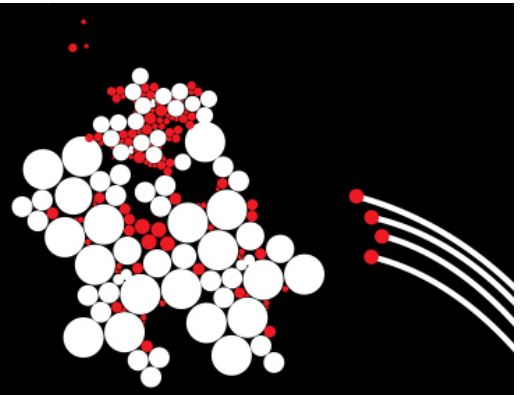
UNIVERSITY OF TWENTE.

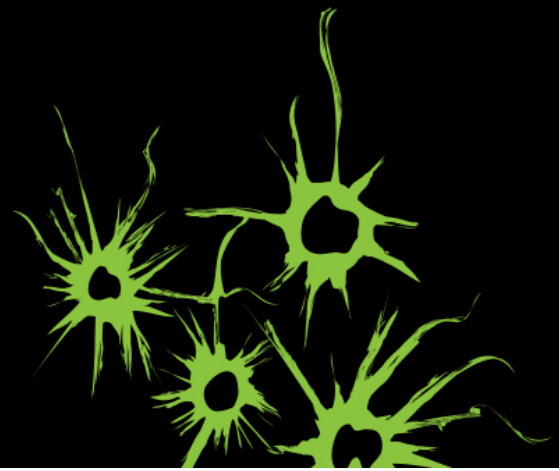


P1.2: CONTROL FLOW

LUÍS FERREIRA PIRES

201700117-1B MODULE 2: SOFTWARE SYSTEMS

12 NOVEMBER 2019



PROGRAMMING LINE OVERVIEW

Week 1 Values and variables Control flow	**Week 2** Classes and objects Testing	**Week 3** Interfaces and Inheritance Subtyping Security 1
Week 4 Arrays and Lists List implementations Collections	**Week 5** Stream I/O and MVC Exceptions Security 2	**Week 6** Concurrency Project kick-off IDE Tips & Tricks
Week 7 Basic Networking Networking and Multithreading GUIs	**Week 8/9** Advanced Java facilities Test	**Week 10** Project Test resit

GRACE MURRAY HOPPER

COMPUTER PIONEER (1906-1992)

- Tried to follow instructions on a bottle of shampoo (‘lather’, ‘rinse’, ‘repeat’) like a computer would, and ran out of shampoo

CONTROL STRUCTURES

- Java programs consist of subroutines (methods) that pass the execution flow to each other, starting from the main method (routine)
- We discuss how you can control what the program does inside each subroutine (method)
  → 'programming in the small' in Eck
- Related to concept of algorithm discussed in Module 1
- This is done by means of control structures

BLOCKS

- Code between { and }
- Defines a scope for variable definitions

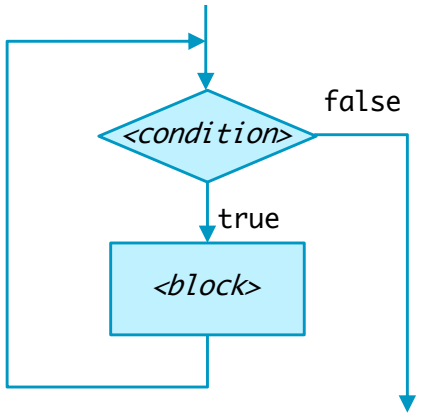
Example

```
//This block exchanges the values of x and y
{
    int temp;   // Temporary variable
    temp = x;   // Save a copy of the value of x in temp
    x = y;      // Copy value of y into x
    y = temp;   // Copy value of temp into x
}
```

- Java is a free-format language → indentation has no meaning!

WHILE LOOP

- If a boolean condition holds, executes a block and tests the condition again, otherwise leave
- Infinite loop: condition remains `true`



Example

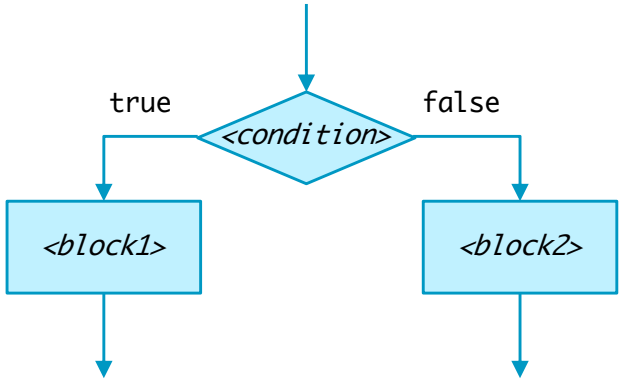
```
int number; // number to be printed
number = 1; // Start with 1

while (number < 6 ) {
    System.out.println(number);
    number = number + 1;
}
```

IF STATEMENT

CONDITIONAL BEHAVIOUR

- Checks a condition, if it is true executes *<block1>*, otherwise executes *<block2>*



Example

```
if (x > y) {
    y = 1;
}
else {
    y = 2;
}
```

- Special case: empty *<block2>*

```
if (x > y) {
    y = 1;
}
```

- `while` and `if ... else` are enough to represent any behaviour

BLOCK WITH ONE EXPRESSION

- If a block has only one expression in an `if` or `while`, the curly brackets can be omitted, but this is not recommended

```
if (x > y)
    y = 1;
else
    y = 2;
    System.out.println(y);
```

```
while (number < 6)
    System.out.println(number++);
    System.out.println("Finished!");
```

- Indentation has no meaning, so last statement is always executed!

WATCH OUT FOR THIS

IS THIS AN ERROR?

```
int y;

if (x < 0) {
    y = 1;
}
else {
    y = 2;
}
System.out.println(y);
```

Certainty that y is initialised

```
int y;

if (x < 0) {
    y = 1;
}
if (x >= 0) {
    y = 2;
}
System.out.println(y);
```

No certainty that y is initialised
Compiler cannot know that conditions are complementary!

RELATION WITH ALGORITMS (MODULE 1)

EXAMPLE FROM ECK (SECTION 3.2.1)

Abstract representation

```
Get the user's input
while there are more years to process:
    Compute the value after the next year
    Display the value
```

refinement

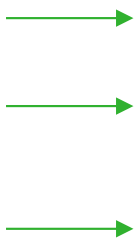
```
Ask the user for the initial investment
Read the user's response
Ask the user for the interest rate
Read the user's response
years = 0
while years < 5:
    years = years + 1
    Compute interest = value * interest rate
    Add the interest to the value
    Display the value
```

Translate to program code!

RELATION WITH ALGORITMS (CONT.)

TRANSLATION TO CODE

```
Ask the user for the initial investment
Read the user's response
Ask the user for the interest rate
Read the user's response
years = 0
while years < 5:
    years = years + 1
    Compute interest = value * interest rate
    Add the interest to the value
    Display the value
```



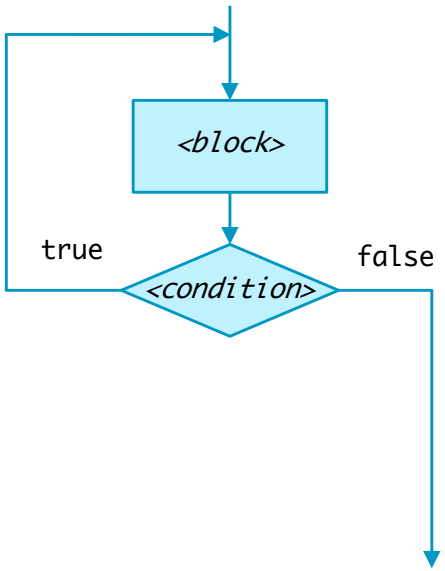
```
double principal, rate, interest;
int years;

System.out.println("Type initial investment: ");
principal = TextIO.getlnDouble();
System.out.println("Type interest rate: ");
rate = TextIO.getlnDouble();
years = 0;
while (years < 5) {
    years = years + 1;
    interest = principal * rate;
    principal = principal + interest;
    System.out.printf("%.2f \n", principal);
}
```

DO WHILE

SHORTCUT NOTATION

- In some situations, a block needs to be executed at least once
- Avoid repeating the block

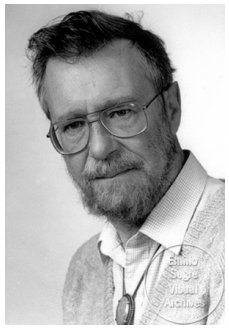


Example

```
int i = 5;
do {
    System.out.println(i);
    i++;
} while (i <= 10);
```

BREAK

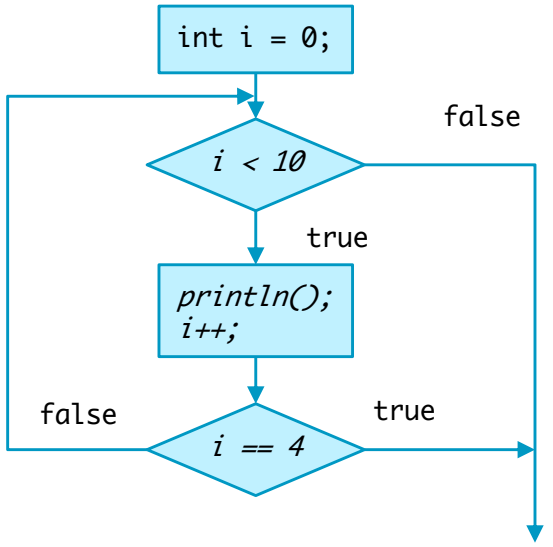
- With `while` you need to wait until the end of the block to leave the loop
- `break`: leave the execution of a loop in the middle of the block
- Should be used carefully (except in combination with `switch`) because they may break the regular block composition structure



See ‘sequencing discipline’ defined by Edsger Dijkstra (1930-2002) in his notes

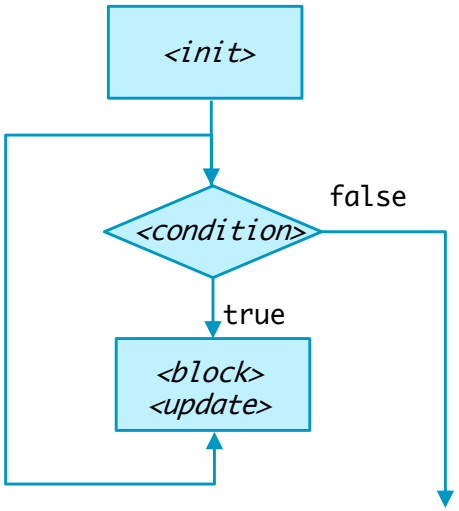
```
int i = 0;

while (i < 10) {
    System.out.println(i);
    i++;
    if (i == 4) {
        break;
    }
}
```



FOR LOOPS

VERY POPULAR LOOP PATTERN



- Compact notation
- Particularly useful to go through a list or array

```
int array[] = new int[10];
// ...
for (int i = 0; i < array.length; i++) {
    System.out.println(array[i]);
}
```

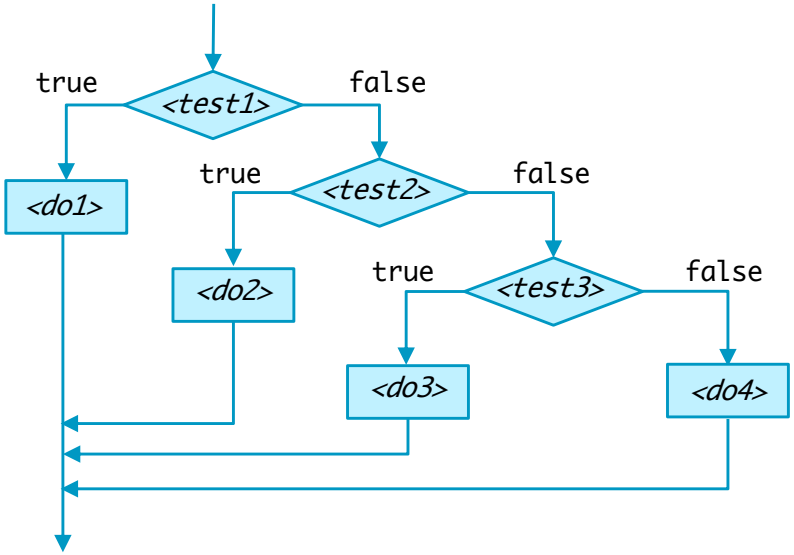
<initialisation> *<condition>* *<update>*

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

MULTIWAY BRANCHING

MULTIPLE IF-THE-ELSE STRUCTURES

- Blocks can be defined inside blocks in many ways
- We may have to test many different conditions on a certain value




```
if (temperature < 18.0) {
    System.out.println("It is cold");
} else if (temperature < 21) {
    System.out.println("It is nice");
} else {
    System.out.println("It is hot");
}
```

SWITCH

SHORTCUT FOR NESTED IF-THE-ELSES

- We may expect different behaviours for specific values
- Nested if-then-else structures may become too verbose

```
switch (number) {
case 0:
    System.out.println("Nothing");
    break;
case 1:
    System.out.println("Hey, good one");
    break;
case 2:
    System.out.println("Even and prime");
    break;
case 3:
case 5:
    System.out.println("Odd and prime");
    break;
default:
    System.out.println("Nothing special");
}
```

SWITCH AND MENUS

- Because it facilitates selection of options, the switch statement is particularly useful in menus

```
public static final int WINTER = 0;
public static final int SPRING = 1;
public static final int SUMMER = 2;
public static final int FALL = 3;
```

```
int currentSeason;

// Simple example of a menu followed by a switch statement
System.out.println("Give your preferred season: ");
System.out.println("0. Winter");
System.out.println("1. Spring");
System.out.println("2. Summer");
System.out.println("3. Fall");
System.out.println("> ");

currentSeason = TextIO.getlnInt();

switch (currentSeason) {
case WINTER:
    System.out.println("December, January, February");
    break;
case SPRING:
    System.out.println("March, April, May");
    break;
case SUMMER:
    System.out.println("June, July, August");
    break;
case FALL:
    System.out.println("September, October, November");
    break;
}
```

EXCEPTIONS

```
double x;
String str = TextIO.getlnWord();

x= Double.parseDouble(str);
System.out.println( "The number is " + x);
```

Things can go wrong!

- Input may have wrong format
- Network connections may break
- Bug may cause program to crash
- …

Exceptions are used to react to these situations!

```
text
Exception in thread "main" java.lang.NumberFormatException: For input string: "text"
        at java.base/jdk.internal.math.FloatingDecimal.readJavaFormatString(FloatingD
        at java.base/jdk.internal.math.FloatingDecimal.parseDouble(FloatingDecimal.ja
        at java.base/java.lang.Double.parseDouble(Double.java:543)
        at BlockExample.exceptionExample(BlockExample.java:224)
        at BlockExample.main(BlockExample.java:232)
```

```
double x;
String str = TextIO.getlnWord();

try {
    x = Double.parseDouble(str);
    System.out.println("The number is " + x);
} catch (NumberFormatException e) {
    System.out.println("Wrong format!");
    x = Double.NaN;
}
```

TAKE HOME MESSAGES

- Computers are stupid because they only do what they are told to do

- Computers are just a mirror of our own stupidity as programmer