

Selection Statements

by Sophia



WHAT'S COVERED

In this lesson, you will learn about conditional statements and how to use their outcomes in Java. Specifically, this lesson covers:

Table of Contents

- [1. if Statements](#)
- [2. Boolean Expressions](#)
- [3. else if Statements](#)
- [4. else Statements](#)

1. if Statements

When writing useful programs, the ability to test conditions and change the behavior of the program is typically required. To accomplish this, selection statements are used and provide the ability to do both of these. A **selection statement** is a statement that controls the flow of execution depending on some condition. The simplest form of a conditional statement is the if statement.



TRY IT

Directions: Type the following code into Replit in a file named If.java.

```
class If {  
    public static void main(String[] args) {  
        float temperature = 82.0f;  
        if(temperature > 0) {  
            System.out.println("Temperature is positive.");  
        }  
    }  
}
```

The output should look like this:

```
Console Shell
> java If.java
Temperature is positive.
> 
```

REFLECT

In the previous code, note how the condition being tested (`temperature > 0`, or "Is temperature greater than 0?") is in parentheses after the `if`. An `if` statement is always followed by parentheses. After the closing parenthesis, there is an opening curly bracket (`{`) that marks the start of the block of code that runs if the condition is true.

In this case, this statement runs if the temperature is greater than 0:

```
System.out.println("Temperature is positive.");
```

In the complete code listing, it is important to pay attention to the indentation. Since the `println()` call is controlled by the selection statement and comes after an open curly bracket, the line is indented one additional tab level. The closing curly bracket that marks the end of the block of code that is controlled by the `if()` is on the following line as back to the same indentation level as the `if()`. Such indentation is an important visual cue about the structure of the program.

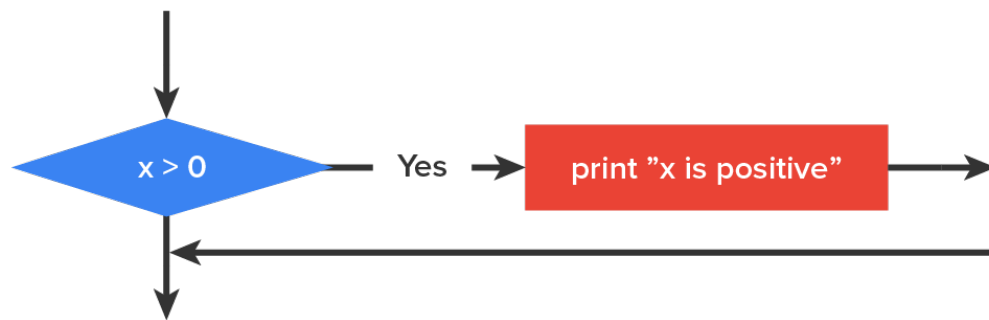
Since there is only one statement in the block of code governed by the `if()`, it would be okay to leave out the parentheses, like this:

```
class If {
    public static void main(String[] args) {
        float temperature = 82.0f;
        if(temperature > 0)
            // Note that the line below is indented further even without
            // curly brackets.
            System.out.println("Temperature is positive.");
    }
}
```

This saves a bit of typing and reduces what some see as clutter, but you are encouraged to leave the curly brackets in, since they become necessary as soon as any further statements are added to the block. Leaving out the brackets when they should be present can cause the program to behave strangely.

The boolean expression that follows the `if()` statement inside the parentheses is called the condition. A **condition** produces a boolean value when processed, and this boolean value determines which branch is executed. To finish this simple `if` statement, we place a pair of curly brackets after the parentheses that contain the condition. The statements that we want to execute when a condition is true go inside these curly brackets.

Here is a visual flowchart of what is happening in this statement.



If the logical condition is true (the temperature value is greater than 0), then the `System.out.println()` statement is run (82.0 is positive). If the logical condition is false, the indented statement is skipped. There is no limit on the number of statements that can appear in the block that is controlled by a selection statement but there should be at least one, since an empty block would do nothing



TERMS TO KNOW

Selection Statement

A selection statement is a statement that controls the flow of execution depending on some condition.

Condition

A condition is a boolean expression in a conditional statement that determines which branch is executed.

2. Boolean Expressions

Previously you learned that a condition is a boolean expression in a conditional statement, and that it determines which branch is executed. A **boolean expression** is either true or false. In the previous example, the greater than symbol (`>`) was used to check if the temperature was greater than 0 and if so, to print to the screen. To set up that boolean expression, use the greater than symbol as the comparison operator. A **comparison operator** is one of a group of operators that compares two values.

Let's list out and define the six (6) comparison operators.

Operator	Name	Example	Comment
<code>==</code>	Equal	<code>x == y</code>	x is equal to y
<code>!=</code>	Not equal	<code>x != y</code>	x is not equal to y
<code>></code>	Greater than	<code>x > y</code>	x is greater than y
<code><</code>	Less than	<code>x < y</code>	x is less than y
<code>>=</code>	Greater than or equal to	<code>x >= y</code>	x is greater than or equal to y
<code><=</code>	Less than or equal to	<code>x <= y</code>	x is less than or equal to y



THINK ABOUT IT

Although these operations are probably familiar to you, the Java symbols are different from the mathematical symbols for the same operations. A common error is to use a single equal sign (`=`) instead of a double equal

sign (==). Remember that = is an assignment operator and == is a comparison operator. There is no such thing as =< or =>. As we continue with this course, you will see these comparison operators used frequently.



TERMS TO KNOW

Boolean Expression

A boolean expression is an expression whose value, when evaluated, is either true or false.

Comparison Operator

A comparison operator is one of a group of operators that compares two values. These include ==, !=, >, <, >=, and <=.

3. else if Statements

Using if() statements on their own can be problematic because they each run independently of each other. This means that more than one code block could run at once in complex programs. Let's take a look at an example.

Consider this snippet of code:

```
class If {
    public static void main(String[] args) {
        int grade = 85;
        if(grade > 90) {
            System.out.println("You got an A");
        }
        if(grade > 80) {
            System.out.println("You got a B");
        }
        if(grade > 70) {
            System.out.println("You got a C");
        }
        if(grade > 60) {
            System.out.println("You got a D");
        }
        if(grade <= 60) {
            System.out.println("You got a F");
        }
    }
}
```

The intended output of the program should indicate that a score of 85 is a B. However, that is not exactly what happens when the program is run.

The resulting output actually looks like this:

Console Shell

```
> java If.java  
You got a B  
You got a C  
You got a D  
> 
```



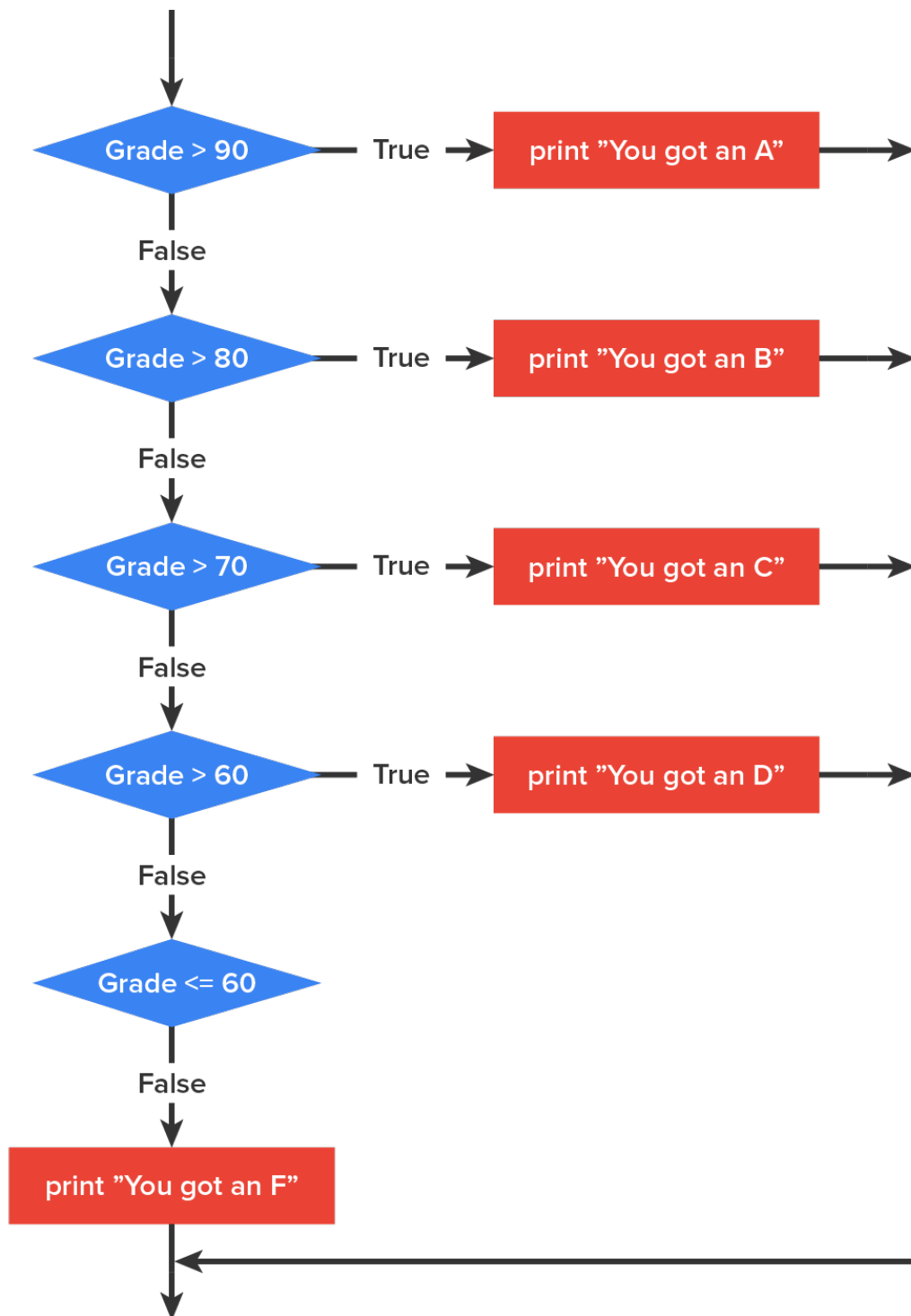
THINK ABOUT IT

Why did this happen? Since each of the `if()` statements runs independently, and they are not connected logically, each `if()` statement is tested separately. This means that more than one condition applied, and more than one block of code ran.

When there are more than two possibilities in a conditional statement, there needs to be two or more branches. One way to express a computation like this is found in a **chained conditional**. A chained conditional is a conditional statement with a series of alternative branches. The conditions are checked one at a time and the program exits the conditional statement if any branch is true.

In order to make our previous code a cohesive single set of statements, we can make a chained conditional by making use of the `else if()` statement. The `else if()` is associated as an existing `if()` statement. It states that if the current condition is not true, the program should move on to try the next condition in the chain. Once any conditional statement is true, only that branch is executed.

Let's see this in flowchart form first:



Consider this adjusted code:

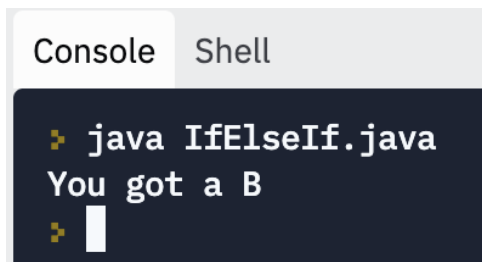
```
class IfElseIf {  
    public static void main(String[] args) {  
        int grade = 85;  
        if(grade > 90) {  
            System.out.println("You got an A");  
        }  
    }  
}
```

```

}
else if(grade > 80) {
    System.out.println("You got a B");
}
else if(grade > 70) {
    System.out.println("You got a C");
}
else if(grade > 60) {
    System.out.println("You got a D");
}
else if(grade <= 60) {
    System.out.println("You got a F");
}
}
}

```

The output looks like this:



```

Console Shell
❏ java IfElseIf.java
You got a B
❏

```

When running the code with `else if()` statements, the first `if()` statement came back false. Then the next condition (first `else if()`) was checked. That came back true ($85 > 80$), so we received that `else if()` statement's indented output as we expected.



TERM TO KNOW

Chained Conditional

A chained conditional is a conditional statement with a series of alternative branches.

4. else Statements

The `else()` statement can be viewed as a catchall, or default case. It runs if none of the `if()` or `else if()` statements apply.



HINT

There is never a condition in parentheses after an `else()` statement.

Using the same example, we can change it such that the last selection statement is an `else()`:

```

class IfElseIfElse {
    public static void main(String[] args) {
        int grade = 15;
    }
}

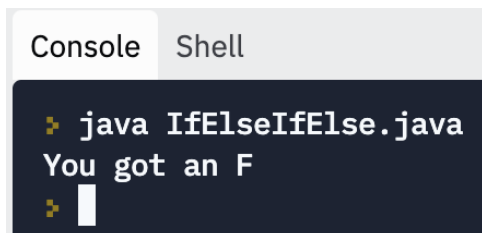
```

```

if(grade > 90) {
    System.out.println("You got an A");
}
else if(grade > 80) {
    System.out.println("You got a B");
}
else if(grade > 70) {
    System.out.println("You got a C");
}
else if(grade > 60) {
    System.out.println("You got a D");
}
else {
    System.out.println("You got an F");
}
}
}

```

The output looks like this:



```

Console Shell
> java IfElseIfElse.java
You got an F
>

```

Since the original grade was variable to 15 points, the output was an F, which should be expected. Each of the `else if()` statements proved false and the program continued until it reached the `else()` statement.



Directions: Use the first example with the temperature check to use all three statements to consider the scenario of the temperature being positive, 0, or negative. For a small adaptation, change the temperature variable to -5 for this test run.

```

class If {
    public static void main(String[] args) {
        int temperature = -5;
        if(temperature > 0) {
            System.out.println("Temperature is positive.");
        }
        else if(temperature == 0) {
            System.out.println("Temperature is 0.");
        }
        else {
            System.out.println("Temperature is negative.");
        }
    }
}

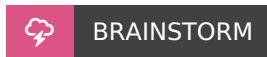
```



```
}
```

The output should look like this:

```
Console  Shell
> java If.java
Temperature is negative.
> 
```



Directions: You have seen examples using `if()`, `else if()` and `else` statements. Use Replit to test out some of the statements you have just learned.

SUMMARY

In this lesson, you learned about conditional statements which included the **if statement**. You discovered that conditional statements are also called compound statements, because they stretch across more than one line to include an indented block. You then explored the importance of indenting these blocks of code even though Java requires it to function correctly. You considered what **boolean expressions** are and that they can be used when creating conditional statements. Finally, you learned that the `if()` statement can be problematic since `if()` statements each run independently; therefore, it makes sense to create a chained conditional utilizing the **`if()` and `else()` statements** to ensure all conditions are tested.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

Boolean Expression

A boolean expression is an expression whose value, when evaluated, is either true or false.

Chained Conditional

A chained conditional is a conditional statement with a series of alternative branches.

Comparison Operator

A comparison operator is one of a group of operators that compares two values. These include `==`, `!=`, `>`, `<`, `>=`, and `<=`.

Condition

A condition is a boolean expression in a conditional statement that determines which branch is executed.

Selection Statement

A selection statement is a statement that controls the flow of execution depending on some condition.