Control flow, part 1

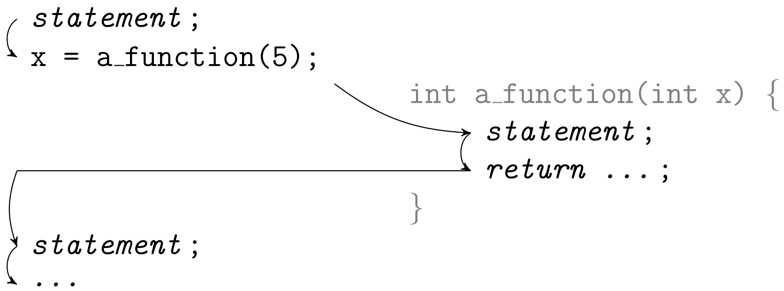
J5

- Control flow
- Branching
- Conditional expressions

Control flow

$$\begin{tikzcd}
\mathit{statement}\ ; \arrow[d, bend right] \\
\mathit{statement}\ ; \arrow[d, bend right] \\
\mathit{statement}\ ; \arrow[d, bend right] \\
\mathit{statement}\ ; \arrow[d, bend right] \\
\ldots
\end{tikzcd}$$

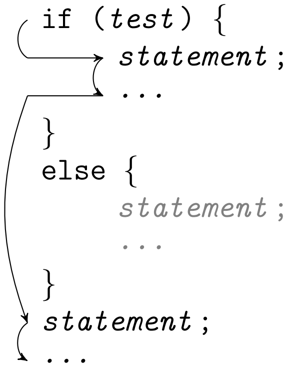
- A simple (imperative) program is a sequence of statements.
- In Java, statements end with a semicolon (`;`).
- Structured imperative programming: *sequence*, *branching* (*selection*), *iteration*.



```
statement;
x = a_function(5);
                              int a_function(int x) {
                                  statement;
                                  return ...;
                              }
statement;
...
```

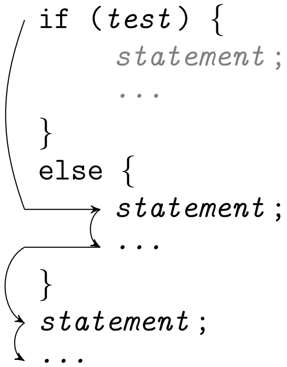
- Function calls “insert” the function body into this sequence, but the sequence remains invariably the same.
- Tip: `https://cscircles.cemc.uwaterloo.ca/java_visualize/`

Branching program flow



```
if (test) {
    statement;
    ...
}
else {
    statement;
    ...
}
statement;
...
```

OR



```
if (test) {
    statement;
    ...
}
else {
    statement;
    ...
}
statement;
...
```

- Depending on the outcome of a test, the program executes one of two branches.

The `if` statement

```
if (condition)
  <block>
```

```
if (condition)
  <block>
else
  <block>
```

- The condition is an expression of type `boolean`.
- A block is either a statement (ending with `;`) or a sequence of statements in braces (`{ }`).
  - Where have we seen a block before?
- The `if` statement is a statement, so can appear inside a block.

Coding
Max & Median

Other ways of branching in Java

- `switch` statement: case matching on the value of an expression. (`https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html`)
- Ternary operator `?` in conditional expression, e.g., `(x < 0 ? -1*x : x)` (`https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op2.html`)
- `switch` expression.

preview (C1): Recursion

- With functions, we can create arbitrarily deeply nested branching statements:

```
double solve(double l, double u, double x) {
  if (u - l < 0.000001)
    return l;
  else {
    double m = (l + u) / 2;
    if ((m * m) < x)
      return solve(m, u, x);
    else
      return solve(l, m, x);
  }
}
```