

Conditionals and Loops Cheat Sheet

Conditionals

IF...ELSE IF...ELSE

The conditional statement or if...else statement directs the flow of code, allowing the programmer to make select code blocks to run based on the results of boolean expressions.

Structure

```
if (boolean expression) {                (required)

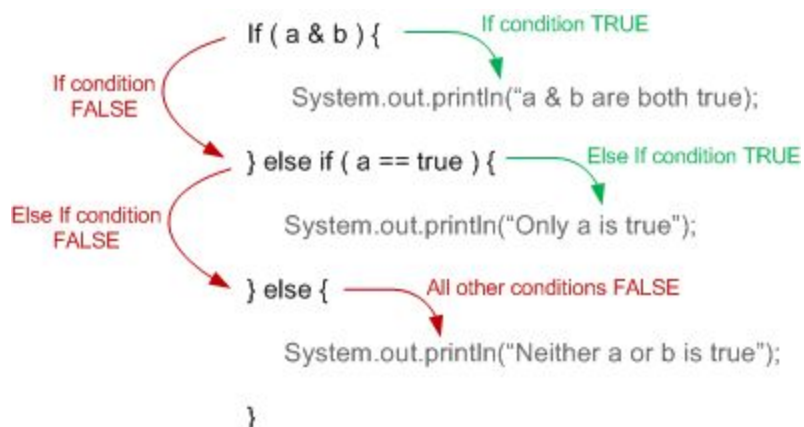
} else if (boolean expression) {         (optional, can have 0...n)

else {                                   (optional, can have 1)

}
```

Explanation

When the first TRUE condition is found, then that block of code is run, and all others are skipped. If all other conditions are FALSE, then the block in the else condition is run.



Examples

```
if ( x < 3 ) {  
  
}
```

```
if ( x < 3 ) {  
  
} else {
```

```
if ( x < 3 ) {  
  
} else if ( x > 6 ) {
```

```
if ( x < 3 ) {  
  
} else if ( x > 6 ) {
```

```
}
```

```
}
```

```
}
```

```
else if ( x < 12 ) {
```

```
} else {
```

```
}
```

Switch

The switch statements works like an if statement, but rather than check a custom boolean condition it checks if a variable is equal to one of a set of values.

Structure

```
switch (variable) {
```

(required)

```
    case 10:
```

(1...n cases required)

```
        break;
```

(break required for each case)

```
    case 20:
```

```
        break;
```

```
    default:
```

(Optional, run if all others false)





```
        break;
```

(break required for default)

```
}
```

Explanation

The switch statement checks the value of the variable, against each possible value that is a case. If runs the code for the first matching value found, and then skips the rest of the cases. If no cases are found that match the value of the variable, then the default code is run, if default has been included.

```
switch ( a ) {  
  case 10:  If a is equal to 10  
    System.out.println("the value of a is 10");  
    break;  
  
  case 20:  If a is equal to 20  
    System.out.println("the value of a is 20");  
    break;  
  
  case 30:  If a is equal to 30  
    System.out.println("the value of a is 30");  
    break;  
  
  default:  All other conditions are FALSE ( a is equal to  
    something other than 10, 20, or 30 )  
    System.out.println("the value of a is something other than 10, 20, or 30");  
    break;  
}
```

Example

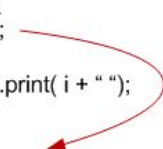

```
switch ( channelName ) {  
  case "ABC":  
    System.println("The channel Name is ABC");  
    break;  
  case "NBC":  
    System.println("The channel Name is NBC");  
    break;  
  case "CBS":  
    System.println("The channel Name is CBS");  
    break;  
  default:  
    System.println("The channel Name is not a known value");  
    break;  
}
```

Loops

Loops allow for a block of code to be repeated. The number of times the block is repeated is determined by the boolean condition set in the loop. The loop will repeat as long as the condition is true and stop once the

condition becomes false. To avoid loops that do not end, infinite loops, care should be taken that either something in the boolean condition changes for each iteration, or code is in place to exit the loop.

There are two special commands that can be used in any loop.

break	immediately ends the execution of the loop	<pre>for (int i = 0; i < 10 ; i++) { if (i == 5) { break; } System.out.print(i + " "); }</pre>  <p>The break causes the loop to exit. No further iterations are performed.</p> <p>The break exits the loop when i is equal to 5, so this loop prints: 0 1 2 3 4</p>
continue	stops the current loop iteration, and continues on the next iteration. This means that the code after the continue in the code block will not execute for that iteration.	<pre>for (int i = 0; i < 10 ; i++) { if (i == 5) { continue; } System.out.print(i + " "); }</pre>  <p>The continue skips the rest of the iteration, and goes to the next one.</p> <p>The continue skips the rest of the code in the block for this iteration, and continues to the next iteration, so this loop prints: 0 1 2 3 4 6 7 8 9</p>

Types of Loops

for

The for loop uses an incrementer to keep track of the number of iterations the loop will repeat. Each time an iteration completes, the incrementer is changed and the loops continues if the condition in boolean expression is still true.

Structure

Standard structure

```
for ( int i = 0 ; i < 10 ; i++ ) {
```

Alternative structure,

```
int x = 500;  
for ( ; i > 0 ; ) {
```

```
}
```

```
i = i - 2;
```

```
}
```

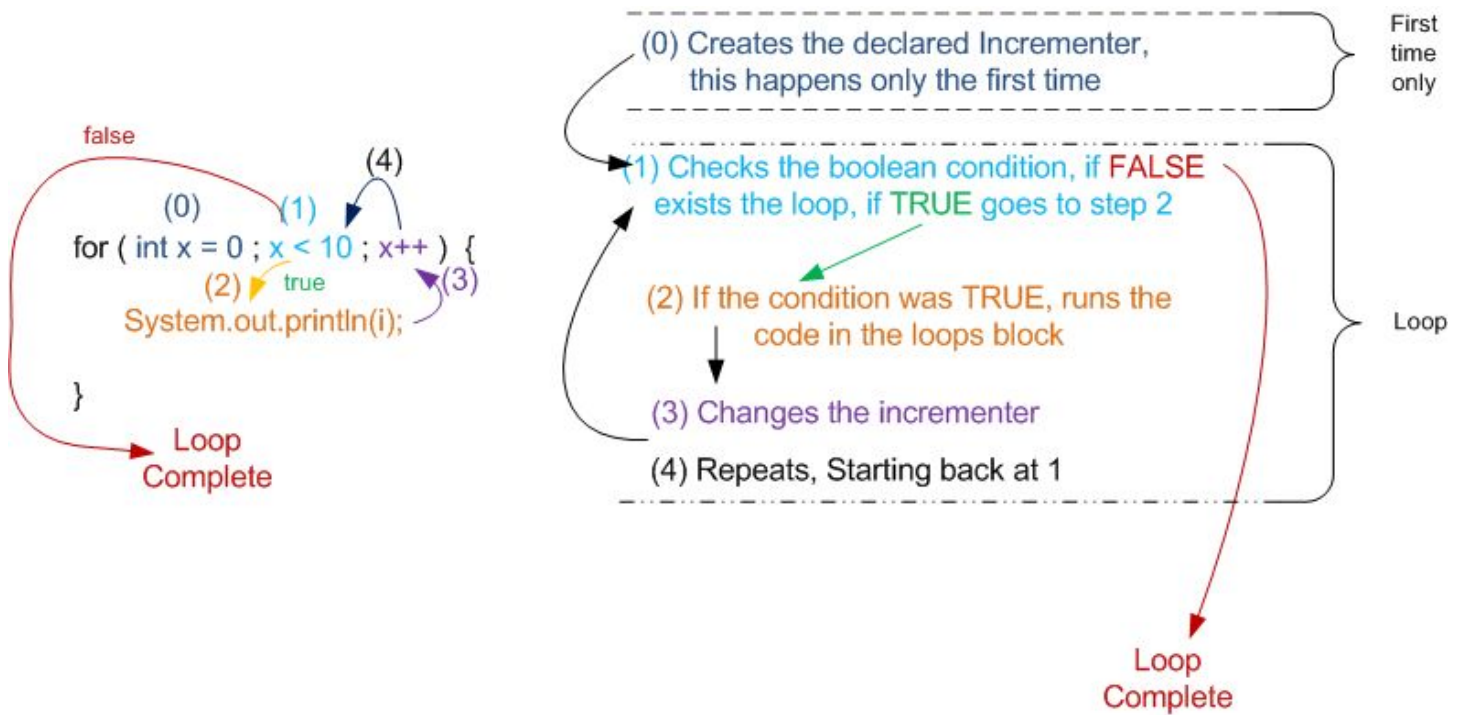
<code>int i = 0</code>	Optional	Defines the incrementer. In this case an int that starts at 0. The starting value can be any value (e.g. int x = 500 would start at 500). The incrementer is optional. It also may be defined here, or can be defined prior to the loop start.
<code>i < 10</code>	Required	The boolean condition that determines if the loop should continue or stop. The loop will continue as long as the boolean condition is TRUE and stop once the boolean condition is FALSE. In this case the loop will continue as long as i is less than 10.
<code>i++</code>	Optional	Changes the incrementer. In this case it uses i++, which is equivalent to $i = i + 1$, so for each loop i is increased by 1. The value of the incrementer can be changed here, in the code block of the loop, or both.

Explanation

The definition of the incrementer is done only the first time the loop is run, it then follows this pattern for each iteration:

1. Check the boolean condition
 - a. if true continue the loop
 - b. if false end the loop
2. Run the code in the loops block
3. Increment/decrement the incrementer
4. Repeat 1-3 until the boolean condition is false

This means that if the boolean condition is false the first time through the loop that the code in the loops block will never execute. Also, since incrementer is changed and then the boolean condition checked, it means that the code in the block will not be run for the incrementer change that makes the boolean condition false.



for each

For each is a simplified version of a For loop that is used to loop through any object that has an iterator, like a Collection or Array. The For Each loop starts at the first item and continues until the last item in the Collection. Both the continue and break command can be used to control the loop, however, items cannot be skipped, accessed by index, and the Collection cannot be modified inside the loop.

Structure

```
for (<type in collection> variableName : collection) {  
  
}
```

<type in collection>	Declares a variable to hold the type of data that is in the Collection. For example, to use a For Each with a List defined as List<String>, the type would be String
variableName	A name that is part of the variable declaration. The next item in the Collection will be stored in this variable, which can be accessed in the loop.
collection	A variable that holds the Array or Collection to loop through

Explanation

For each iteration of the loop the for each gets the next item from the collection and assigns it to the variable that was declared. The loop starts at the first item in the collection and continues until it reaches the last item.

For example, for the following List

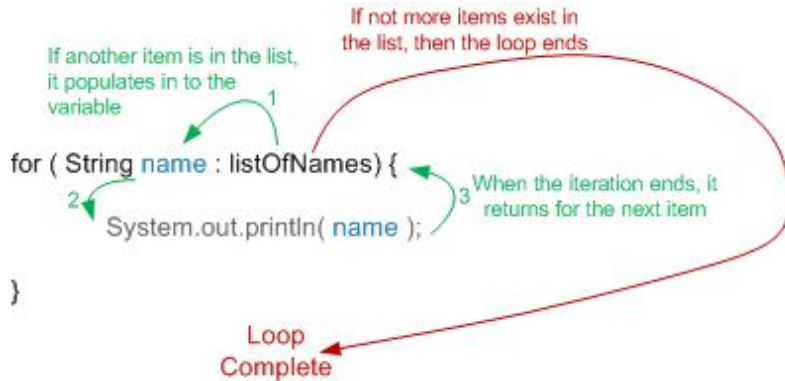
```
List<String> listOfNames = new ArrayList<String>();

names.add("Kelly");
names.add("Joe");
names.add("Steve");
names.add("Jane");
names.add("Susan");
```

And a for each loop defined as:

```
for (String name : listOfNames) {
    System.out.println( name );
}
```

For each iteration of the loop, the variable name would be populated with:



Iteration	name contains
1	Kelly
2	Joe
3	Steve
4	Jane
5	Susan

The output of this for each loop will print:

```
Kelly
Joe
Steve
Jane
Susan
```

while

The while loop uses a boolean condition to determine how long the loop will continue. The loop will continue to iterate as long as the condition is TRUE and will end once the condition is FALSE. Unlike the for loop, the while loop does not have a change to the condition as part of it's syntax, therefore, for a while loop to exit, either the condition must be changed in the body of the loop, or a break statement must be used. This means care must be taken to make sure that a while loop has a way to exit to avoid an infinite loop.

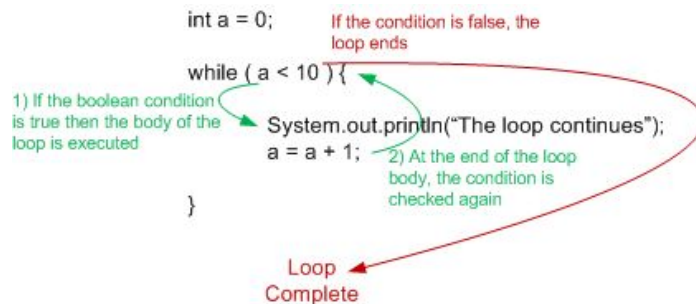
Both the break and continue command can be used in a while loop.

Structure

```
while (boolean condition) {  
  
}
```

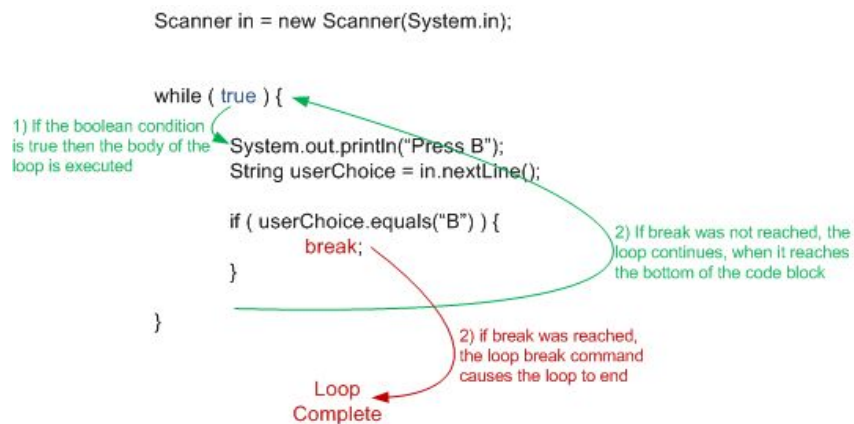
Explanation

Exiting by changing the condition



- If $a < 10$, then the loop body is executed
- a is changed by adding 1 at the end of the loop body
- The condition is checked again, if a is still < 10 , then the loop executes again
- This repeats until a is 10 making the boolean condition $(a < 10)$ FALSE, which causes the loop to exit

Exiting by break



- The boolean condition is checked, and if TRUE, then the code block is executed. In this case, the boolean condition is ALWAYS true.
- The user is asked to press the letter B
- If the user selection is not the letter B, then the end of the code block is reached and the loop repeats
- If the user select is the letter B, then the block of the if() statement is run, which contains the break command, which ends the loop

Choosing the correct Loop

There is no strict rule over which loop to use in any situation, however, different loops provide different functionality allowing the developer to choose which one fits their problem.

for loop	<p>Used when an incrementing/decrementing value is needed (<code>int i = 0</code>). For example, when the value is needed to access an Array by index, or when you need the loop to run a set number of times.</p> <p>The for loop is also the best choice if you need to change the value of the Array or Collection that you are looping through. For example, if you need to switch the position of items in the Array or remove items from a List.</p>
for each loop	<p>Requires an object with an iterator, such as an Array or Collection. Best used when you want to loop through the items from the first until the last in order, and you do not have a need to know the index or count. For example, looping through each item in a list in order to find a match and then exiting, or printing every item in an array in order.</p> <p>Since the for each loop does not allow the Array or Collection it is looping over to be changed, it is not a good choice if you need to change the collection or if you need to know the index of the current item. It is also not a good choice for a loop that you need to run x number of times, for example, if you want to print items 5 through 10 from an Array.</p>
while	<p>Best used when you have a complicated condition to exit the loop, and you need the loop to continue until that condition is met. For example, you want the user to take an action, and want to continue until they take that action.</p> <p>Since the while loop does not contain its own incrementer/decrementer and it does not automatically populate items from a Collection, it is not a good choice for looping through collections or arrays. Also, since it does not contain anyway to control the list by count, it is not a good choice if you want the list to run x number of times.</p>