Lecture 15

- Covers
  - Looping statements
  - The for statement
  - The break statement in loops
  - The exit( ) method
- Reading: Savitch 3.2

- The for statement

while loop revisited (Example of a very common form of while loop)

```
total = 0;
count = 1;
while (count < 11)
{
    total += count;
    count++;
}
System.out.println("The sum of the numbers from 1 to 10 is "
                   + total);
```

initialisation

loop terminator

update actions

Note: It'd better to replace "count < 11" by "count <= 10"

The for loop

- So common is the form of the previous loop that there is a special statement to express it.
- It is the for loop

The for loop

- The for loop is another type of looping statement in Java
- It is similar to the while statement as the body of a for loop is executed zero or more times, i.e. the condition is tested first
- The initialisation action, condition and update actions are placed differently in a for loop

The for loop

- The previous example can be written as a for loop:

```
total = 0;
for (count = 1; count < 11; count++)
{
    total += count;
}
System.out.println("The sum of the numbers from 1 to 10 is "
                                + total);
```

The for loop

- Or quite often as follows (where we declare the count variable inside the for loop):

```
total = 0;
for (int count = 1; count < 11; count++)
{
    total += count;
}
System.out.println("The sum of the numbers from 1 to 10 is "
                                + total);
```

The for loop flow of control

- The previous loop works as follows
  - First, variable count is initialised to 1
  - Then the condition count < 11 is checked, and the loop is repeated so long as the condition is satisfied
  - And at the end of each iteration (repetition) the value of count is incremented by 1

The for loop

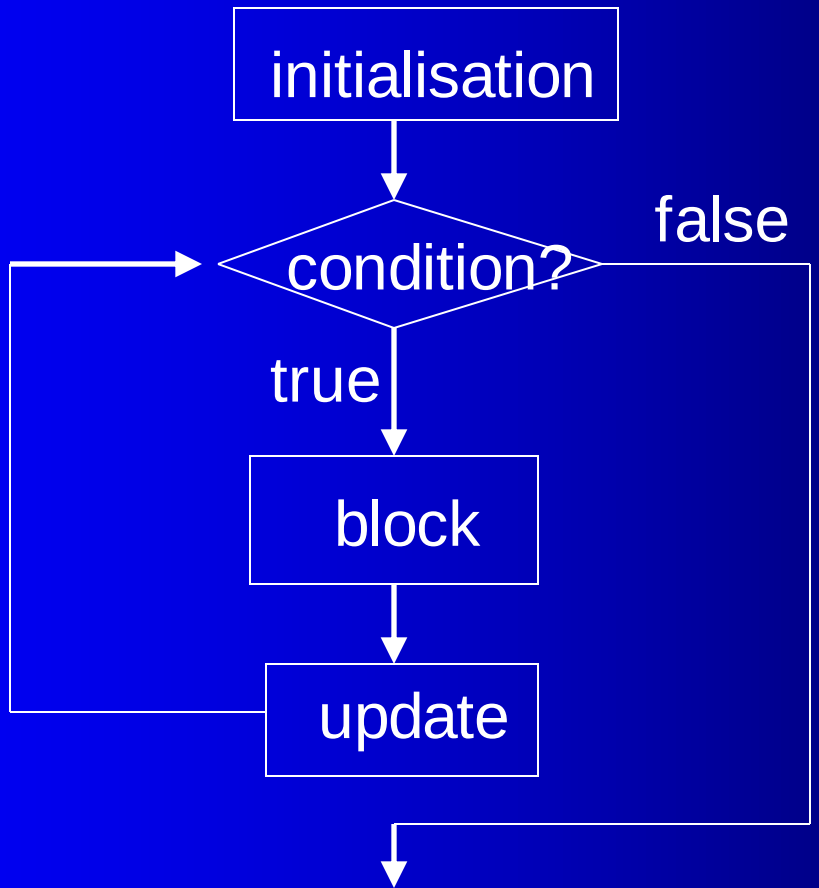
- General form

```
for (initialisation; termination-expression; update-actions)
{
    statement-1;
    statement-2;
    .
    .
    statement-n;
}
```

Syntax and behaviour

for (<initialisation> ; <condition> ; <update>)

< block >



The for loop

```
for (count = 1, total = 0; count < 11; count++)
{
      total += count;
}
System.out.println("The sum of the numbers from 1 to 10 is "
                                 + total);
```

- We can use a comma (,) in a for loop to separate initialisation or update actions

The for loop

```
for (count = 1, total = 0; count < 11; total += count, count++);

System.out.println("The sum of the numbers from 1 to 10 is "
                   + total);
```

- It is good style to restrict the update action to updating a single counter variable
- Keep your code simple and easy to read and understand

Beware of pitfalls

- An extra semi-colon

```
total = 0;
for (count = 1; count < 11; count++);
{
    total += count;
}
System.out.println("The sum of the numbers from 1 to 10 is "
                   + total);
```

Main use of the for loop

- for loops are generally used whenever we require a count-controlled loop
  - We want to repeat the loop a pre-determined number of times
  - And we use a “count” variable to achieve that

Example

- Problem
  - An investment account has the initial balance of $10,000
  - What is the balance after 20 years if the interest is compounded yearly at the rate of 5%?

Solution

```
double initBalance = 10000;
final double RATE = 0.05;
final int YEARS = 20;

double balance = initBalance;
for(  int year = 1; year <= YEARS; year++)
{
    double interest = balance * RATE;
    balance = balance + interest;
}
System.out.println("Balance after 20 years is " + balance);
```

Blocks revisited

- A variable defined in a block is available only within that block
  - In the previous example, variable interest is available only within the block
  - It is not available outside that block
  - It exists only within that block
  - We say its scope is the block
- A variable defined in the header of a for loop is available only within that loop
  - E.g. variable year is available only within the for loop

Class exercise

- Write a for loop that will display the integers from 20 down to 1

- The break statement

The break statement

- Terminating a loop when the condition becomes false is not the only way it can end
- The break statement can be used inside any type of loop statement to end the loop

The break statement (an example)

```
int counter = 0;
do
{
      System.out.println("Enter -1 to terminate, "
                              + " any other integer to loop: ");
      int answer = keyboard.nextInt( );
      if (answer == -1)
            break;
      ++counter;
      System.out.println(answer);
} while (counter < 20);
```

* Why would it be better to declare answer outside the loop?

Unstructured programming

- Using break statements within loops departs from the idea of structured programming
- Usually it is better to rewrite the loop and avoid using the break statement
- Only use breaks for truly exceptional circumstances

Structured programming

- Modules with one entry point and one exit point without unconditional transfer of control
- Only need to use the 3 control structures
  - Sequence
  - Selection
  - Repetition

Structured programming

- Structured programming
  - Increases clarity of code
  - Reduces cognitive load
  - Reduces likelihood of errors
  - Reduces debugging time
  - Facilitates methodical approaches to testing
  - Increases ease of maintenance

Alternative to breaks

```
int counter = 0;
System.out.println("Enter -1 to terminate, "
                        + " any other integer to loop: ");
int answer = keyboard.nextInt( );

while (counter < 20 && answer != -1)
{
      ++counter;
      System.out.println(answer);
      System.out.println("Enter -1 to terminate, "
                              + " any other integer to loop: ");
      answer = keyboard.nextInt( );
}
```

- The exit method of the System class

System.exit( )

- Sometimes in a program a condition occurs that the program cannot handle
- In these cases, we wish to terminate the program gracefully rather than let it hang, crash or give an incorrect result
- The System.exit( ) method allows us to terminate a program

System.exit( )

- For example, your code is about to divide a value by a variable
- If the variable's value is zero, the division will not be successful
- If your program is unable to proceed, at this point a call to System.exit( ) will terminate the program
- Note: do not use System.exit( ) for normal program termination; it is good style only to use it to terminate when an error occurs that cannot be handled

Example

```
if (b == 0)
{
    System.exit(0);
}
c = a / b;
```

System.exit( )

- System.exit( ) expects an argument
- The value of its argument is passed back to the operating system and could be used by it
- The convention for the returned value is
  - the value 0 represents normal termination
  - the value 1 (or any non-zero value) represents abnormal termination

- More examples for loop construction

Class exercise

- Write code to read a line of text from the user and output it backwards

- E.g. if the user input the text

  The Truth is Out There

- the program outputs

  erehT tuO si hturT ehT

- Use a for loop

Class exercise

Class exercise

- Write a for loop that reads a line of text from the user and outputs whether it is a palindrome (i.e. reads the same forwards and backwards)
- Example palindromes
  - kayak
  - live evil
  - racecar

Solution

Class exercise

- Rewrite the last code segment so that it detects palindromes irrespective of capitalisation and punctuation
- Example palindromes
  - Madam, I'm Adam
  - No lemons, no melon.
  - Was it a bar or a bat I saw?
  - Senile felines

Class exercise

- We can use the code from the palindrome checker to test for a palindrome, but we need to create a new version of the original string without punctuation, and with all letters in the same case to test
- The Character class has some class methods that we may use
- Character.isLetter(char) returns a boolean
- Character.toUpperCase(char) returns a char

Solution

Next lecture

- Constructing loop statements
- Nested loops