Lecture 14:

for loops, break, continue and

Which loop to use?

Feb 13 2015
for loops

Syntax:
for (initial-action;
    loop-continuation-condition;
    action-after-each-iteration)
{
    // loop body;
    Statement(s);
}

Example:
int i;
for (i=0; i < 100; i++) {
    System.out.println("Welcome to Java");
}

Loop Continuation Condition?
true
false

Statement(s) (loop body)

Action-After-Each-Iteration

(i < 100)?
true
false

System.out.println("Welcome to Java");

i++
Note on for loops

- The initial-action in a for loop can be a list of zero or more comma-separated expressions.
- The action-after-each iteration in a for loop can be a list of zero or more comma-separated statements.

Therefore, the following two for loops are correct. They are rarely used in practice, however.

```java
for (int i = 1; i < 100; System.out.println(i++));

for (int i = 0, j = 0; (i + j < 10); i++, j++) {
    // Do something
}
```
Caution 1 on **for** loops

If the *loop-continuation-condition* in a **for** loop is omitted, it is implicitly **true**. Thus the statement given below in (a), which is an infinite loop, is “correct”. Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:

```plaintext
for (; ; ) {
   // Do something
}
(a)
```

Equivalent

```plaintext
while (true) {
   // Do something
}
(b)
```
Caution 2 on \texttt{for} loops

Adding a semicolon at the end of the \texttt{for} clause before the loop body is a common \textit{logic error}:

\begin{verbatim}
for (int i=0; i<10; i++);
{
    System.out.println("i is " + i);
}
\end{verbatim}
Semicolons and loops

Similarly, the following loop is also wrong:

```java
int i=0;
while (i < 10);{ //Logic error
    System.out.println("i is "+ i);
    i++;
}
```

In the case of the do-while loop, the semicolon is needed to end the loop.

```java
int i=0;
do {
    System.out.println("i is "+ i);
    i++;
} while (i<10); //Correct
```
**break** and **continue** statements

The **break** statement:

When the **break** statement is executed inside a loop-statement, the loop-statement is terminated immediately.

The execution of the program will continue with the statement *following the loop-statement*.

The **continue** statement

When the **continue** statement is executed inside a loop-statement, the program will skip over the remainder of the loop-body to the end of the loop body, *without exiting the loop completely*.
break and continue statements

- while loops
- do-while loops
- For loops
Which loop to use?

**while**, **do-while**, and **for**, are expressively equivalent; that is, you can write a loop in any of these three forms. Examples:

```
while (loop-continuation-condition) {
    // Loop body
}
```

```
for (initial-action;
     loop-continuation-condition;
     action-after-each-iteration) {
    // Loop body;
}
```

---

```
for ( ; loop-continuation-condition; ) {
    // Loop body
}
```

```
initial-action;
while (loop-continuation-condition) {
    // Loop body;
    action-after-each-iteration;
}
```
Which loop to use?

Use the one that is most intuitive and comfortable for you.

- In general, a **for** loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times.

- A **while** loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.

- A **do-while** loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.