CENG 1004
Introduction to Object Oriented Programming

Spring 2018

Özgür Kılıç
Office: E1-03
Office hours: Mon 13:30-15:20
email: ozgur.kilic10@gmail.com

Course Web Page:
piazza.com/mu.edu.tr/spring2018/ceng1004/home

Sign Up Link
piazza.com/mu.edu.tr/spring2018/ceng1004

if statement

```
if (CONDITION) {
   STATEMENTS
}
```

if statement

```
public static void test(int x){
    if (x > 5){
        System.out.println(x + " is > 5");
    }
}
public static void main(String[] args){
    test(6);
    test(5);
    test(4);
}
```

Comparison operators

x > y: x is greater than y

x < y: x is less than y

x >= y: x is greater than or equal to x

x <= y: x is less than or equal to y

x == y: x equals y

(equality: ==, assignment: =)

Boolean operators

&&: logical AND
||: logical OR

```
if (x > 6) {
    if (x < 9) {
            …
    }
}
```

```
if ( x > 6 && x < 9) {
...
}
```

else

```
if (CONDITION) {
   STATEMENTS
} else {
   STATEMENTS
}
```

else

```
public static void test(int x){
    if (x > 5){
        System.out.println(x + " is > 5");
    } else {
        System.out.println(x + " is not > 5");
    }
}
public static void main(String[] args){
    test(6);
    test(5);
    test(4);
}
```

else if

```
if (CONDITION1) {
   STATEMENTS
} else if (CONDITION2) {
   STATEMENTS
} else if (CONDITION3) {
   STATEMENTS
} else {
   STATEMENTS
}
```

else if

```
public static void test(int x){
    if (x > 5){
        System.out.println(x + " is > 5");
    } else if (x==5) {
        System.out.println(x + " equals > 5");
    } else {
        System.out.println(x + " is not > 5");
    }
}
public static void main(String[] args){
    test(6);
    test(5);
    test(4);
}
```

The Unary Operators

+	**Unary plus operator; indicates positive value (numbers are positive without this, however)**
–	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
−−	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

The Unary Operators

- The increment/decrement operators can be applied before (prefix) or after (postfix) the operand. The code

```
result++;
++result;
```

- If you are just performing a simple increment/decrement, it doesn't really matter which version you choose.

The Unary Operators

- **value = ++result;** //equivalent to

  result = result + 1;

  value = result;
- **value = result++;** //equivalent to

  int temp = result

  value = temp;

  result = result + 1;

Compound Assignments

- You can also combine the arithmetic operators with the simple assignment operator to create compound assignments.

  x+=1; and x=x+1;

  both increment the value of x by 1.

Operator Precedence

Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr !
multiplicative	* / %
additive	+ -
relational	< > <= >=
equality	== !=
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %=

Blocks

- A block is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed.

```
if (condition) { // begin block 1
        System.out.println("Condition is true.");
    } // end block one
```

Variable Scope

- Variables live in the block ({}) where they are defined (scope)

- Method parameters are like defining a new variable in the method

Variable Scope

```
public static void main(String[] args)
  {
     int x=10;
     if(x==10)
    {
        int y=11;
        System.out.println("Both variables are known here: "+x+" and "+y);
    }
    y=y+1;
     System.out.println("The variable y is not known here!");
  }
```

Variable Scope

```
public class SquareChange {
    public static void printSquare(int x){
        System.out.println("printSquare x = " + x);
        x = x * x;
        System.out.println("printSquare x = " + x);
    }
    public static void main(String[] args){
        int x = 5;
        System.out.println("main x = " + x);
        printSquare(x);
        System.out.println("main x = " + x);
    }
}
```

Variable Scope

main x = 5

printSquare x = 5

printSquare x = 25

main x = 5

Variable Scope

```
public class Scope {
    public static void main(String[] args){
        int x = 5;
        if (x == 5){
            int x = 6;
            int y = 72;
            System.out.println("x = " + x + " y = " + y);
        }
        System.out.println("x = " + x + " y = " + y);
    }
}
```

Variable Scope

```
Scope.java:5: error: variable x is already defined in method
   main(String[])
                    int x = 6;
                        ^
Scope.java:9: error: cannot find symbol
          System.out.println("x = " + x + " y = " + y);
                                                  ^
  symbol:   variable y
  location: class Scope
2 errors
```

Loops

Loops

```
public static void main (String[] args){
  System.out.println("Rule #1");
  System.out.println("Rule #2");
  System.out.println("Rule #3");
}
```

What if you want to do it for 200 Rules?

Loops

- Loop statements allow to loop through a block of code.

- There are several loop statements in Java.

The while statement

```
while (condition) {
  statements
}
```

The while statement

```
int i= 0;
while(i < 3){
   System.out.println("Rule #" + i);
   i = i+1;
}
```

- Count carefully
- Make sure that your loop has a chance to finish.

The do-while statement

```
 do {
 statements
}while (condition)
```

The do-while statement

```
int i= 0;
do {
  System.out.println("Rule #" + i);
  i = i+1;
}while(i < 3);
```

- Evaluates its expression at the bottom of the loop instead of the top.
- Statements within the do block are always executed at least once.

The for statement

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

The for statement

```
for(int i = 0; i <3; i=i+1){
    System.out.println("Rule #" + i);
}
```

Note: i = i+1 may be replaced by i++

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);
}
```

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
```

Embedded loops

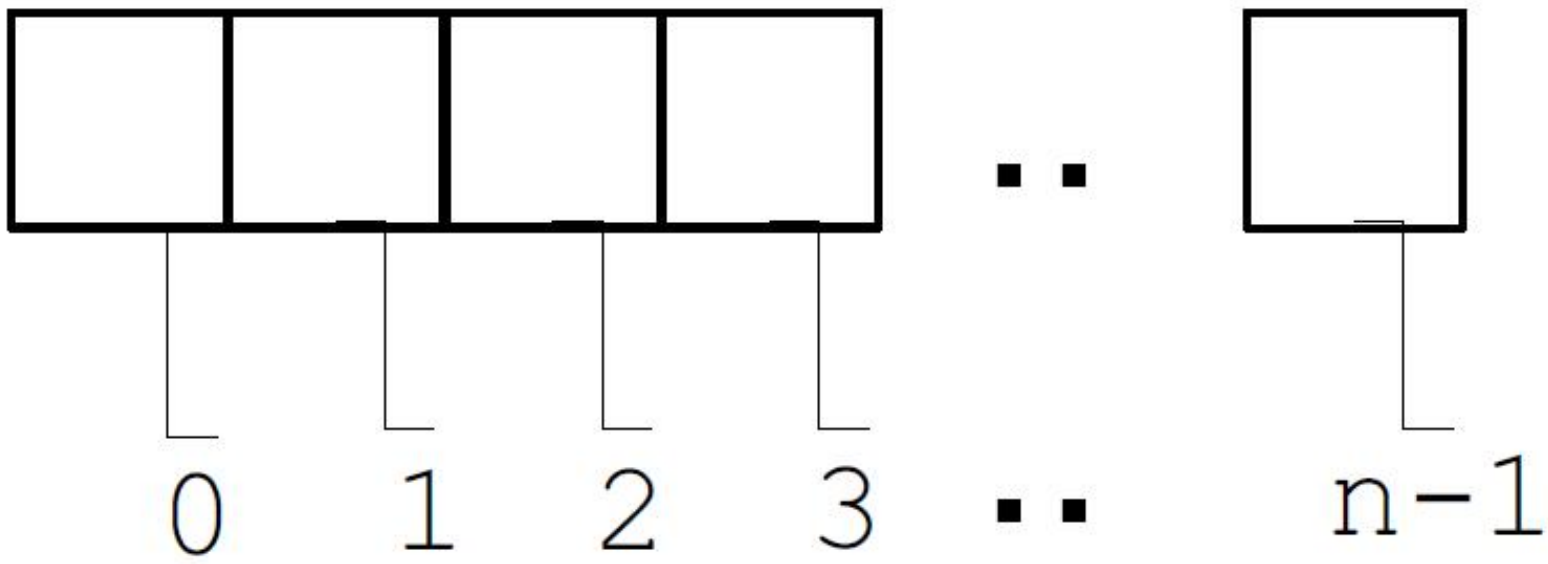
```
for (int i = 0; i < 3; i++) {
   for (int j = 2; j < 4; j++) {
        System.out.println (i + “ “ + j);
   }
}
```

Arrays

Arrays

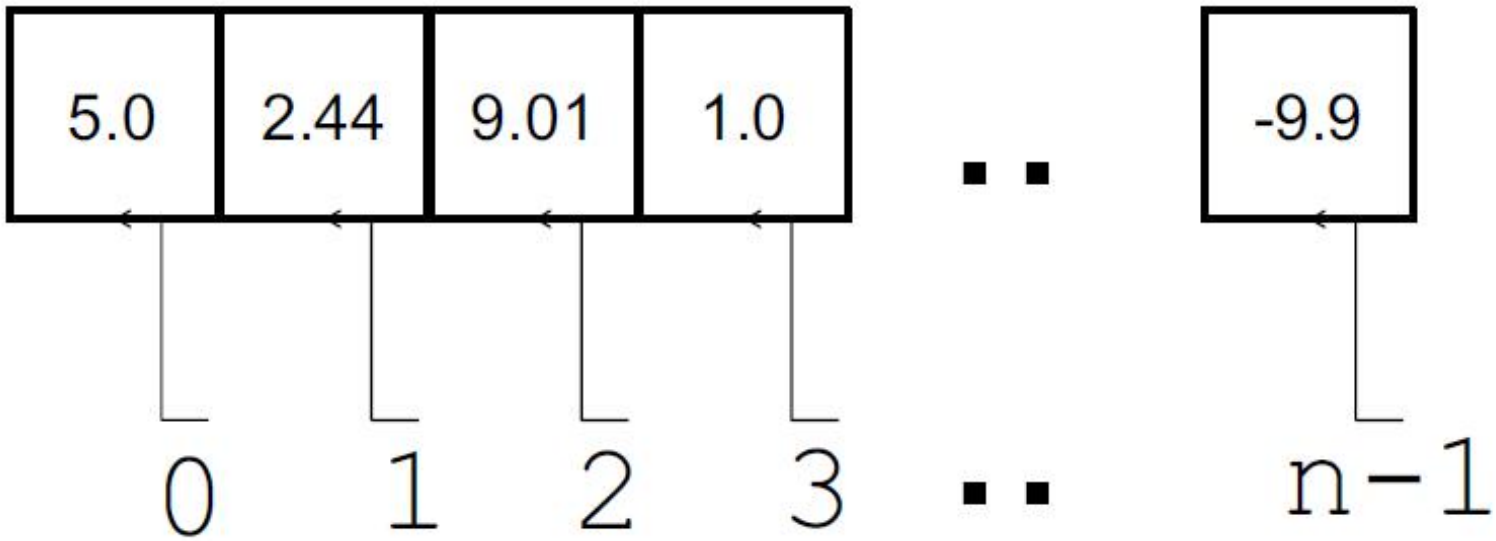
- An array is an indexed list of values.
- You can make an array of any type int, double, String, etc..
- All elements of an array must have the same type.

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]

}
```

Command-line argument

```
public class Echo {
    public static void main (String[] args) {
            for (int i = 0; i< args.length; i++) {
             System.out.println(args[i]);
        }
    }
}
```

Command-line argument

```
public class Echo {
    public static void main (String[] args) {
        for (int i = 0; i< args.length; i++) {
         System.out.println(args[i]);
      }
    }
}
```

```
ozgur@ubuntu:~/java$ javac Echo.java
ozgur@ubuntu:~/java$ java Echo a b c d e
a
b
c
d
e
ozgur@ubuntu:~/java$
```

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://docs.oracle.com/javase/tutorial/java
- https://www.cs.upc.edu/~jordicf/Teaching/programming/pdf/IP07_Recursion.pdf