Operator Precedence

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

What's the output

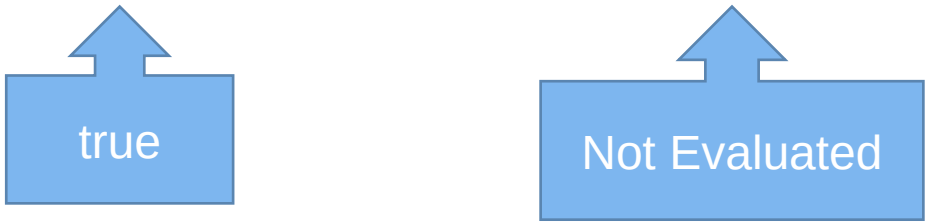
```
int i = 2;
boolean bool = ( i < 3 ) || (++i < 3);
System.out.println("bool = " + bool  + " i = "+ i);

bool = ( i > 3 ) && (++i < 3);
System.out.println("bool = " + bool  + " i = "+ i);
```

&&, and || are short-circuit

- short-circuit operators will evaluate the second operand only when necessary

```
( i < 3 ) || (++i < 3)
```



- short-circuit operators have better performance than & and | operators

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.

What's the output?

```
int i = 2;

boolean bool = (i < 5) || ( i < 3 ) && (++i < 3);

System.out.println("bool = " + bool  + " i = "+ i);
```

Precedence and evaluation order are not the same

What's the output?

```
int i = 2;

boolean bool = (i > 5) && ( i > 3 ) || (i++ < 3);

System.out.println("bool = " + bool  + " i = "+ i);
```

Precedence and evaluation order are not the same

What's the result?

```
int i=0;
int result =++i + ++i * ++i;
```

```
int i=0;
int result =++i + i++ * ++i;
```

Precedence and evaluation order are not the same

Order of evaluation is left to right (except for assignment, which evaluates right to left)

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 1
```

Blocks

- Here, if w is less than h, both statements inside the block will be executed.

```
if (w < h) {
   v = w * h;
   w = 0;
}
```

Start of block

End of block

Blocks

```
/*
  Demonstrate a block of code.

  Call this file BlockDemo.java.
*/
class BlockDemo {
  public static void main(String args[]) {
    double i, j, d;

    i = 5;
    j = 10;

    // the target of this if is a block
    if(i != 0) {
      System.out.println("i does not equal zero");
      d = j / i;
      System.out.println("j / i is " + d);
    }
  }
}
```

The target of the **if** is this entire block.

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;  //Compilation Error
     System.out.println("The variable y is not known here!");
  }
```

Variable Scope

```
public class Scope {
    public static void main(String[] args){
        int x = 5;
        if (x == 5){
            int x = 6;  //Error, x already exists in this scope
            int y = 72;
            System.out.println("x = " + x + " y = " + y);
        }
        System.out.println("x = " + x + " y = " + y);  //Error
    }
}
```

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
```

Scope Demo

```
// Demonstrate block scope.
class ScopeDemo {
  public static void main(String args[]) {
    int x; // known to all code within main

    x = 10;
    if(x == 10) { // start new scope

      int y = 20; // known only to this block

      // x and y both known here.

      System.out.println("x and y: " + x + " " + y);
      x = y * 2;
    }
    // y = 100; // Error! y not known here ←——— Here, y is outside of its scope.

    // x is still known here.
    System.out.println("x is " + x);
  }
}
```

Scope

- Scope of the method parameters is the entire method.

```
public static void foo(int x) {
    int x = 5;  //Error duplicate variable
}
```

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.

Print Alphabet Example

```
// Demonstrate the while loop.
class WhileDemo {
  public static void main(String args[]) {
    char ch;

    // print the alphabet using a while loop
    ch = 'a';
    while(ch <= 'z') {
      System.out.print(ch);
      ch++;
    }
  }
}
```

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++

SqrRoot Example

```
// Show square roots of 1 to 99 and the rounding error.
class SqrRoot {
  public static void main(String args[]) {
    double num, sroot, rerr;

    for(num = 1.0; num < 100.0; num++) {
      sroot = Math.sqrt(num);
      System.out.println("Square root of " + num +
                         " is " + sroot);

      // compute rounding error
      rerr = num - (sroot * sroot);
      System.out.println("Rounding error is " + rerr);
      System.out.println();
    }
  }
}
```

Decrementing For Loop

```
// A negatively running for loop.
class DecrFor {
  public static void main(String args[]) {
    int x;

    for(x = 100; x > -100; x -= 5)
      System.out.println(x);
  }
}
```

Loop control variable is decremented by 5 each time.

Equivalent To

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

}
```

References

- Dr. Öğr. Üyesi Özgür KILIÇ MSKÜ Bilgisayar Mühendisliği Bölümü
- 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