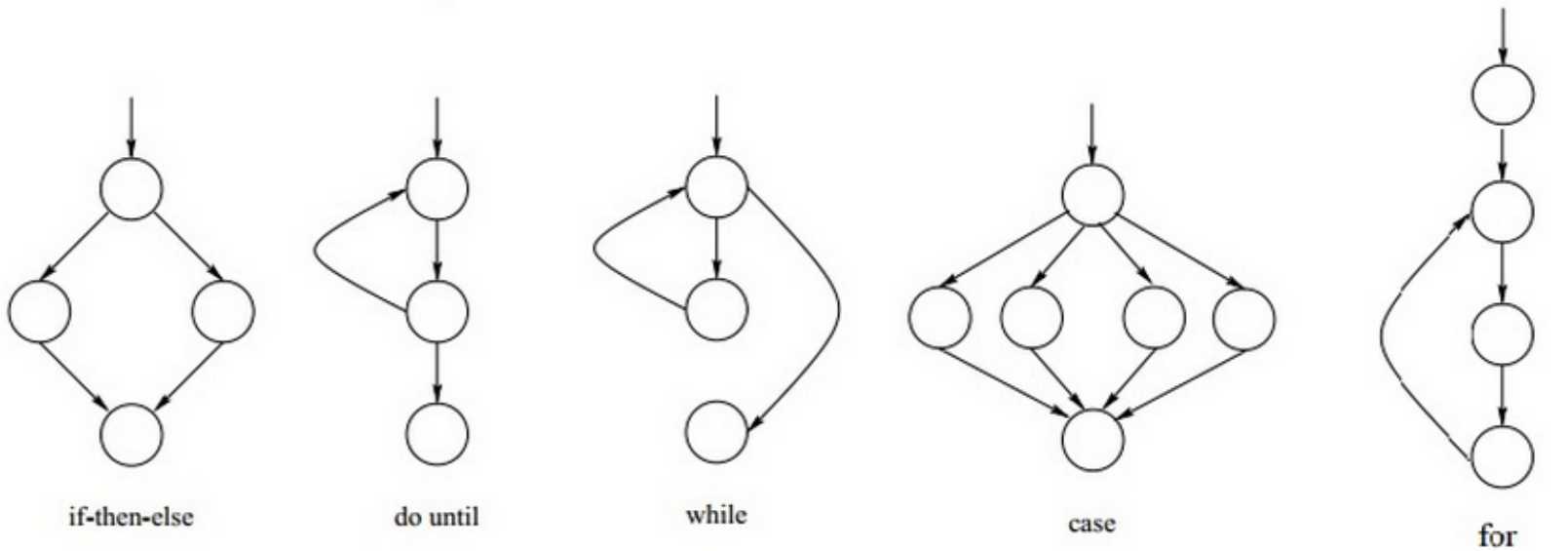
Repetitive Structures

Lecture 5
CGS 3416 Spring 2017

February 4, 2017

Control Flow

Control flow refers to the specification of the order in which the individual statements, instructions or function calls of an imperative program are executed or evaluated



Types of Control Flow

Flow of control through any given function is implemented with three basic types of control structures:

- **Sequential**: Default mode. Statements are executed line by line.
- **Selection**: Used for decisions, branching – choosing between 2 or more alternative paths.
  - if
  - if - else
  - switch
  - conditional statements
- **Repetition**: Used for looping – repeating a piece of code multiple times in a row.
  - while
  - do - while
  - for

The function construct, itself, forms another way to affect flow of control through a whole program. This will be discussed later in the course.

Repetition Statement

- Repetition statements are called *loops*, and are used to repeat the same code multiple times in succession.
- The number of repetitions is based on criteria defined in the loop structure, usually a true/false expression.
- The three loop structures in Java are:
  - while loops
  - do-while loops
  - for loops

  Three types of loops are not actually needed, but having the different forms is convenient.

while loop

- Format for a while loop

```
while (boolean_expression)
{
    statement1;
    statement2;
    // ...
    statementN;
}
```

- The *boolean_expression* in these formats is sometimes known as the **loop continuation condition**.
- The loop body must be a block, or a single statement (like with the if-statements).

do-while loop

- Format for a do-while loop

```
do
{
        statement1;
        statement2;
        // ...
        statementN;
} while (boolean_expression);
```

- The boolean expression is a test condition that is evaluated to decide whether the loop should repeat or not.
  - **true** means run the loop body again.
  - **false** means quit.
- The while and do/while loops both follow the same basic flowchart – the only exception is that:
  - In a while loop, the test expression is checked first
  - In a do/while loop, the loop "body" is executed first

Example

Add all the numbers from 1 through 50

Examples

```
// while loop example
//loop runs 50 times, condition checked 51 times
int i = 1, sum = 0;
while (i <= 50)
{
     sum += i;      // means:  sum = sum + i
     i++;           // means:  i = i + 1
 }

System.out.println("Sum of numbers from 1
          through 50 is " + sum);
```

Examples

```
// do-while loop example
//loop runs 50 times, condition checked 50 times
int i = 1, sum = 0;
do
{
     sum += i;      // means:  sum = sum + i
     i++;           // means:  i = i + 1
 }while (i <= 50);

System.out.println("Sum of numbers from 1
          through 50 is " + sum);
```

The for loop

The *for* loop is most convenient with counting loops – i.e. loops that are based on a counting variable, usually a known number of iterations.

Format of a for loop:

```
for (initialCondition; boolean_Expression;
                        iterativeStatement)
 {
     statement1;
     statement2;
     // ...

     statementN;
}
```

The for loop

How it works:

- The *initialCondition* runs once, at the start of the loop.
- The *boolean_Expression* is checked. (This is just like the expression in a while loop). If it's false, quit. If it's true, then:
  - Run the loop body
  - Run the *iterativeStatement*
  - Go back to the *boolean_Expression* step and repeat

Examples

Add all the numbers from 1 through 50

```
//loop runs 50 times, condition checked 51 times
int i, sum = 0;
for (i = 1; i <= 50; i++)
 {
     sum += i;
 }
System.out.println("Sum of numbers from 1
           through 50 is " + sum);
```

This loop prints “Hello” 10 times.

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

More Examples

Summing.Java
Counting.java
Product.java
SumEven.java

Nested loops

```
for (int i = 0; i <10; i++)
{
     Statements
     for (int j = 0; j <15; j++)
     {
          Statements
     }
     Statements;
     }
}
```

Examples

Prints a rectangle
Prints a triangle

Some notes on the for loop

It should be noted that if the control variable is declared inside the for header, it only has scope through the for loop's execution.

Once the loop is finished, the variable is out of scope:

```
for (int counter = 0; counter <10; counter++)
{
     // loop body
}

System.out.println(counter);
          // illegal.  counter out of scope
```

Some Notes on the for loop

This can be avoided by declaring the control variable before the loop itself.

```
int counter; // declaration of control variable

for (counter = 0; counter <10; counter++)
{
     // loop body
}

System.out.println(counter);
          // OK. counter is in scope
```

Some Notes on the for loop

For loops also do not have to count one-by-one, or even upward. Examples:

```
for (i = 100; i >0; i--)

for (c = 3; c <= 30; c+=4)
```

The first example gives a loop header that starts counting at 100 and decrements its control variable, counting down to 1 (and quitting when i reaches 0).

The second example shows a loop that begins counting at 3 and counts by 4's (the second value of c will be 7, etc).

break and continue

- These statements can be used to alter the flow of control in loops, although they are not specifically *needed*. (Any loop can be made to exit by writing an appropriate *test expression*).
- **break**: This causes immediate exit from any loop (as well as from switch blocks).
- **continue**: When used in a loop, this statement causes the current loop iteration to end, but the loop then moves on to the next step.
  - In a while or do-while loop, the rest of the loop body is skipped, and execution moves on to the *test condition*.
  - In a for loop, the rest of the loop body is skipped, and execution moves on to the *iterative statement*.