Control Statements

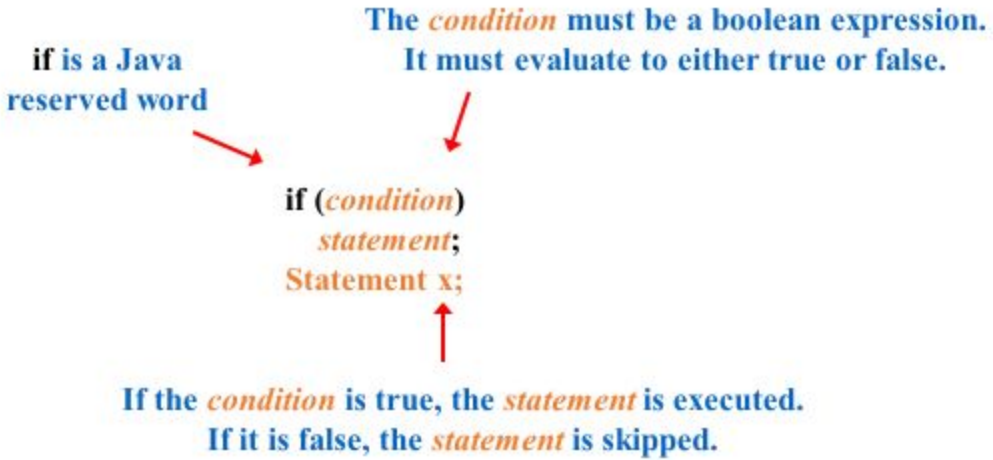
Syeda Ajbina Nusrat, Lecturer, CSE, UITS

Control Statement

- ✓Java's control statements can be put into following categories:
  - ✓Sequence
  - ✓Selection statement
  - ✓Iteration statement
  - ✓Jump statement
- ✓Three selection statements:
  1. if statement
  2. switch statement
  3. conditional operator statement

The if Statement

✓The *if statement* has the following syntax:



The if-else Statement

✓ An *else clause* can be added to an if statement to make an *if-else statement*

```
if ( condition )
  statement1;
else
  statement2;
Statement x;
```

If the `condition` is true, `statement1` is executed; if the condition is false, `statement2` is executed

One or the other will be executed, but not both

Nested if….Else Statements

✓The if..else statement can be contained in another if or else statement.

```
if (test condition1)
{
        if (test condition2)
                statement-1;
        else
                statement-2;
}
else
        statement-3;

statement-x;
```

Nested if….Else Statements

- ✓An else clause is matched to the last unmatched if (no matter what the indentation implies!)

- ✓Example:

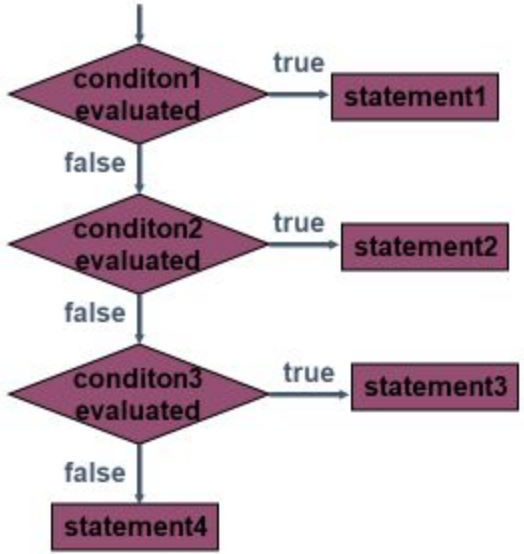
```
if(female)
        if(bal>5000)
                bon = 0.05 * bal;
else
        bon = 0.02 * bal;
bal = bal + bon;
```

- ✓Braces can be used to specify the if statement to which an else clause belongs

The if-else-if ladder

✓Sometime you want to select one option from several alternatives

```
if (conditon1)
   statement1;
else if (condition2)
   statement2;
else if (condition3)
   statement3;
else
   statement4;
```



The switch Statement

- ✓The *switch statement* provides another means to decide which statement to execute next
- ✓The switch statement evaluates an expression, then attempts to match the result to one of several possible *cases*
- ✓The expression of a switch statement must result in an *integral type* (byte, short, int, char etc)

  Note: JDK 7 allows expression can be of type **String**.
- ✓The flow of control transfers to statement associated with the first value that matches

The switch Statement

✓The general syntax of a switch statement is:

```
switch (expression) {
   case value1:
      statement-list1
      break;
   case value2:
      statement-list2
      break;
   case value3:
      statement-list3
      break;
   case default:
      statement-list4
}
```

switch and case are reserved words

If *expression* matches *value2*, control jumps from here

The switch Statement

- A break statement causes control to transfer to the end of the switch statement
- If a break statement is not used, the flow of control will continue into the next case
- Sometimes this can be appropriate, but usually we want to execute only the statements associated with one case

The switch Statement

- ✓A switch statement can have an optional *default case*
- ✓The default case has no associated value and simply uses the reserved word default
- ✓If the default case is present, control will transfer to it if no other case value matches
- ✓If there is no default case, and no other value matches, control falls through to the statement after the switch

Switch example

```
char letter = 'b';

switch (letter) {
    case 'a':
        System.out.println("A");
        break;
    case 'b':
        System.out.println("B");
        break;
    case 'c':
        System.out.println("C");
        break;
    case 'd':
        System.out.println("D");
        break;
    default:
        System.out.println(”?");
}
```

B

```
char letter = 'b';

switch (letter) {
    case 'a':
        System.out.println("A");
    case 'b':
        System.out.println("B");
    case 'c':
        System.out.println("C");
        break;
    case 'd':
        System.out.println("D");
        break;
    default:
        System.out.println(”?");
}
```

B
C

The Conditional Operator

- ✓Java has a *conditional operator* that evaluates a boolean condition that determines which of two other expressions is evaluated
- ✓The result of the chosen expression is the result of the entire conditional operator
- ✓Its syntax is:
- ✓ *condition* ? *expression1* : *expression2*
- ✓If the *condition* is true, *expression1* is evaluated; if it is false, *expression2* is evaluated

The Conditional Operator

✓The conditional operator is similar to an if-else statement, except that it forms an expression that returns a value

✓For example:

✓ larger = ((num1 > num2) ? num1 : num2);

✓
```
if (num1 > num2)
    larger = num1;
else
    larger = num2;
```

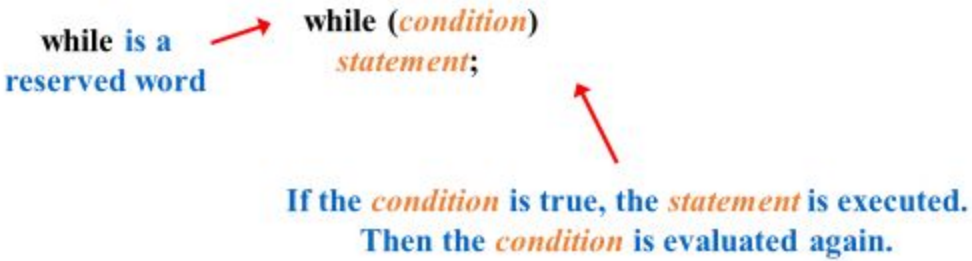
✓The conditional operator is *ternary* because it requires three operands

Iteration Statements

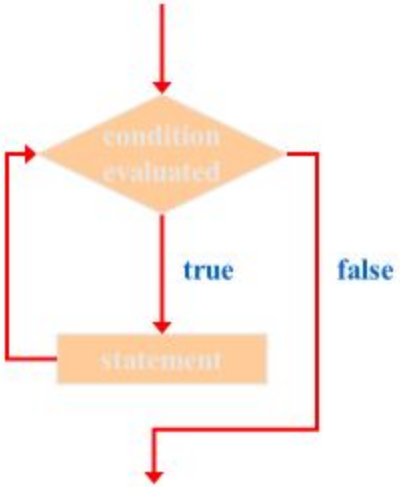
- ✓*Iteration statements* allow us to execute a statement multiple times until a termination condition is met.
- ✓Often they are referred to as *loops*
- ✓Java has three kinds of iteration statements:
  - ✓the *while loop*
  - ✓the *do-while loop*
  - ✓the *for loop*

The while Statement

✓The *while statement* has the following syntax:



Logic of a while Loop



Note that if the condition of a while statement is false initially, the statement is never executed. Therefore, the body of a while loop will execute zero or more times

while Loop Example

```
int LIMIT = 5;
int count = 1;

while (count <= LIMIT) {

    System.out.println(count);
    count += 1;
}
```

--Null statements are valid in java.

Output:

1
2
3
4
5

Nested Loops

- ✓Similar to nested if statements, loops can be nested as well
- ✓That is, the body of a loop can contain another loop
- ✓Each time through the outer loop, the inner loop goes through its full set of iterations

The do-while Statement

✓The *do-while statement* has the following syntax:

do and **while** are reserved words

```
do{
   statement;
} while (condition);
```

The *statement* is executed once initially, and then the *condition* is evaluated

The *statement* is executed repeatedly until the *condition* becomes false

do-while Example

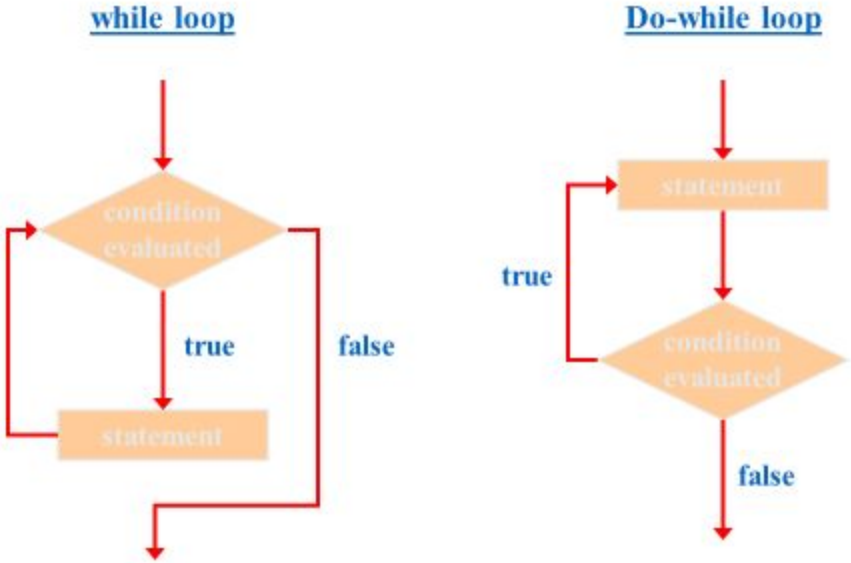
```
int LIMIT = 5;
int count = 1;

do {
   System.out.println(count);
   count += 1;
} while (count <= LIMIT);
```

Output:

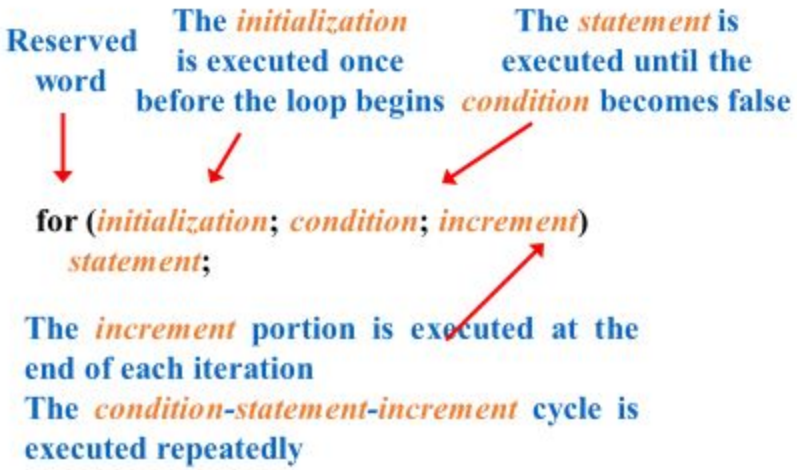
1
2
3
4
5

Comparing while and do-while



The for Statement

✓The *for statement* has the following syntax:

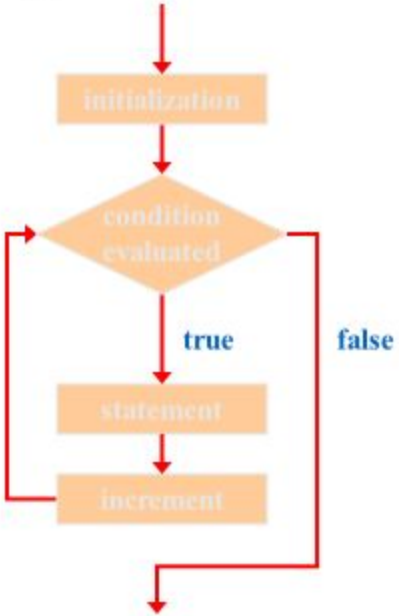


The for Statement

✓A for loop is functionally equivalent to the following while loop structure:

```
initialization;
while (condition) {
   statement;
   increment;
}
```

Logic of a for loop



for Example

```
int LIMIT = 5;

for (int count = 1; count <= LIMIT; count++) {

   System.out.println(count);
}
```

Output:

1
2
3
4
5

The for Statement

- ✓Each expression in the header of a for loop is optional
  - ✓If the *initialization* is left out, no initialization is performed
  - ✓If the *condition* is left out, it is always considered to be true, and therefore creates an infinite loop
  - ✓If the *increment* is left out, no increment operation is performed
- ✓Both semi-colons are always required in the for loop header

Jump Statements-Break

- The break statement has three uses:
  - It terminates a statement sequence in a switch statement.
  - It can be used to exit a loop.
  - It can be used as a civilized form of goto.
- **Civilized Form of Goto Statement**
- break *lable*;

  where *label* is the name of the block enclosing the break statement.
- Labled break is used to transfer control from a set of nested blocks.

Example-break statement

```
class test_break1
{     public static void main(String args[])
      {
            boolean t =true;
             first:{
                  second:{
                        third:{
                              System.out.println("Before the break.");
                            if(t)
                              break second;
                              System.out.println("This won't execute");
                        }
                        System.out.println("This won't execute");
                  }
                  System.out.println("This is after second block.");
            }
      }
}
```

Example-break statement

```
class test_break2
{
    public static void main(String args[])
    {
        one: for(int i=0;i<3;i++)
        {
            System.out.print("Pass: "+i+": ");
        }

        for(int j=0; j<100;j++)
        {
            if(j==10) break one;  //Wrong
            System.out.print(j+" ");
        }
    }
}
```

Jump statement-continue

- Continue statement is used to run a loop but stop processing the remainder of the code in its body for a particular iteration.
- In **while** and **do-while** loop, a continue statement causes control to be transferred directly to the conditional expression. In a **for** loop, control goes first to the iteration portion of the for statement and then to the conditional expression.

Example-Continue statement

```
class test_continue
{       public static void main(String args[])
        {       outer: for(int i=0; i<4;i++){
                        for(int j=0;j<4;j++) {
                        if(j>i) {
                                System.out.println();
                                continue outer;
                        }
                        System.out.print(" "+(i*j));
                }
        }
        System.out.println();
    }
}
```

Output:

```
0
0 1
0 2 4
0 3 6 9
```

Jump Statement-return

```
class test{
  public static void main (String args[])
  {
        boolean t =true;
        System.out.println("Before the return");
        if(t) return;

        System.out.println("This won't execute.");
  }
}
```

Nested Loop

- Read and practice by yourself.


“
The day is what you make it! So why not make it a great one?
STEVE SCHULTE
AVEMATEIU.COM