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

Repetition structures

Repetition structures

Repetition structures allow a program to repeat executing sequences of instructions under certain conditions.

The most basic form of repetition structures is the **while** statement:

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

Repetition structures

The most basic form of repetition structures is the **while** statement:

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

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

if vs. while

In the **if** statement

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

<statements> is executed *just once* in case <condition> is **true**.

if vs. **while**

In the **while** statement

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

<statements> is repeatedly executed *over and over again* as long as <condition> is **true**. That is, the loop terminates only when <condition> becomes **false**.

if vs. while

In the **while** statement

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

just like the **if** statement, if <condition> is **false** to start with, then <statements> is never executed.

Meaningful loops

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

In order for this loop to be useful, the value of `<condition>` must become from **`true`** to **`false`** at some point while `<statements>` is repeatedly executed.

Repetition structures

Example.

```
int i = 0;
while (i < 10) {
   System.out.print(i + " ");
   i = i + 1;
}
```

This simply means that, as long as `i < 10`, print the value of `i` and increment `i` by `1`.

Repetition structures

Example.

```
int i = 0;
while (i < 10) {
   System.out.print(i + " ");
   i++;
}
```

This simply means that, as long as `i < 10`, print the value of `i` and increment `i` by `1`.

Repetition structures

Example.

```
int i = 0;
while (i < 10) {
   System.out.print(i + " ");
   i++;
}
```

Output: `0 1 2 3 4 5 6 7 8 9`

Repetition structures

Example.

```
int i = 0;
while (i < 10) {
   System.out.print(i + " ");
}
```

Output: `0 0 0 0 0 0 0 0 0 0 0...`

Repetition structures

Example.

```
int i = 0;
while (i < 10) {
   System.out.print(i + " ");
   i++;
}
```

We need a statement that changes the value of `i < 10` from **`true`** to **`false`**.

Repetition structures

Example.

```
Scanner s = new Scanner(System.in);
int sum = 0;
System.out.print(”Enter a number: ”);
int n = s.nextInt();
while (n > 0) {
  sum = sum + n;
  System.out.print(”Enter a number: ”);
  n = s.nextInt();
}
System.out.print(”The sum is ” + sum + ”.”);
```

There is no statement that explicitly changes the value of `n > 0` from **`true`** to **`false`**.

Repetition structures

Example.

```
Scanner s = new Scanner(System.in);
int sum = 0;
System.out.print(”Enter a number: ”);
int n = s.nextInt();
while (n > 0) {
  sum = sum + n;
  System.out.print(”Enter a number: ”);
  n = s.nextInt();
}
System.out.print(”The sum is ” + sum + ”.”);
```

It is up to the user to decide when to terminate by entering a value.

Counted repetitions

Counted repetitions

Repetition structures allow a program to repeat executing sequences of instructions.

Another important form of repetition structures is the **for** statement:

```
for (int i = 0; i < n; i++) {
   <statements>
}
```

Counted repetitions

What does this do?

```
for (int i = 0; i < 10; i++)
   System.out.print(i + " ");
```

Output: 0 1 2 3 4 5 6 7 8 9