# More Repetition

Chap. 8

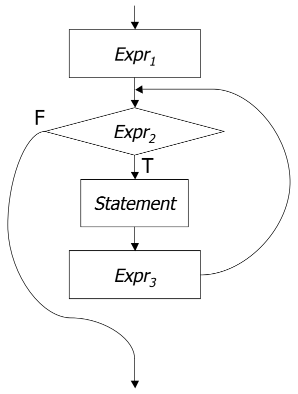
(Read §8.1-8.5)

1

## Review

We've seen that the *for loop* permits a statement to be executed repeatedly:

```
for (Expr1; Expr2; Expr3)
  Statement
```

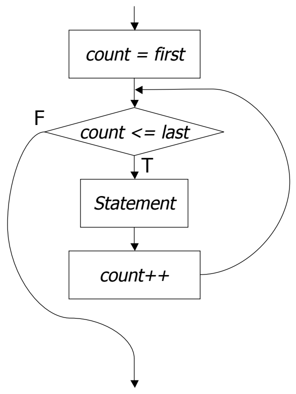


2

## A Counting Loop

The for loop is most commonly used to count from one value *first* to another value *last*:

```
for (int count = first;
         count <= last;
         count++)
   Statement
```



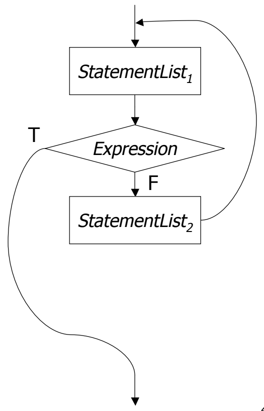
3

## Other Loops

Java also provides the forever loop: a for loop without expressions:

```
for (;;)
{
   StatementList1
   if (Expression) break;
   StatementList2
}
```

Repetition continues so long as *Expression* is false!

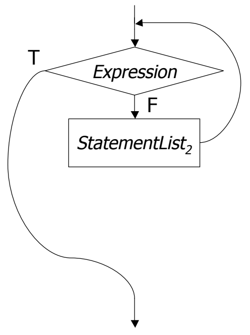


4

## Pretest Loops

If *StatementList$_1$* is omitted from a forever loop, we get a *test-at-the-top* or *pretest* loop:

```
for (;;)
{
  if (Expression) break;
  StatementList2
}
```



5

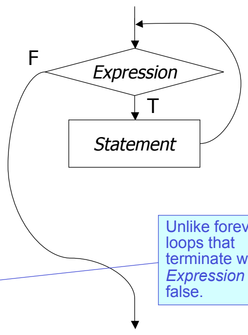
## The while Loop

For such situations, Java provides the more readable *while loop*, whose pattern is:

```
while (Expression)
   Statement
```

*Statement* can be either a single or a compound statement.

Repetition continues so long as *Expression* is true.



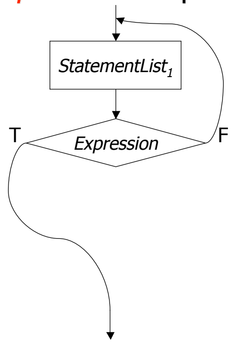
Unlike forever loops that terminate when *Expression* is false.

6

## Post-test Loops

If *StatementList$_2$* is omitted in a forever loop, we get a *test-at-the-bottom* or *post-test* loop:

```
for (;;)
{
  StatementList1
  if (Expression) break;
}
```



7

## The do Loop

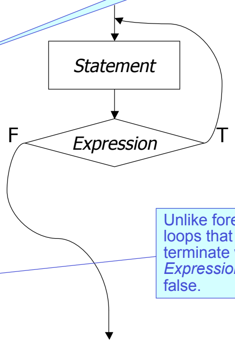
For such situations, Java provides the more readable *do loop*, whose pattern is:

NOTE!

```
do
   Statement
while (Expression);
```

*Statement* can be either a single or compound statement.

Repetition continues so long as *Expression* is true!



Unlike forever loops that terminate when *Expression* is false.

8

## Use in Input Loops

- Recall the two different types of input loops:
  - counting approach
  - sentinel approach
- Counting approach asked for number of inputs to be entered, used for loop.
  - Requires foreknowledge of how many inputs
- Sentinel approach looked for a special input value to signal termination.
  - Requires availability of appropriate sentinel value

9

## Query Approach

- Use a do loop
  - loop body always executed at least one time
- Query user at end of loop body
  - user response checked in loop condition

```
do
{
   // whatever …
   . . .
   theScreen.print("More inputs? (Y/N)  : ");
   response = theKeyboard.readChar();
}
while (response=='y' || response =='Y');
```

10

## Query Methods

- Note the extra code required in the loop
  - to make the query
  - to check the response
- This could be simplified by writing a method to ask the query, get the response, and return the boolean result. We might develop a class **`Query`** of various query methods.

```
do
{
  . . .
}
while (Query.moreValues());
```

11

## Choosing a Loop

With four loops at our disposal, how do we know which one to use?

- Use the *for* loop for counting problems.
- Design algorithms for non-counting problems using a general Loop statement, and see where it is appropriate for repetition to terminate:
  - If at its end, use the *do* loop
  - If at the loop's beginning, use the *while* loop
  - If in its middle, use the *forever* loop.

12

## Example

Write a function that given a positive int value, returns a string of equivalent digits.

Example: 123  "123" , 1  "1"

**Algorithm:**

```
0. Receive intVal.
1. Initialize stringVal to the empty string;
2. Loop
      a. Set intDigit to the remainder of intVal / 10.
      b. Attach intDigit at the beginning of stringVal.
      c. Set intVal to the quotient of intVal / 10.
   End loop
3. Return stringVal.
```

Question:
How/where should repetition terminate?

Our loop should terminate when *intVal* <= 0.

We should check this condition at the beginning, because if *intVal* is initially zero or negative, we do not want any of the statements in the loop's body to execute.

The *pretest* loop is thus the appropriate choice.

**Revised Algorithm:**

```
0. Receive intVal.
1. Initialize stringVal to the empty string;
2. Loop
      a. If (intVal <= 0) terminate repetition.
      b. Set intDigit to the remainder of intVal / 10.
      c. Attach intDigit at the beginning of stringVal
      d. Set intVal to the quotient of intVal / 10.
   End loop
3. Return stringVal.
```

## Coding

We thus choose the while loop for this problem:

```
public static String intToString(int intVal)
{
   String stringVal = "";
   int intDigit;

   while (intVal > 0)
   {
      intDigit = intVal % 10;
      stringVal = intDigit + stringVal;
      intVal /= 10;
   }

   return stringVal;
}
```

## Summary

The four Java loops provide very different behaviors:

- The *while* and *for* loops have their tests at the top. If the loop's condition is initially false, the body of the loop will not execute; this is called ***zero-trip behavior***.
- The *do* loop has its test at the bottom, so the body of the loop will execute at least once, regardless of the value of the loop's condition; this is called ***one-trip behavior.***
- The forever loop has its test in the middle. Statements preceding the test will always be executed at least once. Statements following the test will not be executed if the test causes repetition to terminate. This might be called ***half-trip behavior***.

Which loop you use to solve a given problem should be determined by your *algorithm* for that problem.