

Flow of execution

Flow of execution

The *flow of execution* is the order in which program statements are executed.

Unless specified otherwise, program execution always begins from the first line. Statements are executed one at a time, top down.

Flow of execution

The *flow of execution* is the order in which program statements are executed.

Most programming languages support three basic *control structures* to control the flow of execution.

- Decision structures
- Repetition structures
- Functions/methods

Decision structures

Decision structures

Decision structures allow a program to execute different sequences of instructions for different cases.

The most basic form of decision structures is the **if** statement:

```
if (<condition>) {  
    <statements>  
}
```

Decision structures

In Java, **if** statements have the form:

```
if (<condition>) {  
    <statements>  
}
```

- <condition> is a *boolean* expression.
- <statements> can be any sequence of statements.

Decision structures

Curly braces are optional if there is only one statement under **if**:

```
if (<condition>)  
    <one statement>
```

or, equivalently, if there is only one semicolon under **if**.

Decision structures

A *statement* is the smallest unit of code that can be executed as a stand-alone instruction.

A semicolon *;* is placed to indicate the end of a statement.

Example.

```
if (temp < 32)  
    System.out.println("It's freezing outside.");
```

Boolean expressions?

An *expression* is a code fragment that has a value.

Example.

1823

Boolean expressions?

An *expression* is a code fragment that has a value.

Example.

$1823 + n$ (assuming n has a value)

Boolean expressions?

An *expression* is a code fragment that has a value.

Example.

`"Trinity College"`

Boolean expressions

A *boolean expression* is a code fragment that has a **true/false** value (i.e., its value is either **true** or **false**).

Example.

$0 < 1$ (whose value is **true**).

Boolean expressions

A *boolean expression* is a code fragment that has a **true/false** value (i.e., its value is either **true** or **false**).

Example.

`0 > 1` (whose value is **false**).

Boolean expressions

A *boolean expression* is a code fragment that has a **true/false** value (i.e., its value is either **true** or **false**).

Example.

`temp < 32` (assuming `temp` has a value).

Boolean expressions

A *boolean expression* is a code fragment that has a **true/false** value (i.e., its value is either **true** or **false**).

Example. Using `&&` for “and,”

```
(temp >= 60) && (temp <= 75)
```

Boolean expressions

A *boolean expression* is a code fragment that has a **true/false** value (i.e., its value is either **true** or **false**).

Example. Using `||` for “or,”

```
(temp < 60) || (temp > 75)
```

Boolean expressions

A *boolean expression* is a code fragment that has a **true/false** value (i.e., its value is either **true** or **false**).

Example. Using ! for “not,”

```
!(temp < 32)
```

Comparators

To compare two numbers, Java uses:

Mathematics

Java

=

==

≠

!=

<

<

>

>

≤

<=

≥

>=

Decision structures

The most basic form of decision structures is the **if** statement:

```
if (<condition>) {  
    <statements>  
}
```

This simply means that, if <condition> is true, execute <statements> (otherwise do nothing).

Decision structures

Example.

```
if (temp < 32)  
    System.out.println("It's freezing outside.");
```

This simply means that, if `temp < 32`, print
"It's freezing outside."

Decision structures

In Java, indentation is for readability only.

Example. What does the following output?

```
if (-1 >= 0)
    System.out.print("One ");
    System.out.print("Two ");
    System.out.print("Three ");
```

Output: Two Three

Decision structures

In Java, indentation is for readability only.

Example. What does the following output?

```
if (-1 >= 0) {  
    System.out.print("One ");  
System.out.print("Two ");  
System.out.print("Three ");  
}
```

Output: Nothing

Decision structures

Another important form of decision structures is the **if-else** statement:

```
if (<condition>) {  
    <statements1>  
}  
else {  
    <statements2>  
}
```

This means that, if <condition> is true, execute <statements1>, or else execute <statements2>.

Decision structures

Example.

```
if (temp < 32)
    System.out.println("It's freezing outside.");
else
    System.out.println("It's okay outside.");
```

This means that, if `temp < 32`, print "It's freezing outside." Otherwise, print "It's okay outside."

Decision structures

if-else statements can be nested:

```
if (grade >= 90)
    System.out.print('A');
else
    if (grade >= 80)
        System.out.print('B');
    else
        if (grade >= 70)
            ...
```

Decision structures

if-else statements can be nested:

```
if (grade >= 90)
    System.out.print('A');
else if (grade >= 80)
    System.out.print('B');
else if (grade >= 70)
    ...
```

Dangling **else**

What would this output?

```
int x = 7;  
if (x < 5)  
    System.out.print('A');  
    if (x < 8)  
        System.out.print('B');  
else  
    System.out.print('C');
```

Output: B

Dangling **else**

What would this output?

```
int x = 7;  
if (x < 5)  
    System.out.print('A');  
    if (x < 8)  
        System.out.print('B');  
else  
    System.out.print('C');
```

Output: B

Dangling **else**

What would this output?

```
int x = 7;  
if (x < 5)  
    System.out.print('A');  
if (x < 8)  
    System.out.print('B');  
else  
    System.out.print('C');
```

Output: B

Matching **else** with **if**

In a nested **if-else** statement

```
if (<condition>) {  
    <statements1>  
}  
else {  
    <statements2>  
}
```

else is always matched against the closest **if**.

Multiple choices

In Java, **switch** statements allow selecting one from multiple choices:

```
switch (<expression>) {  
    case <value1>: <statements1> break;  
    case <value2>: <statements2> break;  
    ...  
    default: <statements>  
}
```

Note. <expression> must be of primitive data type.

Multiple choices

In Java, **switch** statements allow selecting one from multiple choices:

```
switch (<expression>) {  
    case <value1>: <statements1> break;  
    case <value2>: <statements2> break;  
    ...  
    default: <statements>  
}
```

Note. **break** instructs Java to jump to the next statement block.

Multiple choices

In Java, **switch** statements allow selecting one from multiple choices:

```
switch (<expression>) {  
    case <value1>: <statements1> break;  
    case <value2>: <statements2> break;  
    ...  
    default: <statements>  
}
```

This is equivalent to ...

Multiple choices

```
if (<expression> == <value1>) {  
    <statements1>  
}  
else if (<expression == <value2>)  
    <statements2>  
}  
...  
else {  
    <statements>  
}
```

Multiple choices

Example.

```
switch (grade) {  
    case 'A': <statements1> break;  
    case 'B': <statements2> break;  
    ...  
    default: <statements>  
}
```

This is equivalent to ...

Multiple choices

```
if (grade == 'A') {  
    <statements1>  
}  
else if (grade == 'B') {  
    <statements2>  
}  
  
...  
else {  
    <statements>  
}
```

break

A *statement block* is a set of statements defined under a single construct (such as a class, a decision structure, a repetition structure, etc.).

In Java, a statement block begins with { and ends with }.

break

A *statement block* is a set of statements defined under a single construct (such as a class, a decision structure, a repetition structure, etc.).

In general, **break** instructs Java to jump to the next statement block *immediately*.