Two operators / and % - under the context of integer division

m / n : m divided by n

m % n: the remainder after m is divided by n

For example: 7 / 2 =3, 7 % 2=1

1) **Equality/Relational/Logical Operators**
2) Relational Operators: <, <=, >, >=
3) Logical Operators: !, &&, ||
4)

IF Statement

- An *if statement* allows a program to choose whether or not to execute a particular statement.
- An *if-else statement* allows a program to do one thing if a condition is true and a different thing if the condition is false
- Using *block statement* to execute more than one statement as the result of evaluating a Boolean condition. (a block is a list of statements enclosed in braces).

```
if ( num1 < num2) {
    min =  num1;
    max = num2;
}
else {
    min =  num2;
    max = num1;
}
```

- In a *nested if statement*, an *else* clause is matched to the closet unmatched *if*.

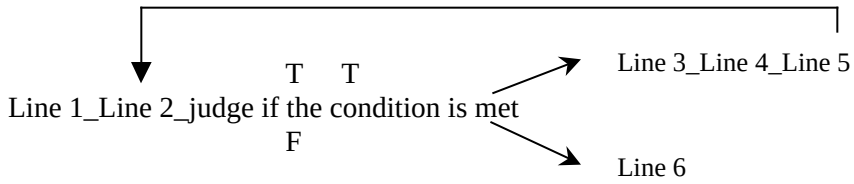
```
if ( num1 < num2)
   if ( num1 < num3)
       min =  num1;
   else
       min =  num3;
else
   if ( num2 < num3)
       min =  num2;
   else
       min =  num3;
```

WHILE Statement – section 3.6

- A *while* statement allows a program to execute the same statement multiple times.

```
Line 1:
Line 2:          while (a condition)
Line 3:          {
Line 4:
Line 5:          }
Line 6:
```

The order of execution



The for statement

The while statement is good to use if you don't know how many times you will have to execute the body of a loop. The for statement allows you to execute the body of a loop a set number of times. For example: Code that would display 10 9 8 7 6 5 4 3 2 1 in the console window written in while and for loops:

while statement	*for* statement
int count = 10; while (count > 0) { System.out.print(count+" "); count = count –1; }	for (int count=10;count>0;count--) { System.out.print(count+ " "); }

The structure of the *while* statement and the *for* statement:

while statement	*for* statement
variable initialization; while (*condition*) { *statement;* *increment;* }	for (*variable initialization; condition; increment*) *statement;*

The header of a *for* loop contains three parts separated by semicolons:

First part (initialization): **sets up a loop control variable**
Second part (boolean condition): **checks the loop control variable**
This condition is evaluated before the loop body.
True: the body of the loop is executed.
False: jump to the next statement after the *for* statement.
Third part (increment): **modifies the loop control variable**
This part is executed after each iteration of the loop. It does not have to perform a simple increment. The examples of this part could be:
count--; // subtract 1 from count
count=count+5; // add 5 to count

Arrays

int [] height; // declares an array variable height which will contain integer values. the
// array type information here is given in *int []*.
height = new int[11]; // sets the length of the array to be 11. height will be a list with
// 11 values, ** they will be indexed from 0 to 10.
height[2]=11; // this statement sets the value of the 3rd element in the array (at position
// 2) to be 11.
String [] names = {"Jim", "Sue", "Sally"};
// declares an array names which will contain string values and gives entire array.

The length of the array once defined is constant. You can determine the length of an array almost the same way you determine the length of a string. For example, if *names = {Jim, Sue, Sally}*, then *names.length* = 3.

Strings

To use any of the methods in the string class, you must follow the following format:
your_string_variable_name.method_name (any input the method requires);
Some useful methods and examples (for examples, assume myname = "John Jones"):

toLowerCase() / toUpperCase ()
myname.toLowerCase(); // returns "john jones"
charAt(int idx)
myname.charAt(0); // returns "J" . Note1st position given value of 0
myname.charAt(6); // returns "o"
length()
myname.length(); // returns 10
replace(char oldcharacter, char newcharacter)
myname.replace('J', 'p'); //returns "pohn pones"

Class: read the last handout