CSEN 202 – Introduction to Computer Programming

Lecture 4: Iterations

Prof. Dr. Slim Abdennadher,
Dr. Wael Abouelsaadat and
Dr Mohammed Abdel Megeed Salem
slim.abdennadher@guc.edu.eg

German University Cairo, Faculty of Media Engineering and Technology

February 25 - March 2, 2017

Previous topics

- What is Java, how to compile and run “Hello World!”
- Primitive datatypes and their properties (**`byte`**, **`short`**, **`int`**, **`long`**, **`float`**, **`double`**, **`boolean`**, **`char`**)
  - Literals and special values
  - Type compatibilities (explicit and implicit cast)
- Simple expressions (`+`, `-`, `*`, `/`, `%`, `&`, `|`, `~`, `>>`, . . .), their properties, *etc.*
  - Operand and result types, precedence, *etc.*
- Assignments (expression with essential side effect), memory changing expressions (`++`, `--`, `+=`, . . .)

Previous topics

- Blocks ({ ... }) for structuring the program
- The **break**-instruction for leaving a block
- Branching and decision constructs
  - **if** (condition) statement,
  - **switch** (condition) { statements },
  - condition ? expression : expression.

Today's topic

iterative constructs

What is life?

"Life is just one damn thing after another."

—Mark Twain

"Life isn't just one damn thing after another. . . it is the same damn thing over and over again."

—Edna St. Vincent Millay

Looping

Looping causes computer to execute section of code repeatedly

- We use boolean expressions (`true` and `false`) as loop condition; when boolean is `false`, loop condition equals exit condition and loop is terminated
- As with conditionals, this section of code can be single statement or multiple statements enclosed in curly braces (blocks)
- We call the section of code executed the loop's body

The loops

Java offers three different iterative constructs:

- The **`while`**-loop,
- The **`do-while`**-loop, and
- The **`for`**-loop

They differ in the relation between loop condition and loop body

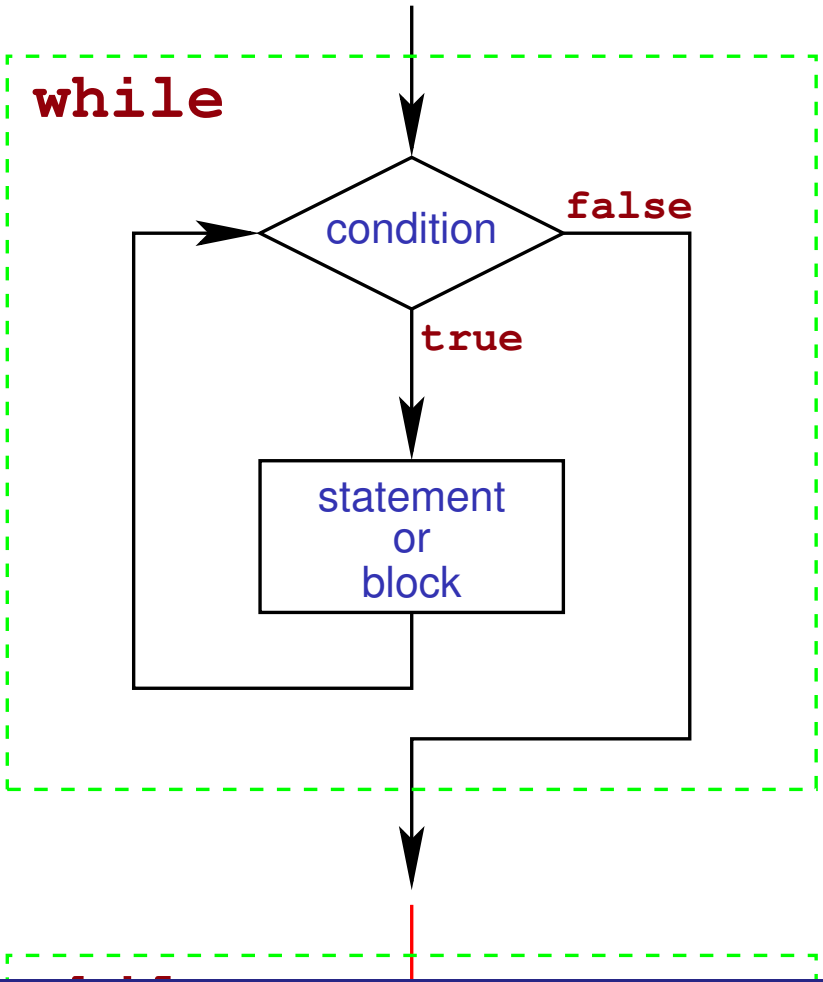
The `while` loop

Format

```
while (condition)
    statement or block
```

- A **`while`**-loop executes the loop body (a statement or a block) as long as the loop condition is true.
- The condition must be of type **`boolean`**
- Before every execution of the loop body, the loop condition is evaluated.
- As soon as the condition evaluates to **`false`**, the loop terminates.
- Note: The loop body may not be executed at all.

The `while` loop—schema



How to construct a `while` loop

1. Formulate the test which tells you whether the loop needs to be run again
   - `count <= 3`
2. Formulate the actions for the loop body which take you one step closer to termination

```
{
  System.out.println( "count is: " + count );
  count = count + 1; // add one to count
}
```

3. In general, initialization is required before the loop and some postprocessing after the loop
   - `int count = 1;`

How to construct a `while` loop

```
class WhileExample
{
  public static void main (String[] args )
  {
    int count = 1;          // start count out at one
    while ( count <= 3 ) // loop while count is <= 3
      {
        System.out.println( "count is: " + count );
        count = count + 1; // add one to count
      }
    System.out.println( "Done with the loop" );
  }
}
```

`while` loop example

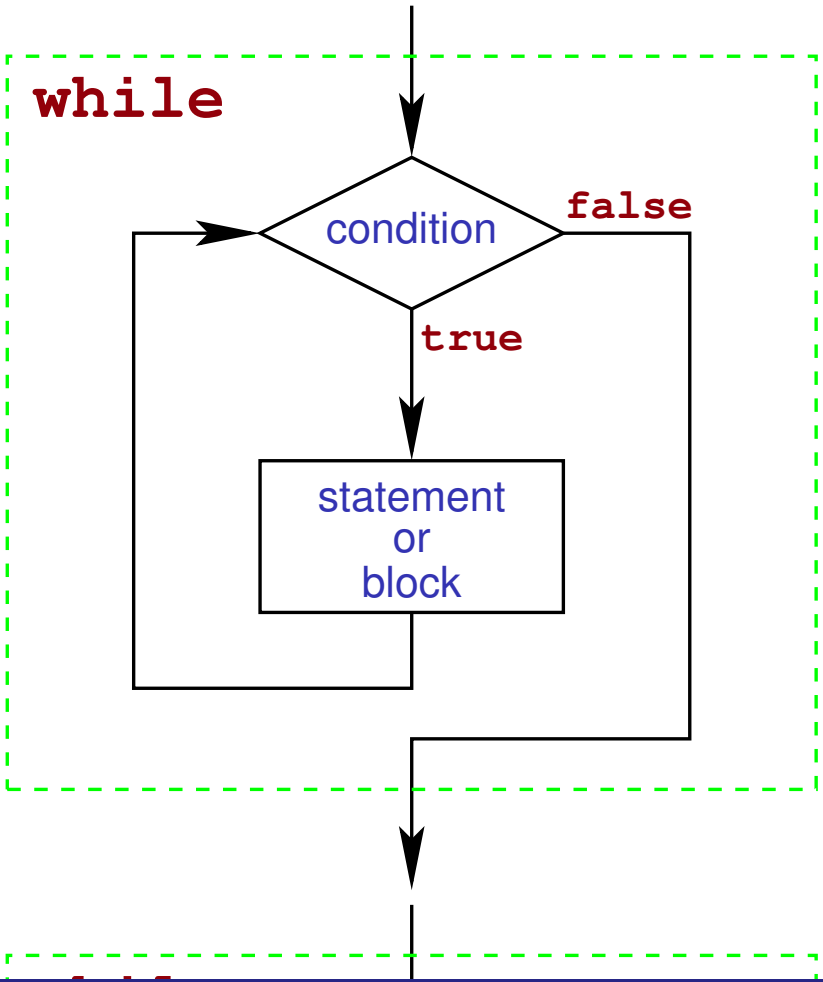
Example (Investment with compound interest)

Invest 10000€ with 5% interest compounded annually:

Year	Balance
0	10,000.00
1	10,500.00
2	11,025.00
3	11,576.25
4	12,155.06

Question. When will the balance be at least 20000 Euro?

`while` loop example



while loop example

```
class InvestmentTest {
  public static void main (String[] args ) {
    double balance = 10000;
    double rate = 5;
    double targetBalance = 20000;
    int year = 0;
    while (balance < targetBalance) {
      year++;
      double interest = balance * rate / 100;
      balance = balance + interest;
    }
    System.out.println("The investment doubled after"+
                       year +"years");
  }
}
```

The do loop

Format

```
do
      statement or block
while (condition);
```

- A **do-while**-loop executes the loop body (statement or block) once and then repeats as long as the condition is true.
- The condition must be of type **boolean**

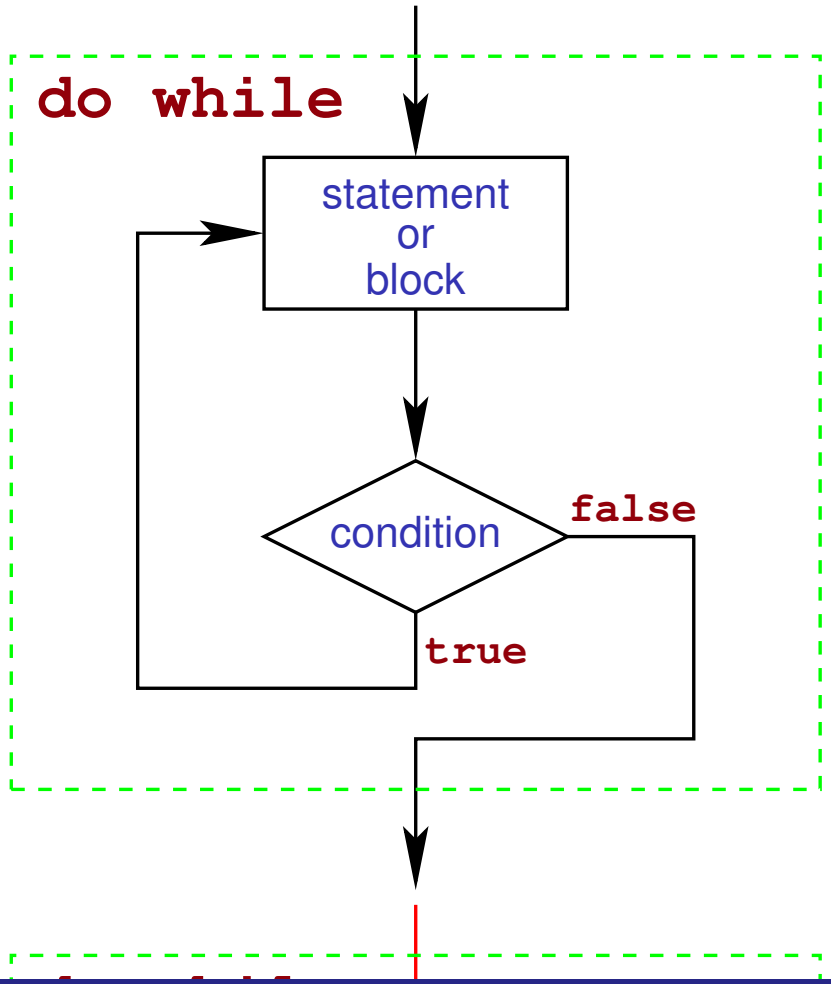
The do loop

Format

```
do
      statement or block
while (condition);
```

- After every execution of the loop body, the loop condition is evaluated.
- As soon as the condition evaluates to **false**, the loop terminates.
- Note: The loop body is executed at least once.

The do loop—schema

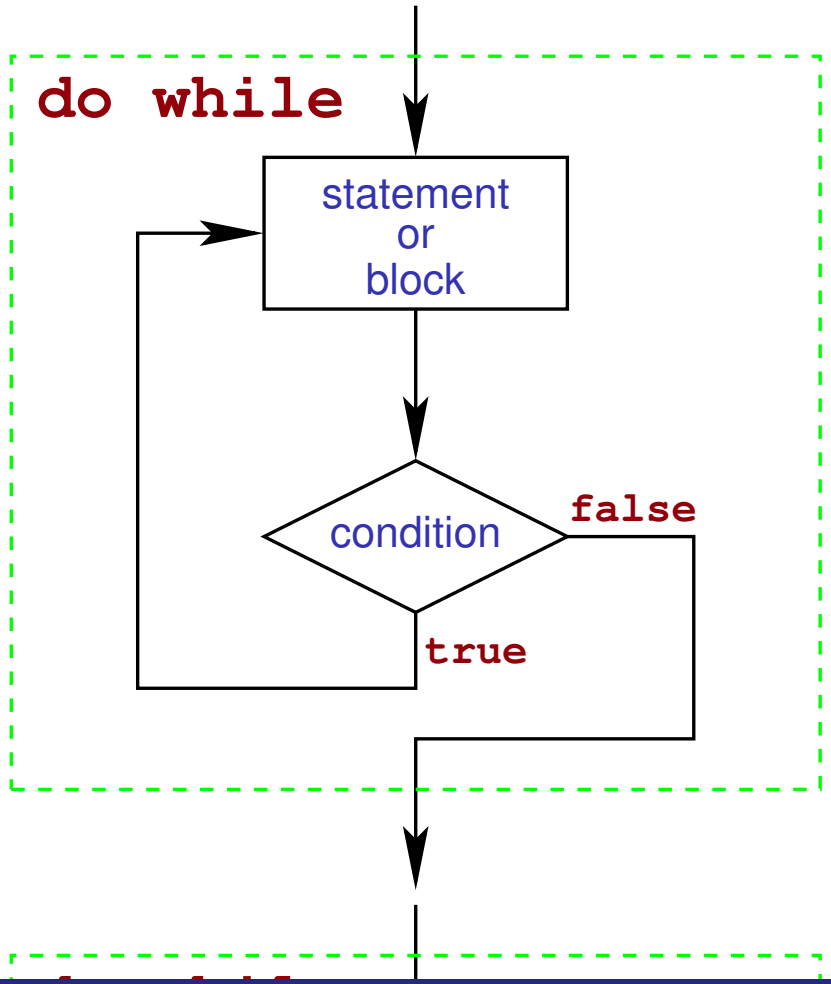


do loop example

Example (Validating an input)

Task: Accept only a positive integer

do loop example



do loop example

```
class ValidateInput {
  public static void main (String[] args ) throws IOException {
    BufferedReader userin = new BufferedReader
      (new InputStreamReader(System.in));
    String inputData;
    int value; // data entered by the user
    do {
      System.out.println( "Please␣enter␣a␣positive␣number:␣" );
      inputData = userin.readLine();
      value = Integer.parseInt( inputData );
    }
    while (value >= 0);
    System.out.println( "Entered␣negative␣number:␣" + value );
  }
}
```

Comparing `while` and `do` loops

- In both loops
  - Stops executing body if loop condition is `false`
  - you must make sure loop condition becomes `false` by some computations
  - Infinite loop means your loop condition is such that it will never turn `false` (*i. e.*, the exit condition never occurs)
- `do-while`
  - body always executed at least once
  - loop condition tested at bottom of loop
- `while`
  - may not execute at all
  - loop condition tested before body; loop condition variables must be set before loop entry

Comparing `while` and `do` loops

`while`-loops and **`do-while`-loops** can be transformed to each other

- **`do-while`** to **`while`**

```
do {
      statement 1;
      statement 2;
} while (condition);
```

→

```
statement 1;
statement 2;
while (condition) {
      statement 1;
      statement 2;
}
```

Comparing `while` and `do` loops

`while`-loops and **`do-while`-loops** can be transformed to each other

- **`do-while`** to **`while`**

```
while (condition) {
    statement 1;
    statement 2;
}
```

→

```
if (condition)
    do {
        statement 1;
        statement 2;
    } while (condition);
```

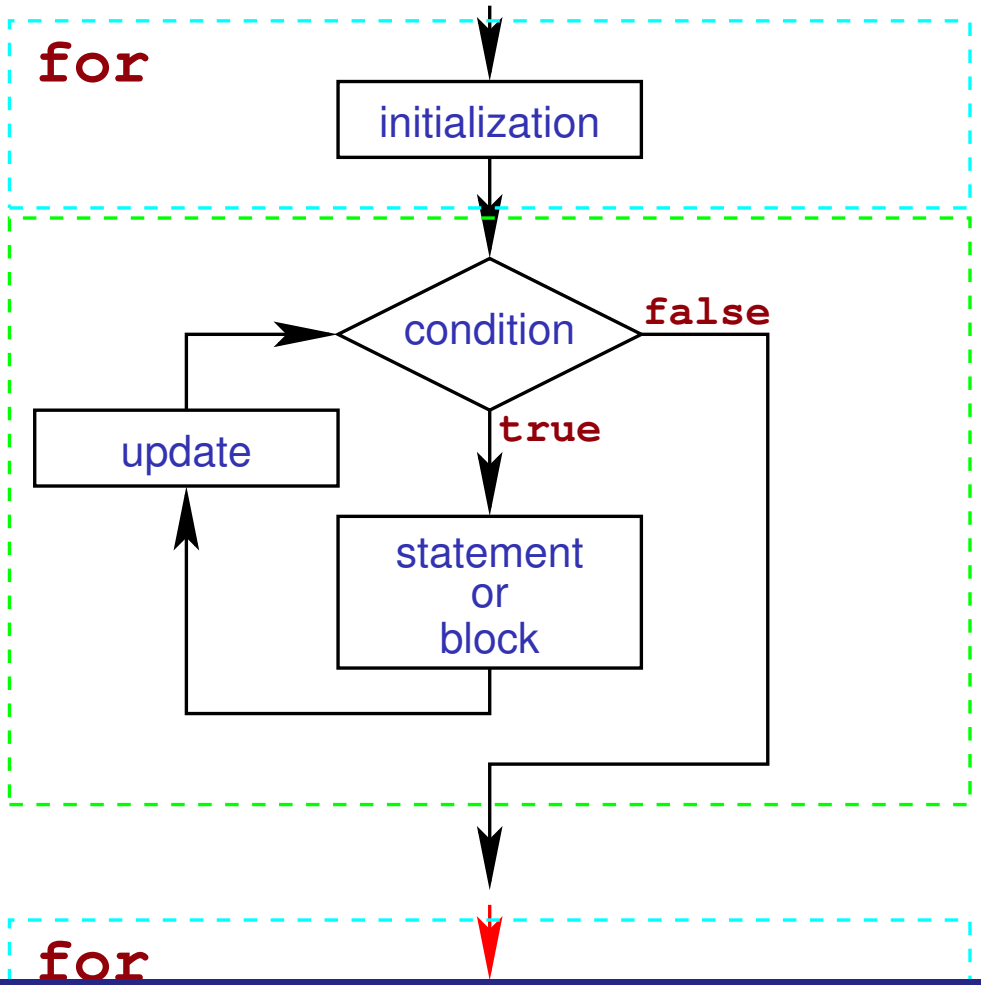
The `for` loop

Format

```
for (initialization; condition; update)
    statement or block
```

- Most common loop construct: just repeats a statement for a fixed number of times (counting loop)
- The initialization is an expression for setting initial value of the loop counter.
- The condition must be of type **`boolean`**
- The update expression modifies the loop counter
- Purpose: To execute an initialization, then keep executing and updating an expression while a condition is true.

The `for` loop—schema

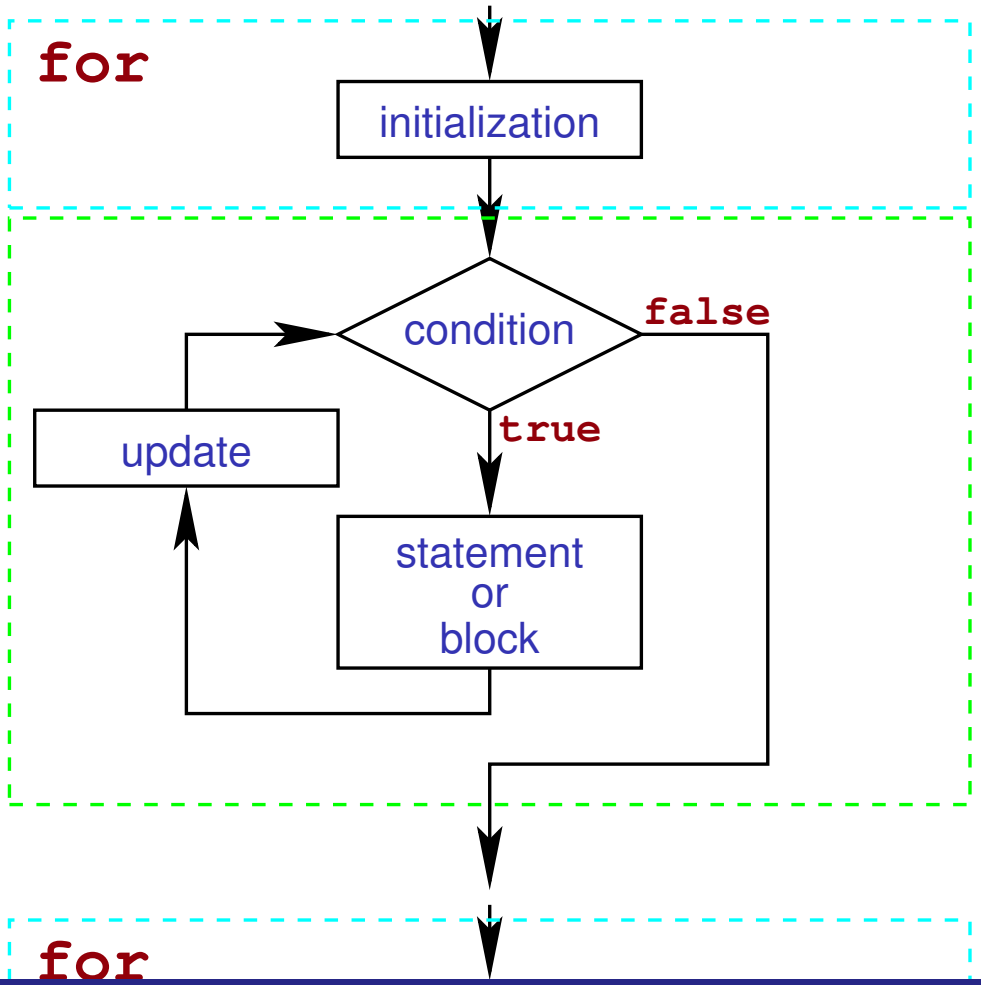


`for` loop example

Example (Investment with compound interest)

Invest 10000 Euro with 5% interest compounded annually.
Question: What will be the balance after `n` years?

`for` loop example



for loop example

```
class Balance {
    public static void main (String[] args ) {
        double balance = 10000;
        double rate = 5;
        int year = 15;
        for (int i = 0; i < year; i++) {
            double interest = balance * rate / 100;
            balance = balance + interest;
        }
        System.out.println("The investment after"+ year +
                           "will be" + balance);
    }
}
```

Another `for` loop example

Print a string backwards

- Recall what you have learned about strings
  - `String` is not a primitive type, it is a class.
  - The instances of a class are called objects
  - Objects provide their own functionality
- We can use the dot-operator "." and the methods
  - `length ()`, and
  - `charAt (position)`

Another `for` loop example

Print a string backwards

```
public class Reverse {
    public static void main (String[] args) {
        String word = "Slim";
        if (word == null) {
            return;
        }
        int max = word.length ();
        for (int i=max-1; i >=0; i--) {
            System.out.print (word.charAt (i));
        }
        System.out.println ("");
    }
}
```

Comparing `while` and `for` loops

- In general a `while`-loop has the form

```
initialization;
while (condition) {
        core loop body
        update/advancement
}
```

- This is exactly matched by the `for`-loop

```
for (initialization; condition; update/advancement) {
        core loop body
}
```

Choosing the right loop

- The `for`-loop is called definite loop because you can typically predict how many times it will loop. `while`- and `do`-loops are indefinite loops, as you do not know a priori when they will end.
- The `for`-loop is typically used for math-related loops like counting finite sums.
- `while`-loop is good for situations where the condition could turn `false` at any time.
- `do` is used in same kind of situation as while loop, but when the body of the loop should execute at least once.
- When more than one type of loop will solve problem, use cleanest, simplest one

Task

Write an algorithm that will print the multiplication table for the numbers from 1 to n.
For example let $n = 4$:

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

Solution

- Loop over all rows:

```
int i = 1;
while (i <= n) {

 ...

    i++;
}
```

- Build an individual row:

```
int j = 1;
while (j <= n) {
    System.out.print (i * j + "␣");
    j++;
}
System.out.println ("");
```

Solution

Putting the elements together:

```
public class MultTable {
    public static void main(String[] args) {
        int n = 5;
        int i = 1;
        while (i <= n) {
            int j = 1;
            while (j <= n) {
                System.out.print (i * j + "␣");
                j++;
            }
            System.out.println ("");
            i++;
        }
    }
}
```

Solution

An alternative solution:

```
public class MultTable2 {
    public static void main(String[] args) {
        int n = 5;
        for (i = 1; i <= n; i++) {
            for (j = 1; j <= n; j++)
                System.out.print (i * j + "␣");
            System.out.println ("");
        }
    }
}
```

Task

Print a triangle pattern using an increasing number of brackets. For example (5 rows):

```
[]
[][]
[][][]
[][][][]
[][][][][]
```

Solution

- Loop over all rows:

```
for (int i = 1; i <= n; i++) {
...
}
```

- Build an individual triangle-row:

```
for (int j = 1; j <= i; j++)
    r = r + "[]";
r = r + "\n";
```

Solution

Putting the elements together:

```
class Triangle {
    public static void main (String[] args ) {
        String r = "";
        int n = 10;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++)
                r = r + "[]";
            r = r + "\n";
        }
        System.out.print (r);
    }
}
```

Next week's events

- The next topic will be the concept of procedures and methods.