Conditionals and Loops

Decision Making

Conditional statements are used to perform different actions based on different conditions.
The **if statement** is one of the most frequently used conditional statements.
If the **if** statement's condition expression evaluates to true, the block of code inside the **if** statement is executed. If the expression is found to be false, the first set of code after the end of the **if** statement (after the closing curly brace) is executed.
Syntax:

```
if (condition) {
  //Executes when the condition is true
}
```

Any of the following comparison operators may be used to form the condition:
< less than
> greater than
!= not equal to
== equal to
<= less than or equal to
>= greater than or equal to

For example:

```
int x = 7;
if(x < 42) {
  System.out.println("Hi");
}
```

Try It Yourself

Remember that you need to use two equal signs (==) to test for equality, since a single equal sign is the assignment operator.

if...else Statements

An **if** statement can be followed by an optional **else** statement, which executes when the condition evaluates to false.
For example:

```
int age = 30;

if (age < 16) {
  System.out.println("Too Young");
} else {
  System.out.println("Welcome!");
}
//Outputs "Welcome!"
```

Try It Yourself

As age equals 30, the condition in the **if** statement evaluates to false and the **else** statement is executed.

Nested if Statements

You can use one **if-else** statement inside another **if** or **else** statement.
For example:

```
int age = 25;
if(age > 0) {
  if(age > 16) {
    System.out.println("Welcome!");
  } else {
    System.out.println("Too Young");
  }
} else {
  System.out.println("Error");
}
//Outputs "Welcome!"
```

Try It Yourself

You can nest as many **if-else** statements as you want.

else if Statements

Instead of using nested **if-else** statements, you can use the **else if** statement to check multiple conditions.
For example:

```
int age = 25;

if(age <= 0) {
  System.out.println("Error");
} else if(age <= 16) {
  System.out.println("Too Young");
} else if(age < 100) {
  System.out.println("Welcome!");
} else {
  System.out.println("Really?");
}
//Outputs "Welcome!"
```

Try It Yourself

The code will check the condition to evaluate to true and execute the statements inside that block.

You can include as many **else if** statements as you need.

Logical Operators

Logical operators are used to combine multiple conditions.

Let's say you wanted your program to output "Welcome!" only when the variable **age** is greater than 18 and the variable **money** is greater than 500.
One way to accomplish this is to use nested **if** statements:

```
if (age > 18) {
  if (money > 500) {
    System.out.println("Welcome!");
  }
}
```

Try It Yourself

However, using the **AND** logical operator (**&&**) is a better way:

```
if (age > 18 && money > 500) {
  System.out.println("Welcome!");
}
```

Try It Yourself

If both operands of the AND operator are true, then the condition becomes true.

The OR Operator

The **OR** operator (||) checks if any one of the conditions is true.
The condition becomes true, if any one of the operands evaluates to true.
For example:

```
int age = 25;
int money = 100;

if (age > 18 || money > 500) {
  System.out.println("Welcome!");
}
//Outputs "Welcome!"
```

Try It Yourself

The code above will print "Welcome!" if age is greater than 18 **or** if money is greater than 500.

The **NOT** (**!**) logical operator is used to reverse the logical state of its operand. If a condition is true, the **NOT** logical operator will make it false.
Example:

```
int age = 25;
if(!(age > 18)) {
  System.out.println("Too Young");
} else {
  System.out.println("Welcome");
}
//Outputs "Welcome"
```

Try It Yourself

!(age > 18) reads as "if age is NOT greater than 18".

The switch Statement

A **switch** statement tests a variable for equality against a list of values. Each value is called a **case**, and the variable being switched on is checked for each case.
Syntax:

```
switch (expression) {
  case value1 :
    //Statements
    break; //optional
  case value2 :
    //Statements
    break; //optional
   //You can have any number of case statements.
   default : //Optional
     //Statements
}
```

- When the variable being switched on is equal to a **case**, the statements following that **case** will execute until a **break** statement is reached.
- When a **break** statement is reached, the **switch** terminates, and the flow of control jumps to the next line after the **switch** statement.
- Not every **case** needs to contain a **break**. If no **break** appears, the flow of control will fall through to subsequent cases until a **break** is reached.

The example below tests **day** against a set of values and prints a corresponding message.

```
int day = 3;

switch(day) {
 case 1:
   System.out.println("Monday");
   break;
 case 2:
   System.out.println("Tuesday");
   break;
 case 3:
   System.out.println("Wednesday");
   break;
}
// Outputs "Wednesday"
```

Try It Yourself

You can have any number of **case** statements within a **switch**. Each **case** is followed by the comparison value and a colon.

The default Statement

A switch statement can have an optional **default** case.
The **default** case can be used for performing a task when none of the cases is matched.

For example:

```
int day = 3;

switch(day) {
  case 6:
    System.out.println("Saturday");
    break;
  case 7:
    System.out.println("Sunday");
    break;
  default:
    System.out.println("Weekday");
}
// Outputs "Weekday"
```

Try It Yourself

No **break** is needed in the default case, as it is always the last statement in the switch.

while Loops

A **loop** statement allows to repeatedly execute a statement or group of statements.

A **while** loop statement repeatedly executes a target statement as long as a given condition is true.

Example:

```
int x = 3;

while(x > 0) {
  System.out.println(x);
  x--;
}
/*
Outputs
 3
 2
 1
*/
```

Try It Yourself

The **while** loops check for the condition x > 0. If it evaluates to true, it executes the statements within its body. Then it checks for the statement again and repeats.

Notice the statement x--. This decrements x each time the loop runs, and makes the loop stop when x reaches 0.
Without the statement, the loop would run forever.

while Loops

When the expression is tested and the result is false, the loop body is skipped and the first statement after the while loop is executed.

Example:

```
int x = 6;

while( x < 10 )
{
  System.out.println(x);
  x++;
}
System.out.println("Loop ended");

/*
6
7
8
9
Loop ended
*/
```

Try It Yourself

Notice that the last print method is out of the while scope.

for Loops

Another loop structure is the **for** loop. A for loop allows you to efficiently write a loop that needs to execute a specific number of times.
Syntax:

```
for (initialization; condition; increment/decrement) {
   statement(s)
}
```

Initialization: Expression executes only once during the beginning of loop
Condition: Is evaluated each time the loop iterates. The loop executes the statement repeatedly, until this condition returns false.
Increment/Decrement: Executes after each iteration of the loop.

The following example prints the numbers 1 through 5.

```
for(int x = 1; x <=5; x++) {
  System.out.println(x);
}

/* Outputs
1
2
3
4
5
*/
```

Try It Yourself

This initializes x to the value 1, and repeatedly prints the value of x, until the condition x<=5 becomes false. On each iteration, the statement x++ is executed, incrementing x by one.

> Notice the semicolon (;) after initialization and condition in the syntax.

---

for Loops

You can have any type of condition and any type of increment statements in the for loop.
The example below prints only the even values between 0 and 10:

```
for(int x=0; x<=10; x=x+2) {
  System.out.println(x);
}
/*
0
2
4
6
8
10
*/
```

Try It Yourself

> A **for** loop is best when the starting and ending numbers are known.

---

do...while Loops

A **do...while** loop is similar to a **while** loop, except that a **do...while** loop is guaranteed to execute at least one time.
Example:

```
int x = 1;
do {
  System.out.println(x);
  x++;
} while(x < 5);

/*
1
2
3
4
*/
```

Try It Yourself

Notice that the condition appears at the end of the loop, so the statements in the loop execute once before it is tested.
Even with a false condition, the code will run once. Example:

```
int x = 1;
do {
  System.out.println(x);
  x++;
} while(x < 0);

//Outputs 1
```

Try It Yourself

Notice that in do...while loops, the while is just the condition and doesn't have a body itself.

Loop Control Statements

The **break** and **continue** statements change the loop's execution flow.
The **break** statement terminates the loop and transfers execution to the statement immediately following the loop.
Example:

```
int x = 1;

while(x > 0) {
 System.out.println(x);
  if(x == 4) {
    break;
  }
  x++;
}

/* Outputs
1
2
3
4
*/
```

Try It Yourself

The **continue** statement causes the loop to skip the remainder of its body and then immediately retest its condition prior to reiterating. In other words, it makes the loop skip to its next iteration.
Example:

```
for(int x=10; x<=40; x=x+10) {
  if(x == 30) {
    continue;
  }
  System.out.println(x);
}
/* Outputs
  10
  20
  40
*/
```

Try It Yourself

As you can see, the above code skips the value of 30, as directed by the **continue** statement.

End.