Software and Programming I

Loops and Expression Types

Roman Kontchakov / Carsten Fuhs

Birkbeck, University of London

Outline

- The `while`, `for` and `do` Loops
  - Sections 4.1, 4.3 and 4.4
- Variable Scope
  - Section 5.8
- Expressions and Types
- Operation Precedence

Boolean Variables and Operators

The Boolean type `boolean` has two values, `false` and `true`

three Boolean operators that combine conditions:

`&&` (and), `||` (or), `!` (not)

A	B	A && B
false	false	false
false	true	false
true	false	false
true	true	true

A	B	A || B
false	false	false
false	true	true
true	false	true
true	true	true

A	!A
false	true
true	false

NB: not `False` and `True`, and not `and`, `or` and `not` (like in Python)

If v Boolean Operations (1)

Can the following code be simplified (e.g., one `println`)?

```java
if (wavelength < 400) // IR
    System.out.println("invisible");
if (wavelength > 700) // UV
    System.out.println("invisible");
```

Yes:

```java
if (wavelength < 400 || wavelength > 700) // IR or UV
    System.out.println("invisible");
```

Avoid code duplication!

If v Boolean Operations (2)

Can the following code be simplified (e.g., one `if`)?

```
if (temp >= 0)
    if (temp <= 100)
        System.out.println("liquid");
```

Yes:

```
if (temp >= 0 && temp <= 100)
    System.out.println("liquid");
```

Avoid code duplication!

Boolean Operators

De Morgan's Laws: `!(A && B)` is equivalent to `!A || !B`

`!(A || B)` is equivalent to `!A && !B`

NB: Java does not use mathematical notation:

(in contrast to Python)

```
if (0 <= temp <= 100) // ERROR - not an expression
```

instead, use

```
if (0 <= temp && temp <= 100)
```

NB: and $\leq$ is NOT a Java operation

NB: do not confuse with & and |

Conditional Operator

conditional operator ? :

lets us write simple conditional statements as **expressions**

```
double abs = (x > 0) ? x : -x; // -x is unary minus
```

an expression

is equivalent to

```
double abs;
if (x > 0)
    abs = x;
else
    abs = -x;
```

Python:

```
while (balance < TARGET) :
      interest = balance * RATE / 100
      balance = balance + interest
      year = year + 1
```

The while Loop

the `while` loop executes instructions repeatedly while a condition is true

```
1 int year = 0;
2 double balance = 1000;
3 while (balance < TARGET) { // RATE = 3, TARGET = 1092
4       double interest = balance * RATE / 100;
5       balance = balance + interest;
6       year = year + 1;
7 }
```

year before	balance before	balance < TARGET	balance after	year after
0	1000.00	true	1030.00	1
1	1030.00	true	1060.90	2
2	1060.90	true	1092.73	3
3	1092.73	false	end of loop	

Loops and Assignments

```
int i = 6;
while (i >= 0) {
    System.out.println(i - 1);
    i = i - 2;
}
```

i before	i >= 0	i - 1	i - 2	i after
6	true	5	4	4
4	true	3	2	2
2	true	1	0	0
0	true	-1	-2	-2
-2	false	end of loop		

Assignment Operations

- shortcuts for increment and decrement:

  `i++;` is the same as `i = i + 1;`

  `i--;` is the same as `i = i - 1;`

- mixing operations and assignment:

  `i += 2;` is the same as `i = i + 2;`

  `i *= 2.5;` is the same as `i = i * 2.5;`

  `...`

- `+=`, etc. are of lowest precedence:

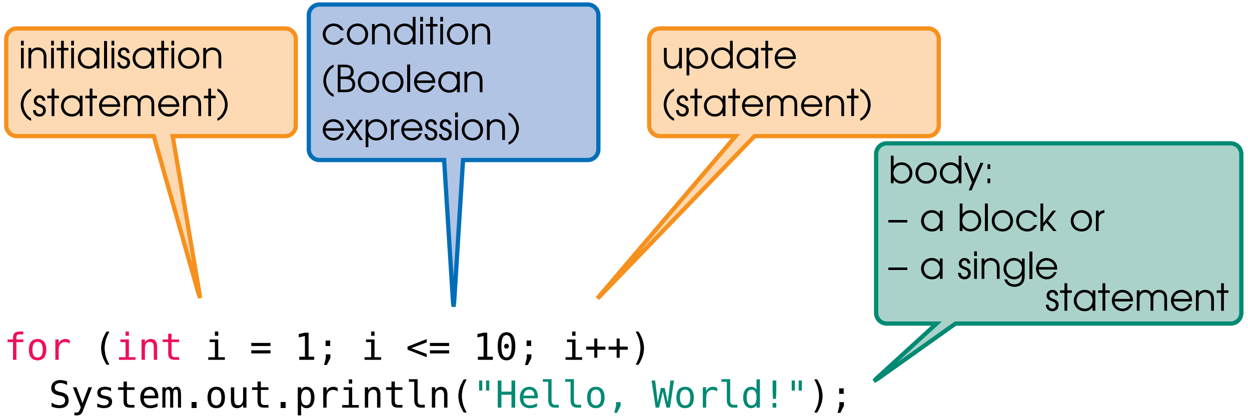
  `i /= 2 + 3;` is the same as `i = i / (2 + 3);`

NB: ONLY assignment operators **change** values of variables

(just writing `i - 1` does NOT change `i`!)

The for Loop

The `for` loop is normally used when instructions are executed repeatedly and a value runs from a starting point to an ending point with a constant increment (or decrement)



```java
for (int i = 1; i <= 10; i++)
  System.out.println("Hello, World!");
```

The for Loop: Example

```java
public class PrintHelloWorld {
  public static void main(String[] args) {
    for (int i = 1; i <= 10; i++)
      System.out.println("Hello, World!");
  }
}
```

Q: How many times is the phrase printed?

The for Loop: Example (cont.)

Q: How many times is the phrase printed?

```java
for (int i = 0; i < 10; i++)
  System.out.println("Hello, World!");
```

```java
for (int i = 0; i <= 10; i++)
  System.out.println("Hello, World!");
```

```java
for (int i = 10; i > 0; i--)
  System.out.println("Hello, World!");
```

The for Loop: Java v Python

Java

Python

```
for(int i = 0; i < 10; i++)
```

```
for i in range(0, 10)
```

loop body is run with `i` set to 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

```
for(int i = 0; i < 10; i += 2)
```

```
for i in range(0, 10, 2)
```

loop body is run with `i` set to 0, 2, 4, 6, 8

```
for(int i = 10; i > 0; i--)
```

```
for i in range(10, 0, -1)
```

loop body is run with `i` set to 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

NB: the `for` loop **does not** iterate over the letters in a string: `for(c : "hello world!")`

The for Loop

initialisation (statement)

condition (Boolean expression)

update (statement)

```
for (int k = 2; k <= 9; k++) {
    String s = s0;
    if (k % 2 == 1)
        s = s1;
    System.out.println(k + " is " + s);
}
```

. . . and the while Loop

initialisation (statement)

condition (Boolean expression)

update (statement)

```java
int k = 2;
while (k <= 9) {
     String s = s0;
     if (k % 2 == 1)
          s = s1;
     System.out.println(k + " is " + s);
     k++;
}
```

The do Loop

the do loop is appropriate when
the loop body must be executed at least once

```java
Scanner in = new Scanner(System.in);
int value;
do {
    System.out.println("Enter an integer < 100: ");
    value = in.nextInt();
} while (value >= 100);
```

NB: do not forget the semicolon ;
at the end of the statement

Scope of a Variable

- The scope of a variable is the part of the program in which it is **visible**
  - from its declaration until the end of the block, for a local variable
  - the entire method of a method's parameter variable
  - the `for` statement, for a local variable declared in the initialisation of a `for` statement
- Two variables can have the same name provided their scopes **do not overlap**

Scope: Example 1

Q: What is wrong here?

```
public static int sumOfSquares(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        int n = i * i;
        sum = sum + n;
    }
    return sum;
}
```

Scope: Example 2

Q: What is wrong here?

```
Scanner in = new Scanner(System.in);
do {
    System.out.println("Enter an integer < 100: ");
    int value = in.nextInt();
    System.out.println("Entered: " + value);
} while (value >= 100);
```

Boolean Expressions (1)

Suppose `a` is `5` and `b` is `4`. What is the value of `a > b` ?

```java
public static boolean greater(int a, int b) {
  return a > b; // returns true if a > b
}
```

```java
boolean found = false;
while (!found) {
  ... // do something
  if (...) // if the condition is met
    found = true;
  ... // do something else
}
```

Boolean Expressions (2)

Q: Why are the following methods not good code?

```
public static boolean greater2(int a, int b) {
  if (a > b)
    return true;
  else
    return false;
}
```

```
public static boolean greater3(int a, int b) {
  return (a > b) ? true : false;
}
```

```
public static boolean greater4(int a, int b) {
  return (a > b) == true; // never use != false either
}
```

Expressions

assignment statement $\underbrace{\texttt{cansPerPack}}_{\text{variable name}} = \underbrace{\texttt{8}}_{\text{expression}}\ ;$

an **expression** is a combination of
variable names, literals, method calls and **operators**

the **type** of an expression is known at compile-time:

- `8` is of type `int`
- `10.2` and `-12.3e-45` are of type `double`

  (**NB:** Java's `double` corresponds to Python's float)
- `"foo^=\nbar"` is of type `String`
- `false` and `true` are of type `boolean`

NB: types of variables are declared

Type Cast Operator

Q: What is wrong with the following?

```
int income = 20000;
int tax = income * 0.13;
```

corrected version:

```
int tax = (int) (income * 0.13);
```

NB: do not forget brackets

because type cast is of very high precedence

Q: Would the following work?

```
int tax = income * (int)0.13;
```

Type Cast Operator

Q: What is printed in the following fragment?

```java
int a = 5, b = 2;
System.out.println(a / b);
```

```java
int a = 5, b = 2;
System.out.println((double) a / b);
```

Operators and Expressions (1)

suppose $expr_1$ and $expr_2$ are expressions

of type `boolean`, `double`, `int`, or `String`

- the type of $expr_1 + expr_2$ is
  - `int` if the type of both $expr_1$ and $expr_2$ is `int`
  - `double` if the type of one of $expr_1$ or $expr_2$ is `double` and the other type is numerical, i.e., `int` or `double`
  - `String` if the type of one of $expr_1$ or $expr_2$ is `String`

otherwise, it is a compile-time error

Q: what is the type of `false + 1`?

- similar rules apply to `-`, `*`, `/` and `%`
  except they are **not** defined on `String`
  (unlike in Python, there is no string formatting operator `%` and no repetition `*`)

Operators and Expressions (2)

suppose $expr_1$ and $expr_2$ are expressions

- $expr_1$ < $expr_2$, $expr_1$ <= $expr_2$, $expr_1$ > $expr_2$ and $expr_1$ >= $expr_2$ are of type `boolean`

  both $expr_1$ and $expr_2$ must be of **numerical** datatypes

  compile-time error otherwise

Q: what is the type of `60 <= marks <= 69`?

- $expr_1$ `||` $expr_2$, $expr_1$ && $expr_2$ and `!` $expr_1$ are of type `boolean`

  both $expr_1$ and $expr_2$ must be of type `boolean`

  compile-time error otherwise

Q: what is the type of `60 <= marks && <= 69`?

Operation Precedence

- `()` method call — highest
- `!`, `(type)` type cast, `++`, `--` — unary
- `*`, `/`, `%` — multiplicative
- `+`, `-` — additive
- `<`, `<=`, `>=`, `>` — relational
- `==`, `!=` — equality
- `&&` — logical AND
- `||` — logical OR
- `?:` — conditional
- `=`, `+=`, `...` — assignments — lowest

NB: there is no Python's `**` (power) and `//` (floor division)

Operation Precedence

```
boolean f = 13 < floor - 1;
```

is the same as

```
boolean f = 13 < (floor - 1);
```

Suppose we have the declaration: `int a = 11;`

Evaluate the following expressions:

```
2 + a % 3
2 * 6 + a % 3 + 1 < 10 && a > 3
2 * 6 + a % 3 + 1 < 10 && !a > 3
2 + a / 3
2 + (double) a / 3
```

Loop Termination

Collatz conjecture Lothar Collatz, 1937

The sequence $a_{n+1} = \begin{cases} a_n/2, & \text{if } a_n \text{ is even} \\ 3a_n + 1, & \text{if } a_n \text{ is odd} \end{cases}$ eventually reaches 1

regardless of which positive integer a_0 is chosen

```
while (a > 1) {
    if (a % 2 == 0)
        a = a / 2;
    else
        a = 3 * a + 1;
}
```

Take Home Messages

- The `while` loop executes instructions repeatedly while a condition is true
- The `for` is used when a value runs from a starting point to an ending point with a constant increment
- Variables can have the same name provided their scopes do not overlap