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.

Java provides three control structures to control the flow of execution.

• Decision structures
• Repetition structures
• Methods
Repetition structures
while loops

In Java, while statements have the form:

\[
\text{while (} <\text{condition}> \text{)} \{ \\
\quad <\text{statements}> \\
\}
\]

- \(<\text{condition}>\) is a boolean expression.
- \(<\text{statements}>\) can be any sequence of statements.
Meaningful while loops

while (<condition>) {
  <statements>
}

In order for this loop to have any value, the value of <condition> must become from true to false at some point while <statements> is repeatedly executed.
Meaningful while loops

Example.

```java
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.
Meaningful while loops

Example.

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

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

Example.

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

Output: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0...
Meaningful while loops

Example.

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

To terminate, we need a statement that changes i < 10 from true to false.
Meaningful while loops

Example.

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

To terminate, we need a statement that changes $i < 10$ from true to false.
Meaningful while loops

Example. This loop has no explicit statement to change \( n > 0 \) from true to false.

```java
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.println("The sum of is " + sum + ".");
```

March 13, 2019
In Java, for statements have the form:

```java
for (<initialization>; <condition>; <update>) {
    <statements>
}
```

- `<initialization>` is a statement to execute once before starting the loop.
- `<condition>` is a boolean expression.
- `<update>` is a statement to execute after each iteration.
for loops

```java
for (<initialization>; <condition>; <update>) {
    <statements>
}
```

is equivalent to:

```java
<initialization>;
while (<condition>) {
    <statements>
    <update>;
}
```

March 13, 2019
**for loops**

**Example.** A for loop to print numbers 0 through 99:

```
for (int n = 0; n < 100; n++)
    System.out.println(n);
```
for loops

for (int n = 0; n < 100; n++)
    System.out.println(n);

is equivalent to:

int n = 0;
while (n < 100) {
    System.out.println(n);
    n++;
}
do-while loops

do-while statements have the form:

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

• <condition> is evaluated at the end.
• It is useful when we always need to execute <statements> at least once.
**do-while loops**

**Example.** This can be simplified using a do-while loop.

```java
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.println("The sum of is " + sum + ".");
```

March 13, 2019
do-while loops

Example. This can be simplified using a do-while loop.

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