Flow control and arrays

GEORGIA'S INNOVATION
AND TECHNOLOGY AGENCY

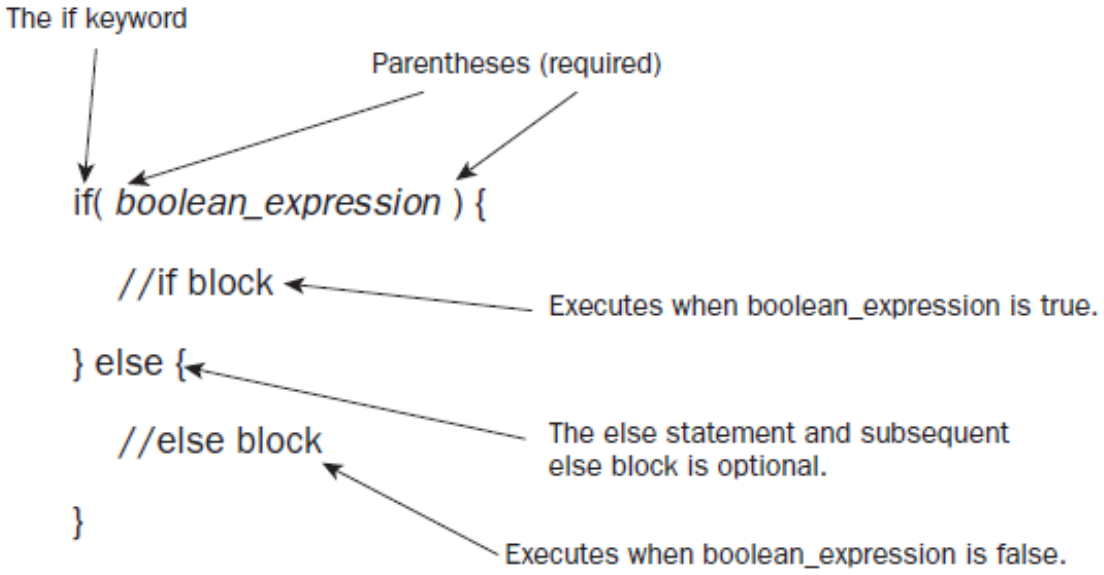
skillUp
The expert of IT education

The *if* and *switch* statement

The ***if*** statement have the following syntax:

```
if (boolexp) { /*operators*/} //1
else { /*operators*/ } //2
```

If the *boolexp* expression equals true, then runs a group of operators 1, otherwise group of operators 2. It is also possible to use the syntax **if {} else if {}**.



The *if* and *switch* statement

The following rules apply to an **if - else** statement:

- The expression in parentheses must evaluate to a **boolean**. Otherwise, a compiler error is generated.

- If the boolean_expression evaluates to **true**, the block of code following the **if** executes.

- If the boolean_expression evaluates to **false**, the else block executes.

- The else block is optional.

- The curly braces are not required in either the **if** or **else** block if the block of code is a single statement. However, for readability it is a good idea to always use the curly braces.

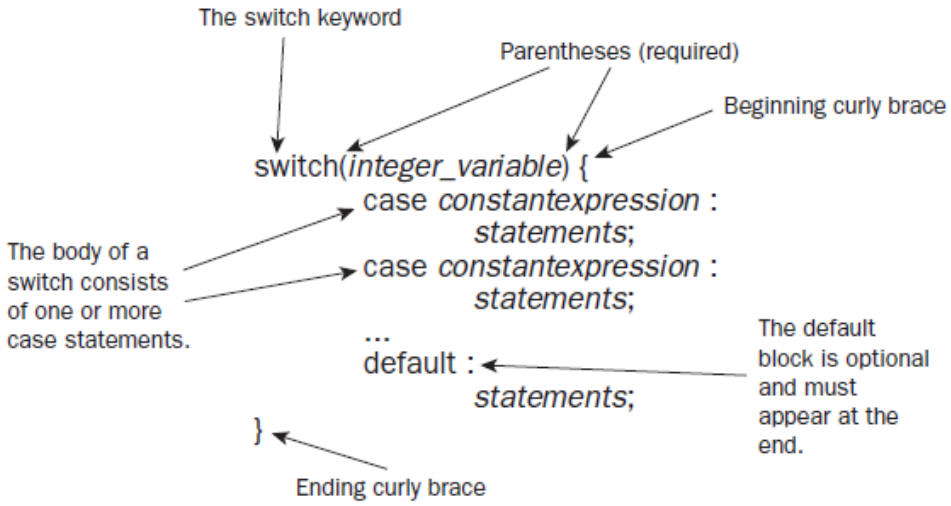
- An else block can contain an additional if statement.

The *if* and *switch* statement

The ***switch*** statement have the following syntax:

```
switch(exp) {
    case exp1:{/*operators*/}
         break;
    case expN:{/* operators */}
         break;
    default:  {/* operators */}
}
```

If the **exp** will be equal to **exp1** performed successively all the blocks of statements as long as the operator does not meet **break.** Values **exp1, ..., expN** must be constants, and can have a value of type int, byte, short, char or enum.



The *if* and *switch* statement

The following rules apply to using **switch** statements:

- The ***integer_variable*** must be compatible with an **int**, which means you can only switch on a **byte , short , char , int , Byte , Short , Character , Integer** , or an **enum** type.

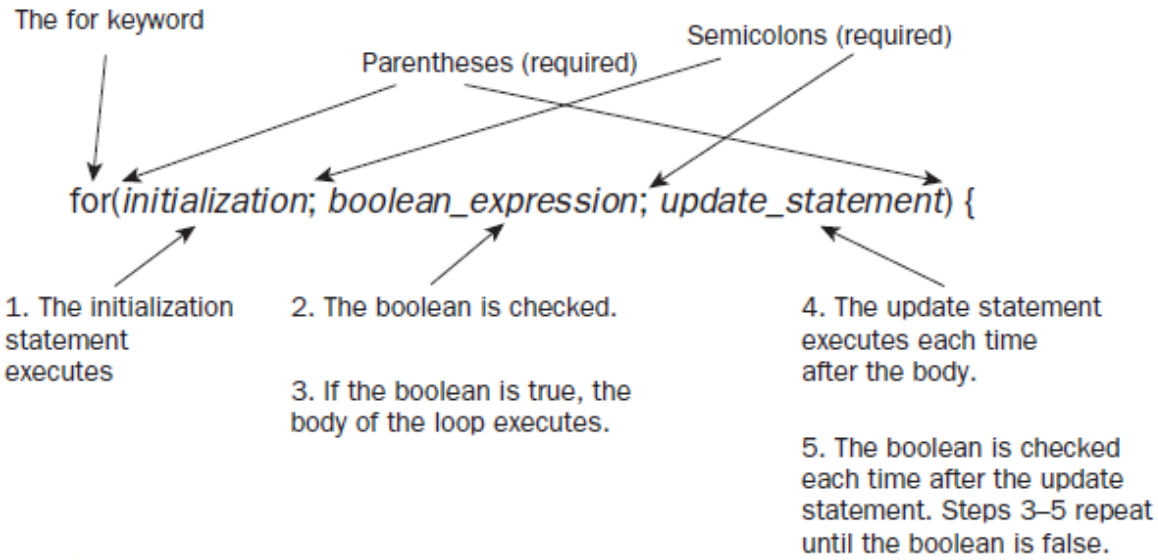
- Any number of case statements can appear.

- The ***constantexpression*** of a **case** must be a literal value or a **final** variable.

- The **default** block is optional and must appear at the end of all the **case** statements. If none of the **case** statements equal the expression, the **default** block executes.

- When a **case** is true, no other **case** statements are tested for equality, and all statements following the **case** execute until a **break** occurs or the end of the switch statement is reached.

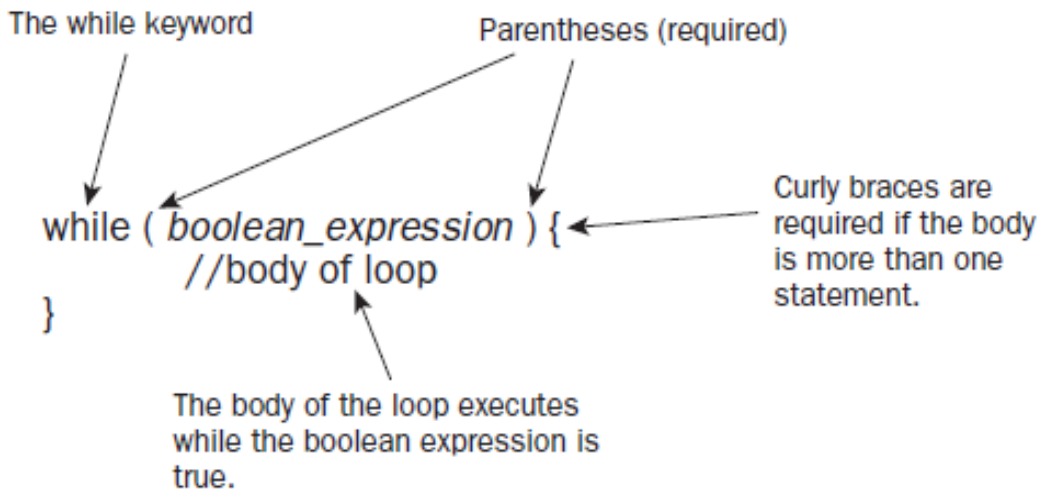
The *for*, *while* and *do while* statement



A basic ***for*** statement has the following properties:

- The two semicolons are required and create three sections: an *initialization* statement, a *boolean* expression, and an update statement.
- The *initialization* step occurs once at the beginning of the loop.
- The *boolean_expression* must evaluate to true or false.
- The *initialization* and *update_statement* sections can contain multiple statements, separated by commas.

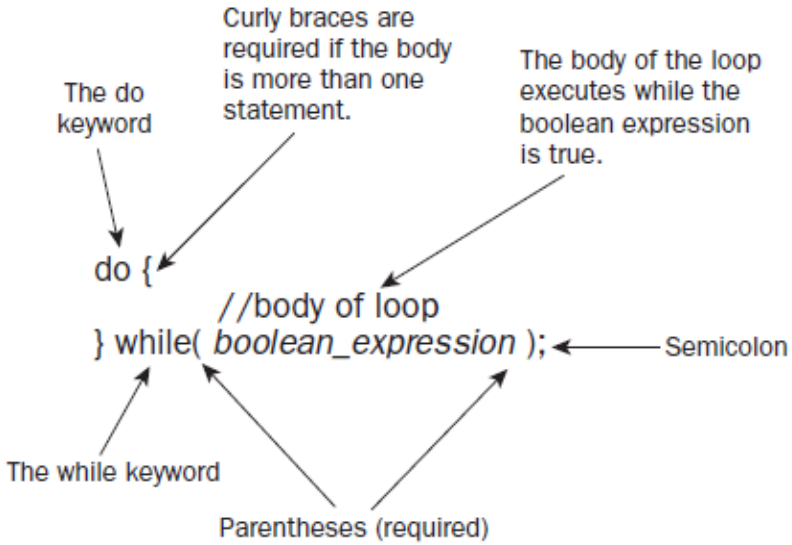
The *for*, *while* and *do while* statement



The following rules apply to a ***while*** statement:

- The value in parentheses must evaluate to a ***boolean*** expression, either true or false.
- If the ***boolean*** expression is true , the body of the loop executes and the ***boolean*** is checked again.
- If the ***boolean*** expression is *false*, the loop does not execute and control jumps to the next statement following the end of the loop.
- The body of the loop executes until the ***boolean*** expression is false.

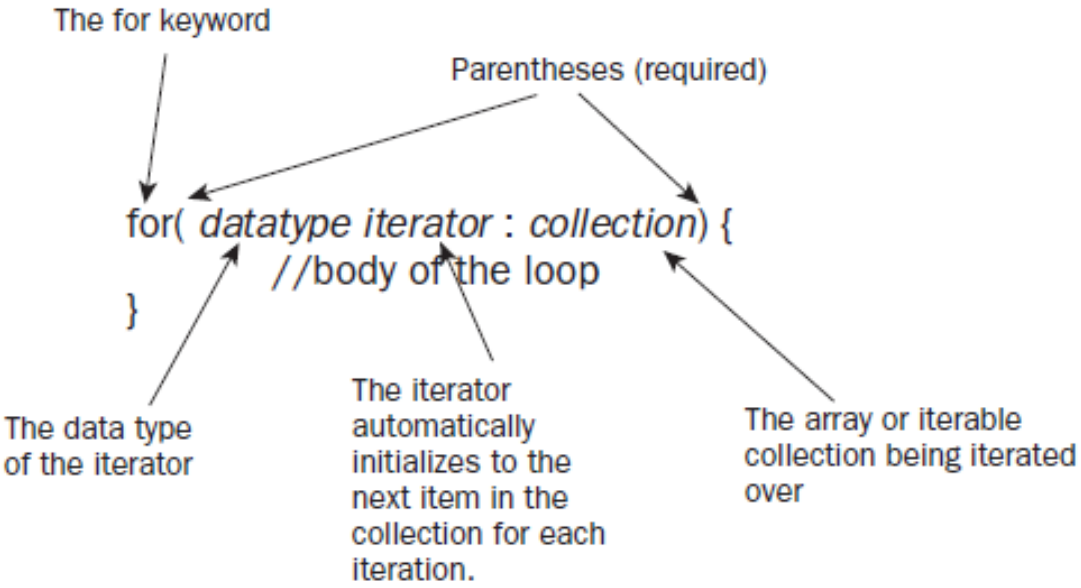
The *for*, *while* and *do while* statement



The following rules apply to a do statement:

- The body of the loop executes once before the boolean expression is tested.
- The value in parentheses must evaluate to a boolean expression, either true or false .
- If the boolean expression is true , the body of the loop executes again, and then the boolean is checked again.
- If the boolean expression is false , the loop does not execute again and control jumps to the next statement following the end of the loop.
- Just like a while loop, the body of the do loop executes until the boolean expression is false.
- Don ’ t forget the semicolon after the boolean expression — it ’ s easy to miss!

The *for*, *while* and *do while* statement



An enhanced for statement has the following properties:

- The data type of the iterator must be compatible with the data type of the collection.
- The scope of the iterator is the body of the loop.
- The number of iterations of the loop equals the size of the collection. If the collection is empty, the body of the loop does not execute.
- The collection must be an array or an object of type java.lang.Iterable , an interface introduced in Java 5.0 exclusively for for - each loops.

The *for*, *while* and *do while* statement

```
/*sample #1: for-each*/
 int[] array = {1, 2, 3};
 for(int i : array)
  System.out.printf("%d ", i);
```

Some of the recommendations in the design loops:
- checking the conditions for all cycles is performed only once per iteration, for ***for*** and ***while*** loop - before the iteration cycle for the ***do/while*** - at the end of the iteration.
- the cycle ***for*** use during execution of the algorithm strictly necessary a number of times. The ***while*** loop is used when the unknown number of iterations to achieve the desired result, for example, searching for the required values in an array or collection.
- for loop ***for*** is not recommended to change the index of loop inside the loop.
- for the indices should be used meaningful names.
- loops should not be too long. This loop applies for division into a separate method.
- nested loops should not exceed three.

Arrays

An array is an object, where the name of the array is an object reference. Array elements can be values of the primitive type or objects. To create array requires memory allocation using the ***new*** operator or direct initialization.

```
/* sample # 2 : arrays and references: ArrayClass.java */
package chapt02;

public class ArrayClass {
  public static void main(String[] args) {
    int myRef[], my; //one array and int
    float[] myRefFloat, myFloat; // two arrays
    // declaring with initialization by zero
    int myDyn[] = new int[10];
    /*declaring with initialization */
    int myInt[] = {5, 7, 9, -5, 6, -2};
    byte myByte[] = {1, 3, 5};
    Object myObj = new float[5];
  }
}
```

Wrapper classes

Multidimensional arrays in Java does not exist, but it is possible to declare an array of arrays. To set the initial values of the array, there is a special form of initializer, for example:

```
int arr[][] = {    { 1 },
                   { 2, 3 },
                   { 4, 5, 6 },
                   { 7, 8, 9, 0 }
               };
```

The first index indicates the serial number of the array, for example arr[2][0] indicates the first element of the third array, a value of **4.**

Question?