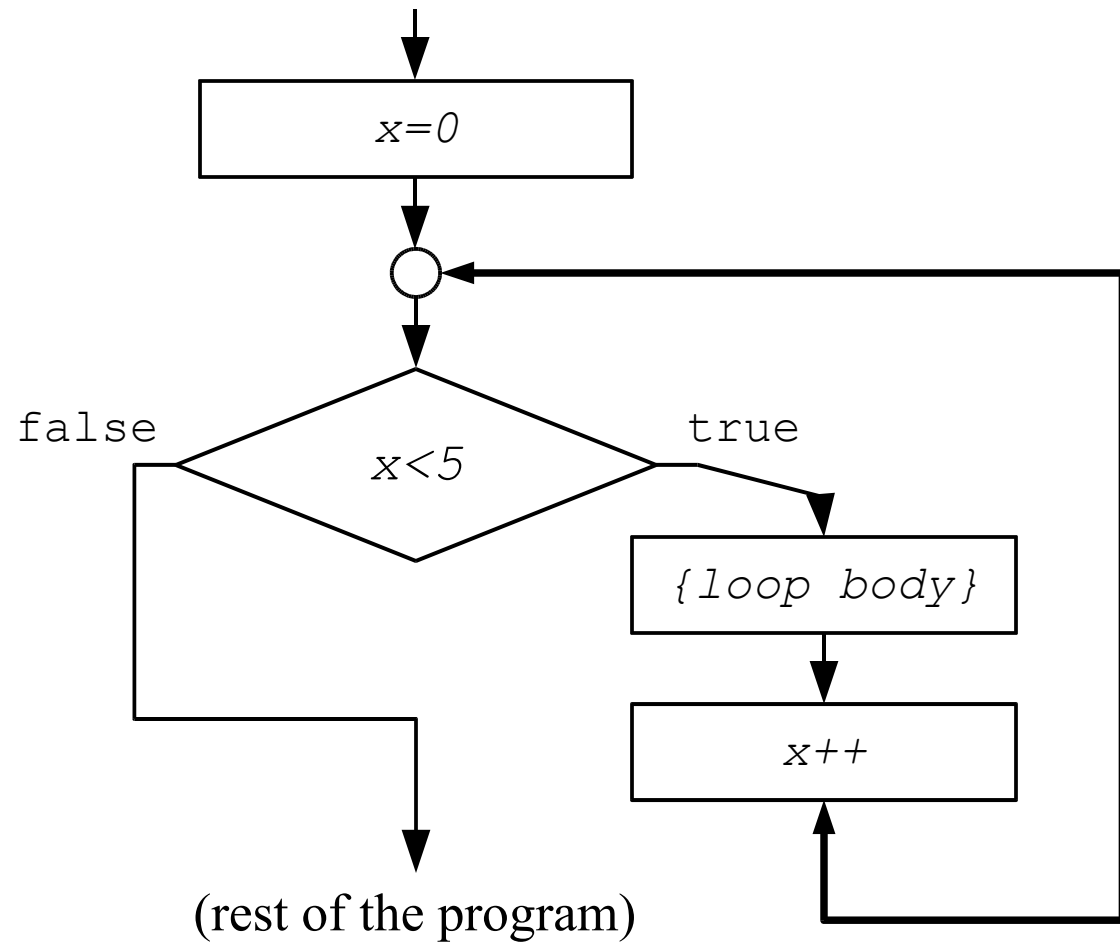
NICE TRY.



Logic of a `for` Statement



Logic of a `for` Statement



for Loops as while Loops

- A `for` loop is equivalent to the following `while` loop structure:

```
initialization;  
while ( condition ) {  
    statement;  
    increment;  
}
```

for Loops as while Loops

- A `for` loop is equivalent to the following `while` loop structure:

```
initialization;  
while ( condition ) {  
    statement;  
    increment;  
}
```

- Excellent exercise to do at home!
- Find a for-loop example and:
- Rewrite it as a while loop
- Rewrite it as a series of if-else statements.

Execution of a `for` Loop

- Like in a `while` loop, the condition of a `for` loop is tested prior to entering the loop body
- If the condition of a `for` loop evaluates to `false` initially (that is, it evaluates to `false` the first time it is evaluated), the statement is never executed
- Therefore, the body of a `for` loop will be executed zero or more times
- A `for` loop is well suited for executing a specific number of times that can be determined in advance

AnotherCounter.java

- `public class AnotherCounter {`
- `public static void main(String[] args) {`
- `final int LIMIT = 3;`
-
- `for (int count=1; count <= LIMIT; count++) {`
- `System.out.println(count);`
- `}`
-
- `System.out.println("Done.");`
- `}`
- `}`

AnotherCounter.java

```
• public class AnotherCounter {  
•     public static void main(String[] args) {  
•         final int LIMIT = 3;  
•  
•         for (int count=1; count <= LIMIT; count++) {  
•             System.out.println(count);  
•         }  
•  
•         System.out.println("Done.");  
•     }  
• }
```

Console:

```
1  
2  
3  
Done.
```

LIMIT: 3

count: ~~1~~ ~~2~~ ~~3~~ 4

What does this display?

What shape does this display?

```
• public class Stars {  
•     public static void main (String[] args) {  
•         final int MAX_ROWS = 10;  
•  
•         for (int row = 1; row <= MAX_ROWS; row++)  
•         {  
•             for (int star = 1; star <= row; star++)  
•             {  
•                 System.out.print("*");  
•             }  
•             System.out.println();  
•         }  
•     }  
• }
```

*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * *

Exercise:

In this exercise, you will write methods to write a small portion of a chess game. You will write methods that list the squares a piece can move to.

- Each method should take as input 1) a char representing the file (column) the piece is on and 2) and int representing the rank (row) the piece is on. Files go from a-h (inclusive). Ranks go from 1-8

Exercise:

Write a class ChessMoves which has the following methods

```
void printRookMoves(char file, int rank)
void printBishopMoves(char file, int rank)
void printQueenMoves(char file, int rank)
void printPawnMoves(char file, int rank)
void printKingMoves(char file, int rank)
void printKnightMoves(char file, int rank)
```

(You may assume that no pieces are in the way or captures can be made)

Ex. printRookMoves .

```
public static void printRookMoves(
    char file, int rank) {
    for (char c = 'A'; c <= 'H'; c++) {
        if (c != file) {
            System.out.println(c + "" + rank);
        }
    }
    for (int r = 1; r <= 8; r++ ) {
        if (r != rank) {
            System.out.println(file + "" + r);
        }
    }
}
//write this with a while loop instead!
```

What Shape does this Display?

```
• public class WhatIsThis {  
•     public static void main(String[] args) {  
•         final int MAX_ROWS = 10;  
•         for (int row = 1; row <= MAX_ROWS; row++) {  
•             for (int space = 1; space <= MAX_ROWS - row; space++) {  
•                 System.out.print(" ");  
•             }  
•             for (int star = 1; star <= row * 2; star++) {  
•                 System.out.print("*");  
•             }  
•             System.out.println();  
•         }  
•         for (int base = 3; base > 0; base--) {  
•             for (int space = 1; space <= MAX_ROWS-1; space++) {  
•                 System.out.print(" ");  
•             }  
•             System.out.println("**");  
•         }  
•     }  
• }  
•
```

**

**

**

**

Details on the `for` Statement

- Each expression in the header of a `for` loop is optional
 - If the *initialization* portion is left out, no initialization is performed
 - If the *condition* portion is left out, it is always considered to evaluate to `true`, and therefore creates an infinite loop
 - If the *increment* portion is left out, no increment operation is performed
- Both semi-colons are always required in the `for` loop header

for Loop: Exercise 1

- Complete the `main()` method of the `Multiples` class by adding code that displays all the multiples of a number entered by the user that exist between 1 and an upper limit also entered by the user
 - The completed program **MUST** display 5 multiples of the number entered by the user per line, except for the last line, which may contain less
 - In addition, the completed program **MUST** display a tab character (`'\t'`) between every multiple displayed on a line
 - Finally, the completed program **MUST** use a `for` loop to generate and display the multiples of the number entered by the user

Multiples.java (1 / 2)

```
• import java.util.Scanner;
•
• public class Multiples {
•     public static void main(String[] args) {
•         Scanner keyboard = new Scanner(System.in);
•         final int PER_LINE = 5;
•         int value;
•         int limit;
•
•         System.out.print ("Enter a positive value: ");
•         value = keyboard.nextInt();
•         System.out.print ("Enter an upper limit: ");
•         limit = keyboard.nextInt();
•
•         System.out.println ("The multiples of " + value +
•             " between " + value + " and " + limit +
•             " (inclusive) are:");
•         // Continued on next slide
```

Multiples.java (2 / 2)

- `// Continued from previous slide`
-
- `// Add your code here`
- `}`
- `}`

for Loop: Exercise 2

- Complete the `main()` method of the `Approximation` class by adding code that approximates the number e (the basis of the natural logarithm). The number e can be approximated by the following sum: $e = (1 / 0!) + (1 / 1!) + (1 / 2!) + (1 / 3!) + \dots$
 - The `main()` method of the `Approximation` class asks the user to enter the number of terms of be computed; your task is to add code which computes each of these terms and adds them together to form the approximation of e
- The obvious way of solving this problem involves using nested loops, where the inner loop computes the required factorial during every iteration of the outer loop. Can you find a way to solve this problem using only one loop?

Approximation.java

```
• import java.util.Scanner;
•
• public class Approximation {
•     public static void main(String[] args) {
•         Scanner keyboard = new Scanner(System.in);
•         int n;
•
•         System.out.print("Please enter the number of terms: ");
•         n = keyboard.nextInt();
•
•         // Add your code here
•     }
• }
```

Stars.java

- `public class Stars {`
- `public static void main (String[] args) {`
- `final int MAX_ROWS = 10;`
-
- `for (int row = 1; row <= MAX_ROWS; row++) {`
- `for (int star = 1; star <= row; star++) {`
- `System.out.print("*");`
- `}`
- `System.out.println();`
- `}`
- `}`
- `}`

What shape does this program display?

WhatIsThis.java (1 / 2)

```
• public class WhatIsThis {  
•     public static void main(String[] args) {  
•         final int MAX_ROWS = 10;  
•  
•         for (int row = 1; row <= MAX_ROWS; row++) {  
•             for (int space = 1; space <= MAX_ROWS - row; space++) {  
•                 System.out.print(" ");  
•             }  
•             for (int star = 1; star <= row * 2; star++) {  
•                 System.out.print("*");  
•             }  
•             System.out.println();  
•         }  
•  
•         for (int base = 3; base > 0; base--) {  
•             for (int space = 1; space <= MAX_ROWS-1; space++) {  
•                 System.out.print(" ");  
•             }  
•             // Continued on next slide
```

WhatIsThis.java (2 / 2)

- `// Continued from previous slide`
- `System.out.println("**");`
- `}`
- `}`
- `}`

What shape does this program display?

Part 4: Exercises

Exercises (1)

1. Write a program which consists of a single class called `Factor` that asks the user to enter a positive integer (including zero). The program then displays that number and its greatest prime factor. The program repetitively does this until the user inputs a negative number.

Exercises (2)

2. Write a program which consists of single class called `Boxes` that asks the user for a positive integer number n . The program then displays a solid square with side length n next to a hollow square with side length n . The program does nothing if the user inputs 0 or a negative value. If example, if the user enters 4, the following will be displayed:

```
* * * * * * * * * *
* * * * * *      *
* * * * * *      *
* * * * * * * * * *
```

Exercises (3)

3. One can approximate the square root of any number a using the following sequence:

$$x_0 = a / 2$$

$$x_{i+1} = (x_i + a / x_i) / 2$$

Write a program which consists of a single class called `SquareRoot`. This class defines a `main()` method that asks the user to enter a number a , and a number of iterations n , and reads these values from the keyboard. The program then computes the n^{th} term in the above sequence, thus approximating the value of the square root of a , and displays it to the screen.

break and continue

If you write “break” in the middle of a loop, the loop will immediately terminate.

If you write “continue” in the middle of a loop, the current step of the loop will finish but the loop will continue afterwards.

break and continue

```
int i=0;
while (i < 10 ) {
    if (i == 5) break; // skips the rest of the loop
    i++;
}
```

```
for (int j =0; j < 10; j++) {
    if (j== 5) {
        continue;
    }
}
```

Example: Writing a program to store grades

Suppose we wanted to write a computer program to store the grades of every comp 202 student on every different assignment.

Let's say we are going to read all of this from the keyboard.

We could make lots of variables and a scheme where we numbered students from 1-300 and then used a _ to separate which assignment/midterm/final it was.

i.e. `int student_1_f` could store the first students final grade.

Example: Writing a program to store grades

```
Scanner s = new Scanner(System.in);
```

```
int student_1_0, student_1_1, student_1_m, student_1_f;  
int student_2_0, student_2_1, student_1_m, student_2_f;  
int student_3_0, student_3_1, student_1_m, student_3_f;  
int student_4_0, student_4_1, student_1_m, student_4_f;  
int student_5_0, student_5_1, student_1_m, student_5_f;  
int student_6_0, student_6_1, student_1_m, student_6_f;  
int student_7_0, student_7_1, student_1_m, student_7_f;  
int student_8_0, student_8_1, student_1_m, student_8_f;  
int student_9_0, student_9_1, student_1_m, student_9_f;
```


Example: Writing a program to store grades

```
Scanner s = new Scanner(System.in);
```

```
student_1_0 = s.nextInt();  
student_1_1 = s.nextInt();  
student_1_m = s.nextInt();  
student_1_f = s.nextInt();  
student_2_0 = s.nextInt();  
student_2_1 = s.nextInt();  
student_2_2 = s.nextInt();  
student_2_3 = s.nextInt();
```

Once we go through all this trouble to enter the grades, we still have to work with the numbers!

One idea would be if we could do some sort of for loop.

For example:

```
for (int i=0; i<300; i++) {
```

```
    System.out.println("The grade for student " + i + " on assignment 1 is " +  
        student_i_1
```

```
}
```

However, Java does not allow us to write variables inside our variable names. We can, however, do something pretty similar.

COMP-202

Unit 6: Arrays

CONTENTS:

Array Usage

Multi-Dimensional Arrays

Reference Types

A variable of type *int*.

int number = 5

In memory:

5

A variable of type *int*.

int number = 5

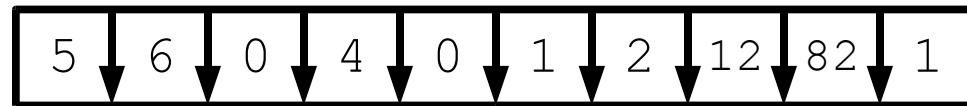
In memory:



An **integer array** corresponds to variable of type *int*[].

`int[] weights = {5, 6, 0, 4, 0, 1, 2, 12, 82, 1}`

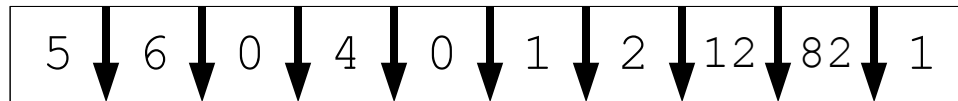
In memory:



An *array* is a fixed-size, ordered collection of elements of the same type.

```
int[ ] weights = {5,6,0,4,0,1,2,12,82,1}
```

In memory:

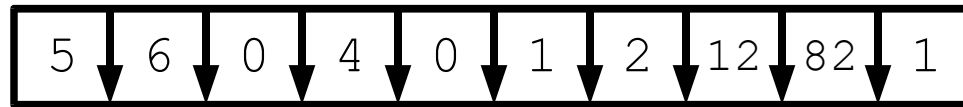


Why use arrays?

They make large amounts of data easier to handle.

```
int[] weights = {5,6,0,4,0,1,2,12,82,1}
```

In memory:



Each cell in the array has an **index**.

e.g. The cell that contains 82 has index 8.

The cell that contains 5 has index 0.

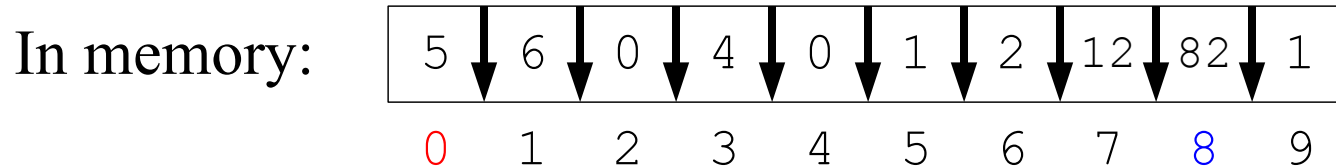
In memory:

5	↓	6	↓	0	↓	4	↓	0	↓	1	↓	2	↓	12	↓	82	↓	1
0		1		2		3		4		5		6		7		8		9

Each cell in the array has an **index**.

e.g. The cell that contains 82 has index 8.

The cell that contains 5 has index 0.



So you can write:

```
int j = weights[8] + weights[0];
```

Part 1: Array Basics

Array Declaration Examples

- `double[] prices;`

- Declares an array called `prices`

- Each element in this array is a `double`; variable `prices` is of type `double[]`

- `char[] code;`

- Declares an array called `code`

- Each element of this array is a `char`; variable `code` is of type `char[]`

- `String[] names;`

- Declares an array called `names`

- Each element of this array is a `String`; variable `names` is of type `String[]`

Assigning values to an array

If you know ahead of time how many numbers you want to store (and their values) you can assign values to an array when you declare it:

```
int[] someNumbers = {1,2,3,4,5};
```

Assigning values to an array

If you do not know ahead of time how many numbers you want to store (or don't know their values), you have to assign the values in 2 phases:

- 1) Tell the computer how many values you want to store
- 2) Set these values

Setting the size of an array

To specify how large an array should be, you do the following

```
sometype[] myArray;  
//declare an array of type sometype  
...  
  
myArray = new sometype[size];
```

Accessing elements of an array

To get or set values in an array, you will always use both the array name, and the index of the value you want.

You can think of the index like a subscript.

Accessing elements of an array

Array indices start from 0

```
//set x to be first value in  
//array
```

```
int x = myArray[0];
```

```
//set 3rd value in myArray
```

```
myArray[2] = 10;
```


What does this display?

```
public class FirstArray {  
    public static void main(String[]  
args){  
        String[] names = {"Jordan",  
"Jesse", "Joshua"};  
        for(int i = 1; i >= -1; i = i - 1)  
            System.out.println(names[i+1]);  
    }  
}
```

What does this display?

```
public class FirstArray {  
    public static void main(String[]  
args){  
        String[] names = {"Jordan",  
"Jesse", "Joshua"};  
        for(int i = 1; i >= -1; i = i - 1)  
            System.out.println(names[i]);  
    }  
}
```

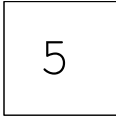
Review: how primitive types are stored in memory

Example:

```
int number = 5
```

In memory:

`number`



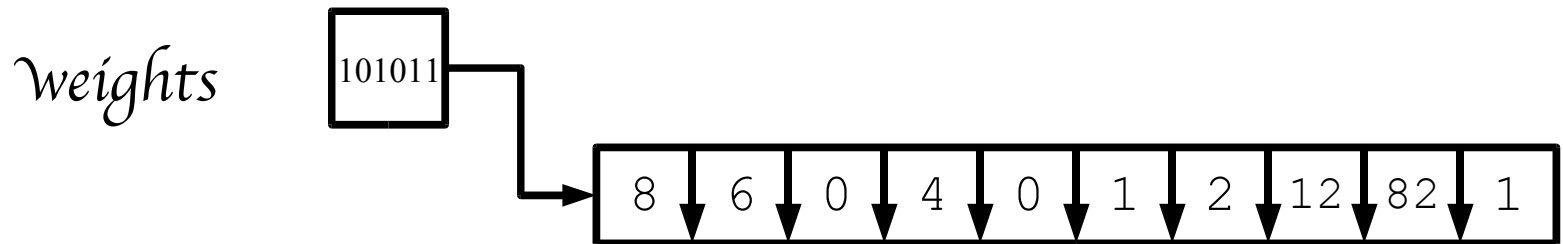
5

`number` represents a location in memory where the integer 5 is stored

Array types are NOT primitive types

Example:

```
int[] weights = {8,6,0,4,0,1,2,12,82,1}
```



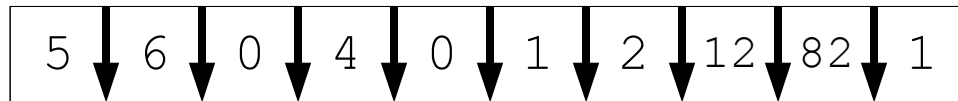
weights represents a location in memory where the **address** of the first array cell is stored.

Primitive vs. reference types

- Primitive types:
 - The variable represents the location in memory at which an **actual value**, like the integer 175.
- Reference types:
 - The variable represents the location in memory at which **another memory address** (or “**reference**”) is stored.

```
int[] weights = {5,6,0,4,0,1,2,12,82,1}
```

In memory:



Initializer Lists

```
int[] numbers = {2, 3, 5};
```

- The above statement does all the following in one step:
 - It declares a variable of type `int[]` called `numbers`
 - It creates an array which contains 3 elements
 - It stores the address in memory of the new array in variable `numbers`
 - It sets the value of first element of the array to 2, the value of the second element of the array to 3, and the value of the last element of the array to be 5
- Often, these steps are carried out separately.

Array types are reference types

- The declaration

```
int[ ] numberArray;
```

creates a reference variable, which holds a *reference* to an `int[ ]`

- **No array is created yet, just a reference to an array.**

Reference vs Primitive in methods

-When you call a method with a *primitive* type, remember that you are only passing to the method the *value* of the variable.

Reference vs Primitive in methods

```
public static void badSwap(int a, int b) {  
    int temp = a;  
    a = b;  
    b = temp;  
}
```

If I call this method `badSwap(x,y)`, it will not swap the two variables. The method `badSwap` only knows the *values* of `x` and `y`

Reference types

- When you call a method with input of *reference* types, we still pass the value of the variable, but the value now represents an *address* in memory.
- If I swap the *address* inside the method, nothing will change outside.
- But if I swap the *contents* at the address, it will be “permanent”

Arrays as reference types

For example, if I want to swap two arrays in a method, I have to swap the *contents* of the arrays.

The array addresses will still be the same, but each array would now store what used to be in the other.

```
{
```

```
....
```

```
int[] array1 = {1,2,3,4,5};
```

```
int[] array2 = {6,7,8,9,10};
```

```
badSwap(array1,array2)
```

```
....
```

```
}
```

```
public static void badSwap(  
    int[] a1, int[] a2) {  
    int[] temp = a1;  
    a1 = a2;  
    a2 = temp;  
}
```

This swaps a1 and a2 indeed, but the change will not matter in the calling function

You can figure out how many elements are in an array by writing

arrayname.length

```
public static void goodSwap(int[] array1,  
    int[] array2) {  
    int temp;  
  
    for (int i=0; i < array1.length;i++) {  
        temp = array1[i];  
        array1[i] = array2[i];  
        array2[i] = temp;  
    }  
}
```


Array Allocation (1)

```
final int SIZE = 5;
```

```
➔ int[] numbers;
```

```
numbers = new int[2 * SIZE];
```

numbers



SIZE

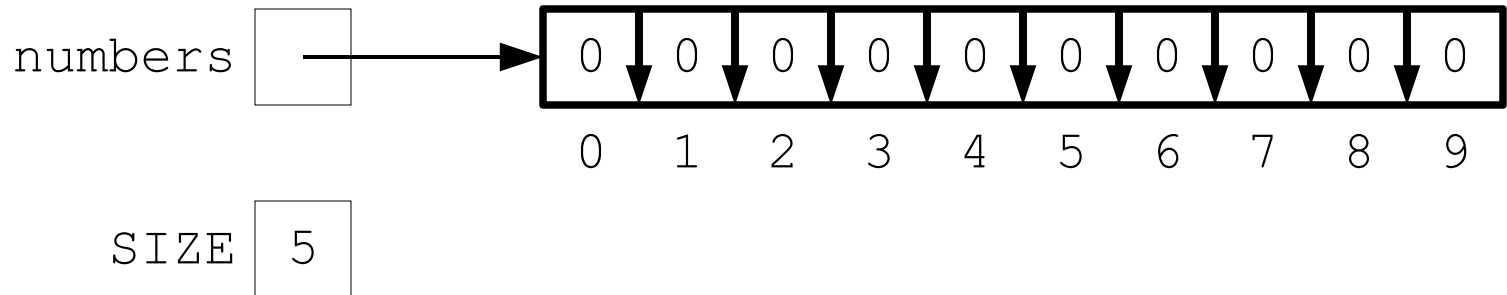


Array Allocation (1)

```
final int SIZE = 5;
```

```
int[] numbers;
```

```
➔ numbers = new int[2 * SIZE];
```



Allocating Arrays (2)

```
variableName = new type [ size ] ;
```

↑
The variable in which
the location in memory
of the newly created
array will be stored

new is a reserved word
in Java

↑
An expression which
specifies the number
of elements in the
array

↑
The type of the
elements to be stored
in the array

Allocating Arrays (3)

- Once an array has been created, its size cannot be changed
- As with regular variables, the array declaration and the array allocation operation can be combined:

type[] *variableName* = *new type*[*size*];

Initializer Lists

```
int[] numbers = {2, 3, 5};
```

is equivalent to the following code fragment:

```
int[] numbers = new int[3];  
numbers[0] = 2;  
numbers[1] = 3;  
numbers[2] = 5;
```

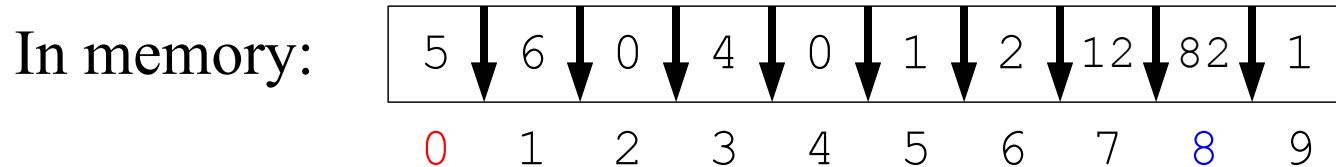
Exercise on reference types: what does this display?

```
public class ArrayCopy {  
    public static void main(String[] args){  
        int[] numbers = {1, 2, 3};  
        int[] differentNumbers = new int[3];  
        differentNumbers = numbers;  
        numbers[1] = 2;  
        differentNumbers[1] = 3;  
        System.out.println(numbers[1]);  
        System.out.println(differentNumbers[1]);  
    }  
}
```

Each cell in the array has an **index**.

e.g. The cell that contains 82 has index 8.

The cell that contains 5 has index 0.



So you can write:

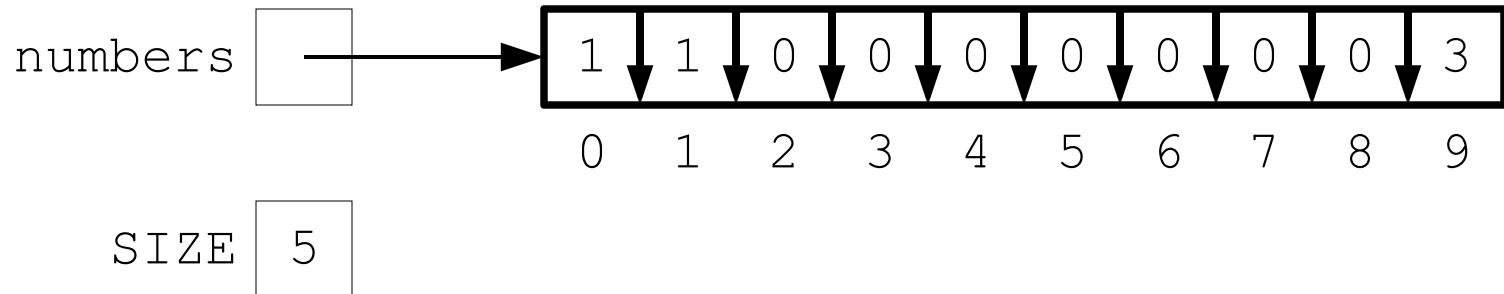
```
int j = weights[8] + weights[0];
```

Array Access Example

```
numbers[0] = 1;
```

```
numbers[1] = numbers[0];
```

```
numbers[2 * SIZE - 1] = 2 * numbers[0] + 1;
```



Array Length

```
int[] weights = {5,6,0,4,0,1,2,12,82,1}  
total = weights.length;
```

We can get the length of any array
with `[arrayName].length`

Here, `total` is assigned the value 10.

Array Length

```
int[] weights = {5,6,0,4,0,1,2,12,82,7}  
total = weights.length;
```

What is the index of the cell containing **7**
in terms of `weights.length`?

Array Length

```
int[] weights = {5,6,0,4,0,1,2,12,82,7}  
total = weights.length;
```

What is the index of the cell containing **7**
in terms of `weights.length`?

Answer: **`weights.length - 1`**

The length of an array is a constant

```
int[] weights = {5,6,0,4,0,1,2,12,82,1}  
weights.length = 2; // illegal!
```

The `length` field of an array can be used like any other *final* variable of type `int`.

Fill in the method `init` so that it initializes the array `list` with a decreasing list of integers starting at `start`.

e.g. If `myArray` has length 5, a call to `init(myArray, 20)` will assign the following values to an array: `{20, 19, 18, 17, 16}`.

```
public static void init(int[] list, int start){  
    // use a loop to assign a value to each cell  
}
```

Bounds Checking (1)

```
int[] myArray = new int[5];  
myArray[5] = 42;  
    // Array myArray contains 5 elements, so  
    // the valid indices for myArray are 0-4  
    // inclusive  
    // Therefore, the program crashes
```

- When this occurs, the error message will mention that the program threw an *ArrayIndexOutOfBoundsException*
 - The error message should also mention which line in your program caused the latter to crash

IndexOutOfBoundsDemo.java

```
1  public class IndexOutOfBoundsDemo {  
2      public static void main(String[] args) {  
3          final int SIZE = 5;  
4          int[] myArray;  
5  
6          myArray = new int[SIZE];  
7  
8          System.out.println("Attempting to retrieve the " +  
9              "element at position " + SIZE + " of an array of " +  
10             "size " + SIZE);  
11          myArray[SIZE] = 42;  
12          System.out.println("The element at position " + SIZE +  
13             " of this array is " + myArray[SIZE]);  
14      }  
15  }
```



Reading Exception Output (1)

- The program's output:

```
Attempting to retrieve the element at position 5 of an array of  
size 5
```

```
Exception in thread "main" java.lang.
```

```
ArrayIndexOutOfBoundsException: 5
```

```
    at IndexOutOfBoundsDemo.main(IndexOutOfBoundsDemo.java:11)
```

Method where the problem occurred

File where the problem occurred

Line number where the problem occurred

Nature of the problem and additional information

- Index we tried to access and caused the crash

Off-By-One Errors Revisited

- Off-by-one errors are common when using arrays:

```
int[] array = new int[100];  
int i;  
  
i = 0;  
while (i <= 100) {  
    array[i] = 2 * i;  
    i = i + 1;  
}
```