Control Structures: Conditionals

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria

Wolfgang.Schreiner@risc.jku.at
http://www.risc.jku.at

Control Structures

- Control flow: order in which statements are executed.
  - Default: from first statement of a method to the last; control flow forms a single path.
  - Control structures: statements that modify control flow.
- Conditional statements:
  - Selection of a statement from a set of candidates.
  - Control flow is split into multiple alternative paths.
  - `if`, `if-else`, `switch`.
- Iteration statements:
  - Repetition of a statement until the termination condition holds.
  - Control flow forms a loop that may be traversed multiple times.
  - `while`, `do`, `for`.
- Other statements: `{ }`, `break`, `continue`.

Assertions

```
assert boolean_expression;
assert boolean_expression : message;
```

- Statement that indicates knowledge about program.
  - We are convinced that the boolean expression evaluates to true.
  - If not, the assertion has failed and the program is aborted.
  - Examples:

```
x = 1;   assert x == 1;
x = a+b; assert x == a+b;
x = x+1; // x == x+1 does not hold!
```

  - Assertions are only checked, if the Java option -ea is used:

```
java -ea Main
```

Most valuable technique for debugging programs.

Block Statement

Grouping of multiple statements into one.

```
{
  statement;
  statement;
  ...
}
```

Block statement can appear wherever a single statement can appear.

Declarations in Block Statements

Variable/constant declarations are local to a block.

- Correct:

```
{
  int i = 1;
  System.out.println(i);
}
{
  int i = 2;
  System.out.println(i);
}
```

- Wrong:

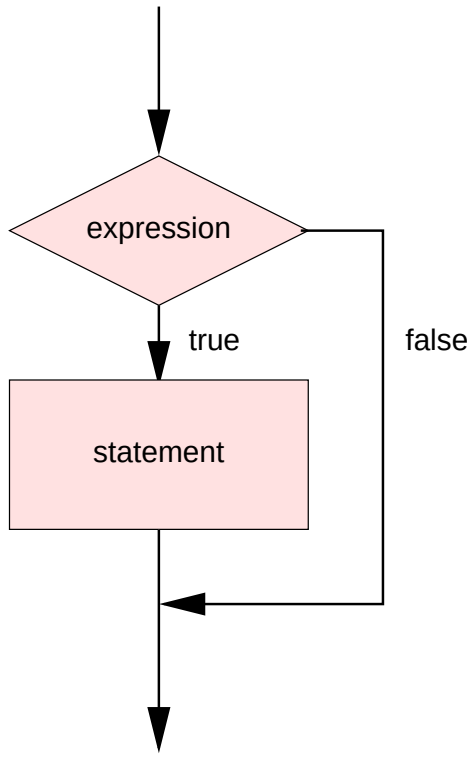
```
{
  int i = 1;
}
System.out.println(i);
```

```
Main.java:9: cannot resolve symbol
symbol  : variable i
location: class Main
   System.out.println(i);
```

The `if` Statement

```
if (boolean expression)
  statement;
```

- One-branch conditional
  - Boolean expression is evaluated.
  - If result is true, statement is executed.
  - If result is false, statement is not executed.
  - Execution continues with the statement after the conditional.



Conditional execution of a statement.

Examples

- If statement with a single statement.

```
if (count < max)
  count = count+1;
```

- If statement with a block:

```
if (count < max)
{
   count = count+1;
   sum = sum+count;
}
```

If branch may be single statement or a block.

Assertions

- At the beginning of an if branch, the condition holds:

```
if (expression)
{
  assert expression;
  ...
}
```

- Assertion holds after assignment for the old variable values.

```
if (count < max)
{
   assert count < max;     // if branch
   count = count+1;
   assert count-1 < max;   // count-1 = old count
   sum = sum+count;
}
```

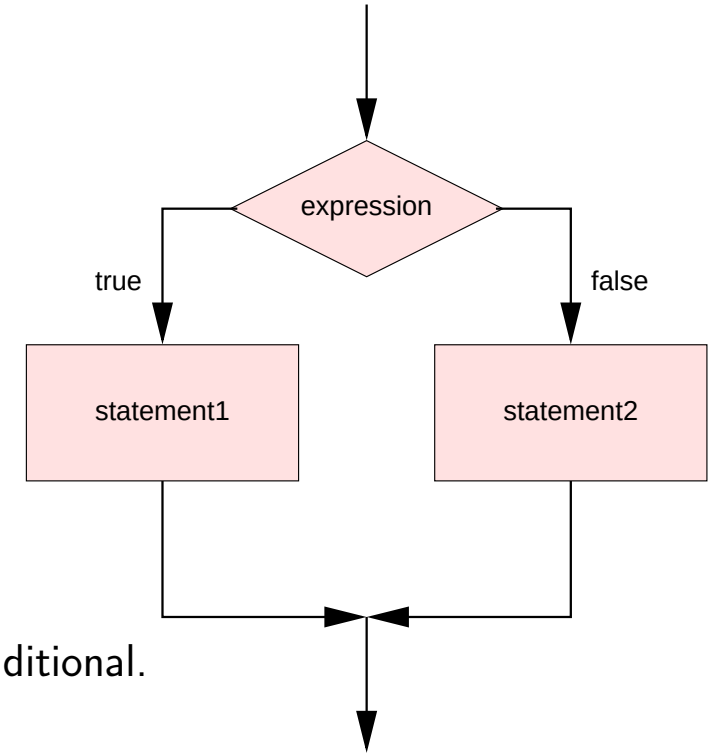
Assertions

- If assertion holds before if statement, it holds at start of branch.
  - May concatenate assertions with the && operator.

```
if (sum < max)
{
  assert sum < max;                          // first if branch
  if (count < max)
  {                                          // second branch with
     assert sum < max && count < max;        // previous condition
     count = count+1;
     assert sum < max && count-1 < max;      // count-1 == old count
     sum = sum+count;
     assert sum-count<max && count-1<max;    // sum-count == old sum
  }
}
```

The `if-else` Statement

```
if (boolean expression)
  statement1;
else
  statement2;
```



- Two-branch conditional.
  - Boolean expression is evaluated.
  - If result is true, first statement is executed.
  - If result is false, second statement is executed
  - Execution continues with the statement after the conditional.

Selection of a statement from two candidates.

If Statements with Blocks

```
if (boolean expression)
{
  statement;
  statement;
  ...
}
else
{
  statement;
  statement;
  ...
}

if (value % 2 == 0)
  value = value/2;
else
{
  value = value-1;
  count = count+1;
}
```

Dangling Else Problem

- Program:

```
if (a > b)
  if (a > 0) max = a;
else
  max = b;
```

- Compiler reads:

```
if (a > b)
  if (a > 0)
    max = a;
  else
    max = b;
```

- Solution:

```
if (a > b)
{
  if (a > 0) max = a;
}
else
  max = b;
```

Assertions

```
if (expression)
{
  assert expression;
  statement;
  ...
}
else
{
  assert !expression;
  statement;
  ...
}
```

At start of the second branch the boolean expression does not hold.

Minimum of Three Numbers

Variable min shall hold minimum of three variables a, b, and c.

```
if (a < b)
  if (a < c)
    min = a;
  else
    min = c;
else
  if (b < c)
    min = b;
  else
    min = c;
```

Are we sure that the program is correct?

Assertions

```
if (a < b)
{
  assert a < b;
  if (a < c)
  {
    assert a < b && a < c;
    min = a;
  }
  else
  {
    assert a < b && c <= a;
    min = c;
  }
}
...
```

Assertions

```
...
else
{
  assert b <= a;
  if (b < c)
  {
    assert b <= a && b < c;
    min = b;
  }
  else
  {
    assert b <= a && c <= b;
    min = c;
  }
}
```

Multi-Branch Conditionals

The `else` branch may be an `if-else` statement.

```
if (expression1)
  statement1;
else if (expression2)
  statement2;
else if (expression3)
  statement3;
...
else
  statementN;
```

- Multi-Branch conditional:
  - One boolean expression after the other is evaluated.
  - First branch is executed, for which evaluation returns true.
  - If no expression yields true, final `else` branch is executed (if any).

Example: Input Check

```
int i = Input.readInt();
if (Input.isOkay())
{
  handle input i
}
else if (Input.hasEnded())
{
  handle case that input stream has ended
}
else
{
  String error = Input.getError();
  handle error case
  Input.clearError();
}
```

Assertions

```
if (exp1)
{
  assert exp1;
  ...
}
else if (exp2)
{
  assert !exp1 && exp2;
  ...
}
...
else
{
  assert !exp1 && !exp2 && !exp3 && ... && !expN-1;
  ...
}
```

Example

```
if (value < 5)
  count = count+1;     // value < 5
else if (value < 10)
  count = count+2;     // 5 <= value < 10
else if (value < 20)
  count = count+3;     // 10 <= value < 20
else
  count = count+4;     // 20 <= value
```

Decomposition of variable range into four parts.

Multiple Case Tests

Frequently control flow is determined by value of a single expression:

```
if (expression == const1)
  stat1;
else if (expression == const2)
  stat2;
else if (expression == const3)
  stat3;
...
else
  statN;
```

Inefficient and cumbersome.

The `switch` Statement

```
switch (expression)
{
  case const1:
    stat1;
    break;
  case const2:
    stat2;
    break;
  case const3:
    stat3;
    break;
  ...
  default:
    statN;
}
```

The `switch` Statement

- Syntax:
  - Expression must be of integer or string type.
  - Case tags must be literals/constants of this type.
  - Same case tag must not appear in different branches.
  - Default statement is optional.
- Execution:
  - Expression is evaluated.
  - Statement whose tag matches value is executed.
  - If no tag matches, `default` statement (if any) is executed.
- Implementation:
  - Compiler constructs a table of jump addresses for each case constant.
  - At runtime, table lookup determines address of statement to execute.

Example

Multiple tags may prefix a branch.

```
switch (i%5) // value is in 0..4
{
  case 0:
    System.out.println("divided by 5");
    break;
  case 1: case 3:
    System.out.println("odd remainder");
    break;
  default:
    System.out.println("even remainder");
}
```

Value range is decomposed into three cases.