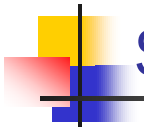
Introduction to Programming

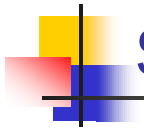
5: Program Control Structures:

Part 2 – Iteration



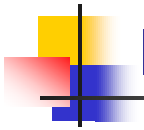
Structured Programming

- We have seen that structured programming imposes structure on the program code
  - Any section of code should have one entry point and one exit point
  - General flow of a structure (e.g. a loop) should always be top to bottom
- In Java, blocks are used to group statements together so that they are treated as a single statement in control structures such as the branches of an if statement



Structured Programming

- In structured programming the flow of control is limited to three forms
  - Sequence
    - Statements are executed in the order that they occur in the text
  - Selection
    - Involving a decision of whether a statement is executed or not, or choosing which statement to execute from a set of alternatives
  - Iteration
    - Repeatedly executing a statement - the subject of the bulk of this lecture



Iteration

- Computers are capable of performing repetitive tasks accurately and tirelessly
- Many applications involve the same steps being applied to each element in a large set of data
  - Charles Babbage's "Analytical Engine", designed in the 1830s, was the first description of a programmable computer
    - intended to calculate the thousands of entries for tables of mathematical functions such as logarithms (a very repetitive task and the tables of his day, produced by manual calculations, contained many errors)

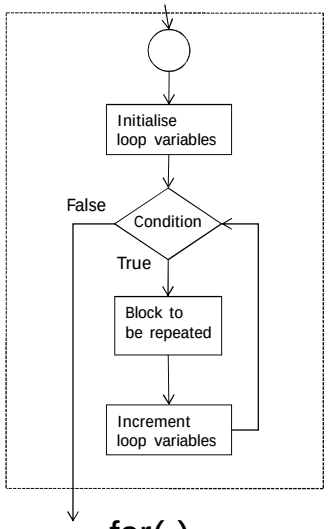
Iteration continued

- **Iteration** provides for control flow in which the program executes the same instructions over and over again (so also sometimes called *repetition*)
- Achieved using a structure known as a **loop**

Iteration in Java

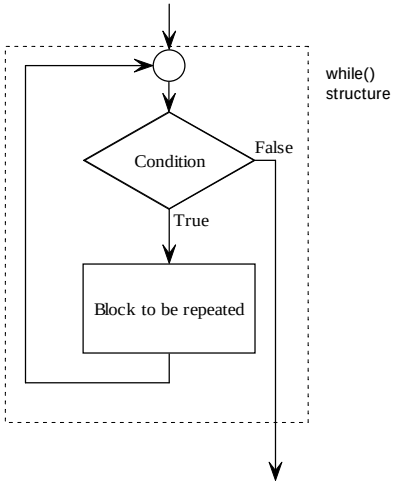
- Java provides three kinds of loops
  - The for() loop
    - Intended as a counting loop which executes a fixed number of times
    - There is also a variant called a for-each loop where the number of times is set by the size of a collection of data
  - The while() loop
    - A conditional loop which is only entered if the condition is true at the beginning of the loop
  - The do...while() loop
    - A conditional loop which is only repeated if the condition is true at the end of the loop

Counting loop

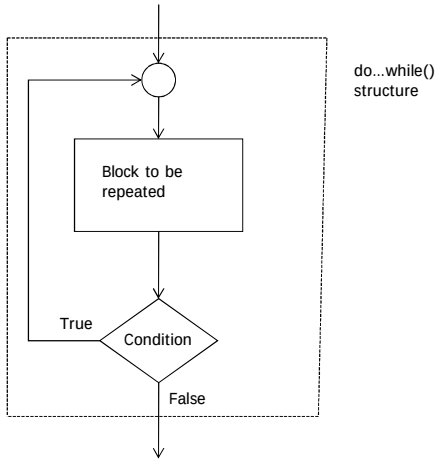


for( )

Conditional Loops



while( )



do...while( )

Loops - choices

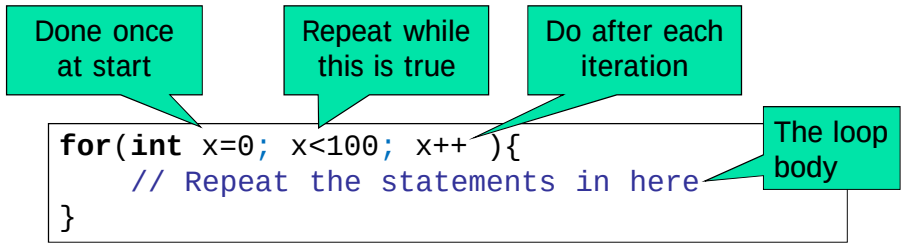
- Definite or Indefinite?
  - Can we work out how many iterations there will be at the entry point of a loop?
    - Yes – a definite loop (typically a **for**( ) loop)
    - No – an indefinite loop
- Indefinite loops
  - Zero or more iterations
    - Exit point at the start (a **while**( ) loop)
  - One or more iteration
    - Exit point at the end (a **do**…**while**( ) loop)
  - Sometimes, n½ iterations (not covered today)
    - Exit point in the middle – either **while**(true) or **for**(;;), with a **break**; within the loop body.

Definite versus Indefinite

- Prefer a solution that uses definite loops over one that uses indefinite loops if you can structure the problem solution that way
  - Only a definite loop is guaranteed not to loop forever!
- Often you have no choice but to use an indefinite loop as the problem does not allow you to predict how many iterations are needed to solve it

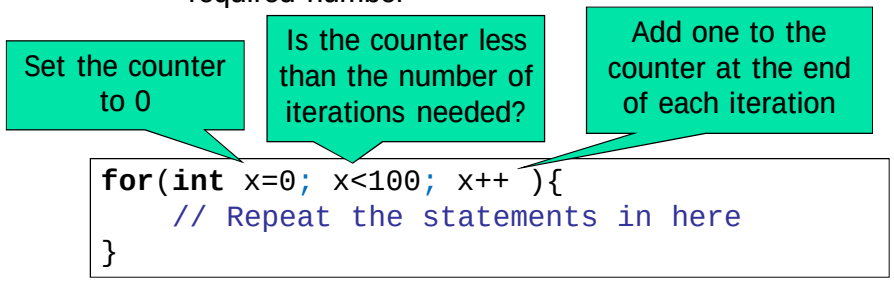
for( ) loops

- The standard for() loop structure has
  - A part that is done at the start before the first iteration of the loop
  - A **boolean** condition that is checked *before* each iteration of the loop, and the loop is only entered if the condition is true
  - A part that is done *after* every iteration of the loop
  - A loop body - the statements that are to be repeated



for( ) loops continued

- For() loops are sometimes called counting loops because they typically execute a fixed (definite) number of times and so need to count their iterations
  - Before the first loop set the count of iterations to zero
  - After each iteration increment the count of iterations
  - The condition tests whether the count has reached the required number



The initialisation part

- The part done just once before the start of the loop is used to declare and initialise the loop variables
  - Any variables you declare are local to the loop
  - If you declare more than one they must all be the same type
  - Can also initialise variables declared previously

The update or action part

- The part done after every iteration of the loop is intended to update the loop variables
  - This is the last thing that happens in the loop before the condition is checked again
  - You can put other statements in this part in addition to statements that update the loop variables, so sometimes called the *action* part

On for() loops generally

- The for() loop is a very common structure in Java programs
- Has a wide variety of uses!
- The next few slides give some examples

E.g. processing a range

- A for() loop can be useful if you want to process every number in a range
  - Example: calculate N!
    - factorial N = 1x2x3x..(N-1)xN, where N is a positive integer
    - grows very quickly so will use long instead of int for the answer
      - Might still go out of range!
  - Note that this is a for() loop which is definite but where the number of iterations is not known till runtime (depends on the value entered for n)

```
int n = TextIO.getlnInt();
long factorialN = 1;
for(int i = 2; i <= n; i++){
   factorialN = factorialN * i;
}
```

E.g. counting backwards

- A for() loop is good for counting backwards

```
for(int bottles = 10; bottles > 0; bottles--) {
   for(int i=0; i < 2; i++) {
       TextIO.put(bottles + " green bottles ");
       TextIO.putln("hanging on the wall.");
   }
   TextIO.put("And if 1 green bottle should ");
   TextIO.putln("accidentally fall...");
   TextIO.put("There will be " + (bottles-1));
   TextIO.put(" green bottles hanging ");
   TextIO.putln("on the wall.");
}
```


Notes on the examples - 1

- Nesting for() loops, as in the 10 green bottles example, is common
  - This was a simple case as the inner for() loop always executes just twice – often the loop variable in the inner loop is initialised with the value of the loop variable in the outer loop (e.g. if we had chosen “1 man went to mow” as the song we would have done this – try it!)

Notes on the examples - 2

- In the examples, have declared the loop variables in the loop start part so that they are destroyed on exit from the loop
  - Start part of loop can initialise variables previously declared (so they can be accessed after the loop exits) but in general it is good practice to limit your loop variables to the loop body when you can

Indefinite loops

- In the next few slides we will look at the conditional loops in Java
  - The while() loop
  - The do... while() loop
- These are generally used when we have an indefinite loop and do not know how often the loop body will need to be repeated

while() loops

- The most general type of loop
- An example problem you can solve with a while() loop
  - Find the greatest common divisor (gcd) of two positive integers (the largest integer that exactly divides them both)
  - If *a* and *b* are the two integers and their gcd is *d*, and *a* > *b*, then (*a* - *b*) will also have a gcd of *d* with *a* and *b* (division is equivalent to repeated subtraction)
    - This gives an algorithm where we repeatedly replace the larger of *a* and *b* with their difference until *a* and *b* are the same – the number we obtain is the greatest common divisor of the two numbers.

```
while (a != b) {
   if (a > b) { a = a – b; } else { b = b – a; }
}
TextIO.putln("The greatest common divisor is " + a);
```


while() loops

- A while() loop begins by testing its boolean condition
  - The condition is tested at the beginning of every iteration of the loop
  - If the condition is true the loop body is entered
  - If the condition is false the loop terminates (that is, the loop body does not execute and control passes to the first statement after the loop body)
- To avoid a while() loop executing over and over again forever, something in the loop body has to ensure that eventually the condition becomes false
- As the condition could be false the first time it is tested a while() loop may iterate zero times

Another example while() loop

- We looked at a program in lecture 2 that calculates interest after one year (from Eck's book)
- Can use a while() loop to calculate how many years it will take to reach a target amount

```
double goal = TextIO.getlnDouble(); // retirement sum
int years = 0;
while (balance < goal)  {
       balance += annualDeposit;
       double interest = balance*interestRate/100;
       balance += interest;
       years++;
}
TextIO.putln("You can retire in " + years + " years");
```

Thanks to Cay Horstmann for this example!

do...while() loops

- A do...while() loop begins by executing its loop body
  - The loop condition is tested at the end of every iteration of the loop
  - If the condition is true the loop body is repeated
  - If the condition is false the loop terminates
- As with a while() loop, to avoid a do...while() loop executing over and over again forever, some statement in the loop body has to ensure that eventually the condition becomes false
- As the loop condition is not evaluated till after the loop body has executed a do...while() loop body must be executed at least once

A do...while() loop

- One common usage of an indefinite do...while() loop is when processing user input until a given response is received, e.g. from a menu or a prompt:

```
char input;
do {   // repeat until user types 'N'
       // do at least once
       ...
       TextIO.put(" Continue (Y or N)? ");
       input = TextIO.getlnChar();
} while (input == 'Y' || input == 'y');  // note semicolon
```


TextIO.getlnBoolean()

- Note that instead of using a **char** for input one can often use a **boolean** as the **TextIO.getlnBoolean()** and getBoolean() methods return **true** if the user types “Y”, “Yes”, “y”, “yes” (as well as if they type “true” etc) and **false** if the user types “N”, “No”, “n” or “no” (as well as if they type “false” etc).

```
boolean continuing;
do {   // repeat until user types 'N'
       // do at least once
       ...
       TextIO.put(" Continue (Y or N)? ");
       continuing = TextIO.getlnBoolean();
} while (continuing);  // note semicolon
```


A note on semicolons...

- In Java, a semicolon (;) is a statement terminator
  - Terminates ***executable*** statements
  - No semicolon after a block
  - Control structures are *part* of statement, do NOT insert a semicolon after the control part
    - Except do...while() IS terminated by semicolon (as the **while** condition terminates the statement)
- Semicolon on its own is an empty statement

Infinite loop... (never stops)

Don’t do this!!

```
double goal = TextIO.getlnDouble() ; // retirement sum
while( balance<goal ) ; { // block is not in the loop!
       balance = balance + annualDeposit;
       double interest = balance*interestRate/100;
       balance = balance + interest;
       years++;
}
TextIO.putln("You can retire in " + years + " years");
```

The **break** statement

- You can exit a loop body anywhere using a break statement
- This should be conditional, that is, the break will be part of an if statement and only executes if the condition is true
- Using this statement allows you to construct a loop where the exit is not at the beginning or the end of the loop but somewhere in the loop body (*N½ loops*)
  - Should still aim to follow the rule of having just one exit point in the loop!
  - Will come back to this in a future lecture.

Summary on iteration

- Loops can be definite or indefinite depending on whether, before the first entry to the loop, one knows how many times the loop will execute
- Typically one should use for() loops for definite loops
- The while() and do...while() loops are appropriate for indefinite loops
  - while() if zero iterations are possible
  - do...while() if at least one iteration is required
  - An indefinite loop goes on repeating indefinitely
    - an infinite loop if repeats forever!
- Will meet many examples of programs with loops!

Reading for next week

- Eck
  - Finish chapter 3
  - There are some good programming examples in this chapter, study these and try them in Eclipse
    - 3N+1 problem
    - Section 3.6.2 on menus and switch statements
      - You will need to write a program with a menu for the project so you really should study this!

Questions?