Scope in Loop

```
// Demonstrate lifetime of a variable.
class VarInitDemo {
  public static void main(String args[]) {
    int x;
    for(x = 0; x < 3; x++) {
      int y = -1; // y is initialized each time block is entered
      System.out.println("y is: " + y); // this always prints -1
      y = 100;
      System.out.println("y is now: " + y);
    }
  }
}
```

Scope in Loop

```
// Declare loop control variable inside the for.
class ForVar {
  public static void main(String args[]) {
    int sum = 0;
    int fact = 1;

    // compute the factorial of the numbers through 5
    for(int i = 1; i <= 5; i++) {
      sum += i; // i is known throughout the loop
      fact *= i;
    }

    // but, i is not known here

    System.out.println("Sum is " + sum);
    System.out.println("Factorial is " + fact);
  }
}
```

The variable **i** is declared inside the **for** statement.

Scope in Loop

- What's wrong here?

```
public static void main(String[] args) throws IOException {
    for (int x = 100; x > -100; x-=5) {
        System.out.println(x);
        x-=5;
    }
    System.out.println(x);
}
```

Multiple Loop Control Variables

```
// Use commas in a for statement.
class Comma {
  public static void main(String args[]) {
    int i, j;

    for(i=0, j=10; i < j; i++, j--)
      System.out.println("i and j: " + i + " " + j);
  }
}
```

Notice the two loop control variables.

For Loop Missing Expressions

```
// Parts of the for can be empty.
class Empty {
  public static void main(String args[]) {
    int i;

    for(i = 0; i < 10; ) {
      System.out.println("Pass #" + i);
      i++; // increment loop control var
    }
  }
}
```

The iteration expression is missing.

For Loop Missing Expressions

```
// Move more out of the for loop.
class Empty2 {
  public static void main(String args[]) {
    int i;

    i = 0; // move initialization out of loop
    for(; i < 10; ) {
      System.out.println("Pass #" + i);
      i++; // increment loop control var
    }
  }
}
```

The initialization expression is moved out of the loop.

Infinite Loops

```
for(;;) // intentionally infinite loop
{
  //...
}
```

```
while (true) {
    //statements
}
```

Loop with no body

```
// The body of a loop can be empty.
class Empty3 {
  public static void main(String args[]) {
    int i;
    int sum = 0;

    // sum the numbers through 5
    for(i = 1; i <= 5; sum += i++) ;  ← No body in this loop!

    System.out.println("Sum is " + sum);
  }
}
```

Branching Statements

- break terminates a for or while loop

```
for (int i=0; i<100; i++) {
      if(i == 50)
            break;
      System.out.println( “Rule #”  + i);
}
```

Break Infinite Loop

```
// Read input until a q is received.
class Break2 {
  public static void main(String args[])
    throws java.io.IOException {

    char ch;

    for( ; ; ) {
      ch = (char) System.in.read(); // get a char
      if(ch == 'q') break;
    }
    System.out.println("You pressed q!");
  }
}
```

This "infinite" loop is terminated by the **break**.

Branching Statements

- continue skips the current iteration of a loop and proceeds directly to the next iteration

```
for (int i=0; i<100; i++) {
        if(i == 50)
                continue;
        System.out.println( “Rule #”  + i);
}
```

Branching Statements

- The return statement exits from the current method, and control flow returns to where the method was invoked.

```
return count;
```

- The data type of the returned value must match the type of the method's declared return value.

```
return; //when a method declared void
```

Nested loops

```
for (int i = 0; i < 3; i++) {
   for (int j = 2; j < 5; j++) {
        System.out.println (i + “, “  + j);
   }
}
```

Break with Nested Loops

```
for (int i = 0; i < 3; i++) {
        for (int j = 2; j < 5; j++) {
                if (j ==4)
                        break; //only breaks the inner
                System.out.println (i + "," + j);
        }
}
```

Labeled break

- Block that will be broken is labeled and use this label as the target of a break statement

```
outer: for (int i = 0; i < 3; i++) {
        for (int j = 2; j < 5; j++) {
            if (j ==4)
                break outer;
            System.out.println (i + "," + j);
        }
    }
```

Methods

```
public static void main(String[] arguments)
{
    System.out.println("hi");
}
```

Adding Methods

public static void ***NAME***() {

STATEMENTS

}

To call a method:

```
NAME();
```

Calling Methods

```
public class NewLine {
    public static void newLine() {
        System.out.println("");
    }
    public static void threeLines() {
        newLine();
        newLine();
        newLine();
    }
    public static void main(String[] args){
        System.out.println("Line 1");
        threeLines();
        System.out.println("Line 2");
    }
}
```

Parameters

```
public static void NAME(TYPE NAME) {
    STATEMENTS
}
```

To call:

```
NAME(EXPRESSION);
```

Parameters

```
public class Square {
    public static void printSquare(int x) {
        System.out.println(x*x);
    }
    public static void main(String[] args){
        int value = 2;
        printSquare(value);
        printSquare(3);
        printSquare(value*2);
    }
}
```

Parameters

```
public class Square {
    public static void printSquare(int x) {
        x=x*x;
        System.out.println(x);
    }
    public static void main(String[] args){
        int value = 2;
        int x=5;
        printSquare(value);
        printSquare(x);
        sout(value); sout(x);
        printSquare(3);
        printSquare(value*2);
    }
}
```

What's wrong here?

```
public class Square2 {
   public static void printSquare(int x) {
        System.out.println(x*x);
   }
   public static void main(String[] args) {
        printSquare(5.5);
   }
}
```

What's wrong here?

```
public class Square3 {
   public static void printSquare(double x) {
        System.out.println(x*x);
   }
   public static void main(String[] args) {
        printSquare("hello");
   }
}
```

Multiple Parameters

[…] ***NAME***(***TYPE NAME, TYPE NAME***) {

STATEMENTS

}

To call:

```
NAME(arg1, arg2);
```

Multiple Parameters

```
public class Multiply {
   public static void times (double a, double b){
        System.out.println(a * b);
   }
   public static void main(String[] args){
        times (2, 2);
        times(3, 4);
   }
}
```

Return Values

```
public static TYPE NAME() {
    STATEMENTS
    return EXPRESSION;
}
```

`void` means “no type”

Return Values

```
public class Square3 {
   public static void printSquare(double x) {
        System.out.println(x*x);
   }
   public static void main(String[] args) {
        printSquare(5);
   }
}
```

Return Values

```
public class Square4 {
   public static double square(double x) {
        return x*x;
   }
   public static void main(String[] args) {
        System.out.println(square(5));
   }
}
```

What's wrong here?

```
public class DivisionDemo {

    public static void divide(int operand1, int operand2) {
        if (operand2 != 0)
            System.out.println(operand2 + "is not zero");
            System.out.println("Result = " + operand1 / operand2);

    }


    public static void main(String[] args) {
        divide(9,0);

    }
```

else

```
public class DivisionDemo {

    public static void divide(int operand1, int operand2) {
        if (operand2 != 0) {
            System.out.println(operand2 + "is not zero");
            System.out.println("Result = " + operand1 / operand2);
        }else {
            System.out.println("Can not divide by zero!");
        }

    }


    public static void main(String[] args) {
        divide(9,0);

    }

}
```

Recursion

Recursion

- A method of defining a function in terms of its own definition
- Example: the Fibonacci numbers

  $f(n) = f(n-1) + f(n-2)$

  $f(0) = f(1) = 1$ <<<<< Base Case

- In programming recursion is a method call to the same method. In other words, a recursive method is one that calls itself.

Recursion

- To solve a problem recursively
  - break into smaller problems
  - solve sub-problems recursively
  - assemble sub-solutions

- Write a function that computes the sum of numbers from 1 to n

  int sum (int n)

  1. use a loop
  2. recursively

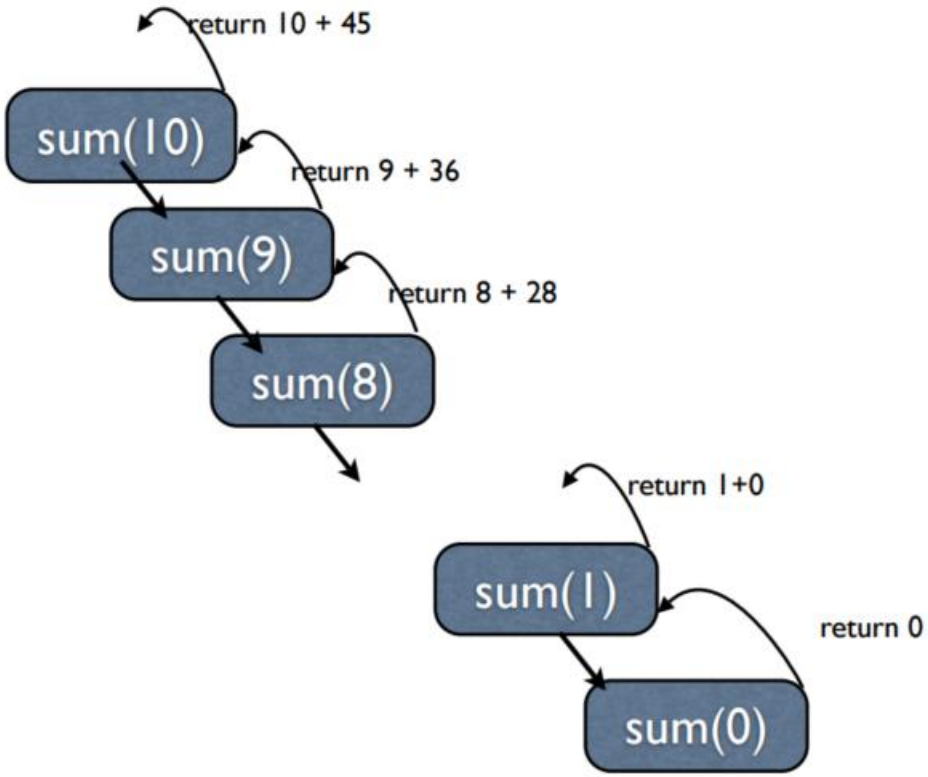
Recursion

```
//with a loop
int sum (int n) {
    int s = 0;
    for (int i=0; i<n; i++)
        s+= i;
    return s;
}
```

```
//recursively
int sum (int n) {
    int s;
    if (n == 0) return 0;
     //else
     s = n + sum(n-1);
     return s;
}
```

How does it work?

Recursion


return 10 + 45
sum(10)
return 9 + 36
sum(9)
return 8 + 28
sum(8)
return 1+0
sum(1)
return 0
sum(0)

Recursion

- Factorial n!

```
public static int factorial(int n) { // iterative solution
	int f = 1;
	int i = 0;
	while (i < n) {
		i = i + 1;
		f = f*i;
	}
	return f;
}
```

Recursion

- Definition of factorial:

$$n! = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1$$

- Recursive definition:

$$n! = \begin{cases} n \cdot (n-1)!, & n > 0 \\ 1, & n = 0 \end{cases}$$

Recursion

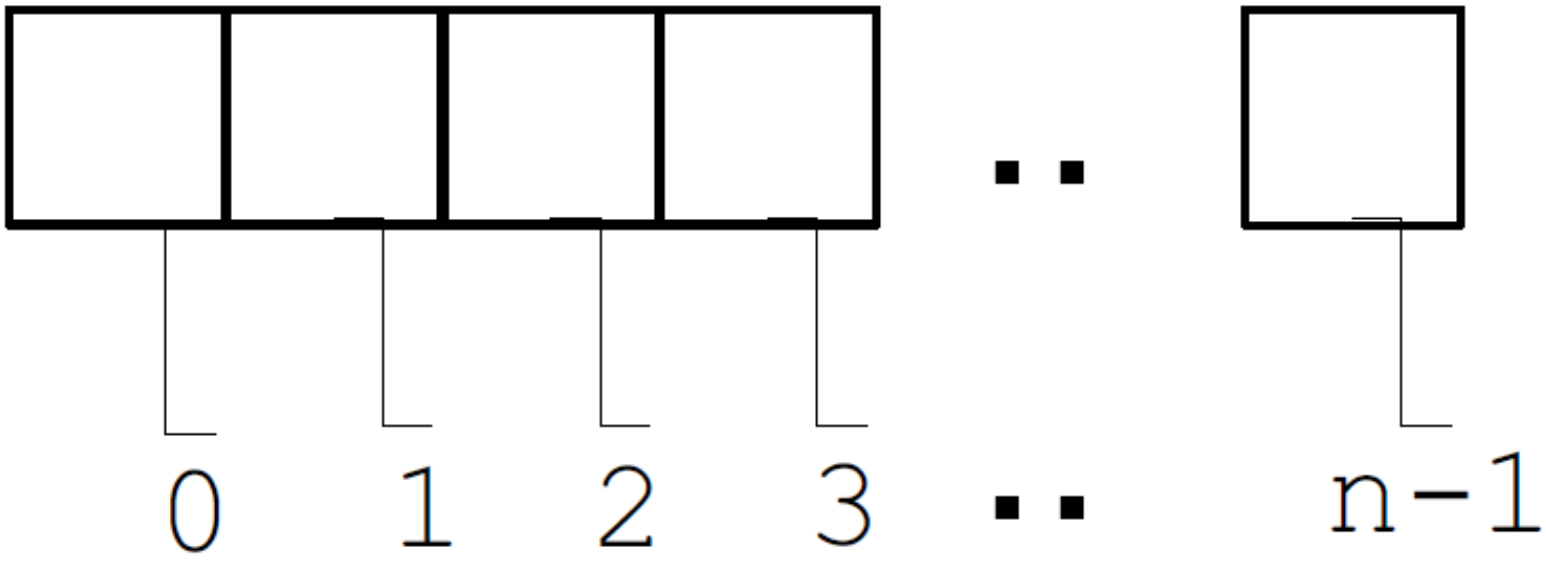
```
public static int factorial(int n) {
    if (n == 0)
        return 1;
    else{
        return n * factorial(n - 1);
    }
}
```

Arrays

Arrays

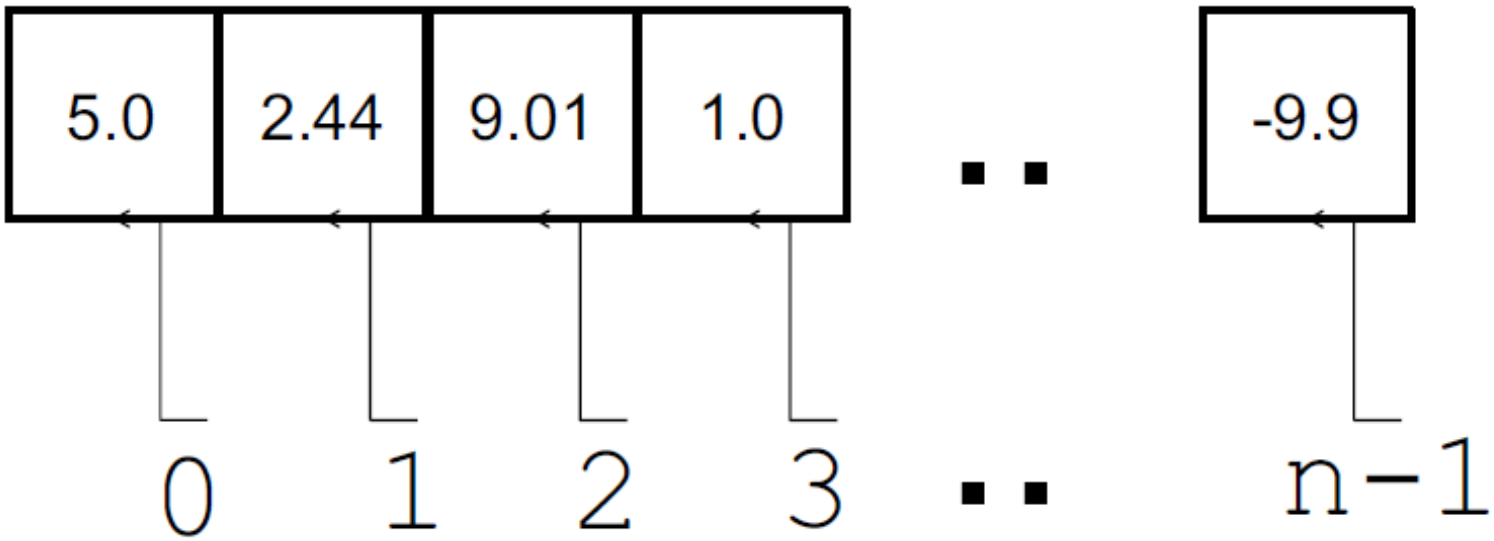
- An array is an indexed list of values.
- You can make an array of any type int, double, String, etc..
- All elements of an array must have the same type.

Arrays



Arrays

Example: double [ ]



Declaring a Variable to Refer to an Array

- The preceding program declares an array (named anArray) with the following line of code:

```
// declares an array of integers
int[] anArray;
```

Declaring a Variable to Refer to an Array

```
double[] anArrayOfDoubles;

boolean[] anArrayOfBooleans;

String[] anArrayOfStrings;
```

Creating, Initializing, and Accessing an Array

- One way to create an array is with the new operator.

```
anArray = new int[10];
```

- The above statement allocates an array with enough memory for 10 integer elements and assigns the array to the anArray variable.

Creating, Initializing, and Accessing an Array

To create an array of a given size, use the operator `new` :

```
int[] values = new int[5];
```

or you may use a variable to specify the size:

```
int size = 12;
int[] values = new int[size];
```

Creating, Initializing, and Accessing an Array

The index starts at zero and ends at length-1.

Example:

```
int[] values = new int[5];
values[0] = 12; // CORRECT
values[4] = 12; // CORRECT
values[5] = 12; // WRONG!! compiles but
                // throws an Exception
                // at run-time
```

Creating, Initializing, and Accessing an Array

Curly braces can be used to initialize an array.

It can ONLY be used when you declare the variable.

```
int[] values  = { 12, 24, -23, 47 };
```

Question?

- Is there an error in this code?

```
int[] values  = {1, 2.5, 3, 3.5, 4};
```

Accessing Arrays

To access the elements of an array, use the `[]` operator:

```
values[index]
```

Example:

```
int[] values  = { 12, 24, -23, 47 };
values[3] = 18;          // {12,24,-23,18}
int x = values[1] + 3; // {12,24,-23,18}
```

The length variable

Each array has a `length` variable built-in that contains the length of the array.

```
int[] values =  new int[12];
int size =      values.length; // 12

int[] values2 = {1,2,3,4,5}
int size2 =     values2.length; // 5
```

Arrays as parameters

The main method accepts a single argument: an array of elements of type String.

```
public static void main (String[] arguments){
    System.out.println(arguments.length);
    System.out.println(arguments[0]);
    System.out.println(arguments[1]);
}
```

Command-line argument

- Command-line arguments let users affect the operation of the application without recompiling it.

```
java MyApp arg1 arg2
```

Parsing Numeric Command-Line Arguments

```
int firstArg;
if (args.length > 0) {

        firstArg = Integer.parseInt(args[0]);
}
```

Multidimensional Arrays

An array is defined using TYPE [ ] .

Arrays are just another type.

```
int[]    values;    // array of int

int[][] values;    // int[] is a type
```

Multidimensional Arrays

- You can also declare an array of arrays by using two or more sets of brackets, such as String[][] names.

```
String[][] names = {
        {"Mr. ", "Mrs. ", "Ms. "},
        {"Smith", "Jones"}
};

// Mr. Smith
System.out.println(names[0][0] + names[1][0]);

// Ms. Jones
System.out.println(names[0][2] + names[1][1]);
```

Combining Loops and Arrays

Looping through an array

- How would you print the values of an array?

```
int[] values  = { 12, 24, -23, 47 };
```

Looping through an array

int[] values = { 12, 24, -23, 47 };

```
for (int index = 0; index <values.length; index++){

  System.out.println(values[index])

}
```

```
// Find the minimum and maximum values in an array.
class MinMax {
  public static void main(String args[]) {
    int nums[] = new int[10];
    int min, max;

    nums[0] = 99;
    nums[1] = -10;
    nums[2] = 100123;
    nums[3] = 18;
    nums[4] = -978;
    nums[5] = 5623;
    nums[6] = 463;
    nums[7] = -9;
    nums[8] = 287;
    nums[9] = 49;
    min = max = nums[0];
    for(int i=1; i < 10; i++) {
      if(nums[i] < min) min = nums[i];
      if(nums[i] > max) max = nums[i];
    }
    System.out.println("min and max: " + min + " " + max);
  }
}
```

Using array initializer in previous Code

```
// Use array initializers.
class MinMax2 {
  public static void main(String args[]) {
    int nums[] = { 99, -10, 100123, 18, -978,
                   5623, 463, -9, 287, 49 };
    int min, max;

    min = max = nums[0];
    for(int i=1; i < 10; i++) {
      if(nums[i] < min) min = nums[i];
      if(nums[i] > max) max = nums[i];
    }
    System.out.println("Min and max: " + min + " " + max);
  }
}
```

Array initializers

Array with more dimensions

type name[ ][ ]...[ ] = new type[size1][size2]...[sizeN];

following declaration creates a 4 × 10 × 3 three-dimensional integer array.

int multidim[ ][ ][ ] = new int[4][10][3];

Initializing Multidimensional Arrays

type-specifier array_name[ ] [ ]
{ *val, val, val, ..., val* },
{ *val, val, val, ..., val* },
.
.
.
{ *val, val, val, ..., val* }
};

```
// Initialize a two-dimensional array.
class Squares {
  public static void main(String args[]) {
    int sqrs[][] = {
      { 1, 1 },
      { 2, 4 },
      { 3, 9 },
      { 4, 16 },
      { 5, 25 },
      { 6, 36 },
      { 7, 49 },
      { 8, 64 },
      { 9, 81 },
      { 10, 100 }
    };
    int i, j;

    for(i=0; i < 10; i++) {
      for(j=0; j < 2; j++)
        System.out.print(sqrs[i][j] + " ");
      System.out.println();
    }
  }
}
```

Notice how each row has its own set of initializers.

References

- http://math.hws.edu/javanotes/
- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/lecture-notes/
- https://www.tiobe.com/tiobe-index/
- Head First Java 2nd Edition
- Java a Beginner's guide 7th Edition