Programmierpraktikum

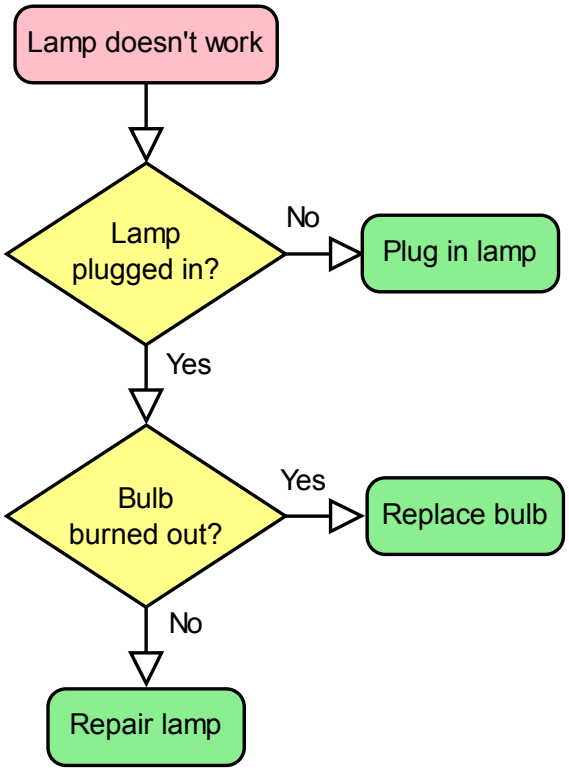
Claudius Gros, SS2012

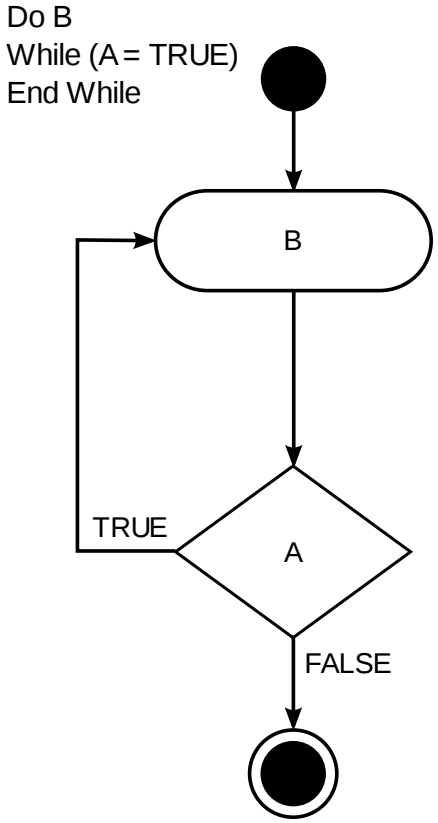
Institut für theoretische Physik
Goethe-University Frankfurt a.M.

Java - Control Flow

controlling the flow

- do different things depending on circumstances





logical operators

- comparison and logics

Symbol	Description	Examples
< <= > >=	less/-equal, greater/-equal	`if (i<=8) {}`
== !=	equal, unequal	`if (i==3) {}`
&& || !	logical AND, OR, NOT	`if ((a==0)&&(b!=3)) {}`
instanceof	object type comparison	`if (s instanceof String) {}`

conditional if

conditional – if ... then ... else

- the **`String[]`** array *args* contains the input parameters when launching the program with
  `java` *`IfThenElse par1 par2 ...`*

```
 1. public class IfThenElse {
 2.   public static void main(String[] args) {
 3.
 4.     if (args.length == 1) {
 5.       System.out.println("one: " + args[0]);
 6.     }
 7.
 8.     if (args.length == 2) {
 9.       System.out.println("two: " + args[0] + " " + args[1]);
10.     }
11.
12.     if (args.length == 1) {
13.       System.out.println("one");
```

```
14.     } else {
15.       System.out.println("not one");
16.     }
17.
18.     if (args.length > 2)
19.       System.out.println("more than two parameters");
20.   }
21. }
```

nested if

conditional – nested if

```
 1. public class NestedIf {
 2.
 3.   public static void main(String[] args) {
 4.     if (args.length == 0)
 5.       System.out.println("zero");
 6.     else if (args.length == 1)
 7.       System.out.println("one");
 8.     else if (args.length == 2)
 9.       System.out.println("two");
10.     else if (args.length == 3)
11.       System.out.println("three");
12.     else
13.       System.out.println("more");
14.   }
15. }
```

conditional switch

conditional – switch

```
 1. public class Switch {
 2.
 3.   public static void main(String[] args) {
 4.     char operator = args[0].charAt(0);
 5.        // get first letter of first parameter in command line
 6.
 7.     int operand1 = 12;
 8.     int operand2 = 4;
 9.
10.     switch (operator) {
11.     case '+':
12.       System.out.println(operand1 + operand2);
13.       break;
14.     case '-':
15.       System.out.println(operand1 - operand2);
16.       break;
17.     case '*':
```

```
18.       System.out.println(operand1 * operand2);
19.       break;
20.     case '/':
21.       System.out.println(operand1 / operand2);
22.       break;
23.     default:
24.       System.out.println("unknown operator");
25.       break;
26.     }
27.   }
28. }
```

control Flow: loop while

loop – while

```
 1. public class While {
 2.   public static void main(String[] args) {
 3.
 4.     int i = 0;
 5.     while(i <= 100){
 6.       System.out.println(i);
 7.       i += 10;
 8.     }
 9.   }
10. }
```

do ... while - loop

loop – do ... while

```
 1. import java.util.Scanner;
 2.
 3. public class DoWhile {
 4.   public static void main(String[] args) {
 5.
 6.     int i = 0;
 7.     do {
 8.       System.out.println(i);
 9.       i += 10;
10.     } while (i <= 100);
11.
12.     Scanner scanner = new Scanner(System.in);
13.     System.out.println(scanner.delimiter());
14.
15.     String input;
16.     do {
17.       input = scanner.next();
```

```
18.     } while (!input.equalsIgnoreCase("quit"));
19.   }
20. }
```

for - loop

loop – for

- loops are everywhere

```
1. public class For {
2.   public static void main(String[] args) {
3.
4.     for (int i = 0; i < 10; i++) {
5.       System.out.println(i);
6.     }
7.   }
8. }
```

break & continue

loop – break & continue

- break out of the loop: **`break;`**
- re-entry the loop: **`continue;`**

```
 1. public class BreakContinue {
 2.   public static void main(String[] args) {
 3.
 4.     while(true){   // infinite loop
 5.                    // label α
 6.       int i = (int) Math.round(Math.random() * 100); // random number ∈ [0, 99]
 7.       if(i > 50)
 8.         continue;  // re-entry the loop (go to label α)
 9.       System.out.print(i + ", ");
10.       if(i == 0)
11.         break;     // break out of the loop (go to label β)
12.     }
13.                   // label β
```

```
14.     System.out.println("done");
15.   }
16. }
```

throwing and catching exceptions

exception – try … catch … finally

- *exceptions* are non regular program events
  exceptions are **thrown** whenever something goes wrong, like
  - *division by zero (runtime)*
  - *trying to access a non-existing array elemnt (out of bound)*
  - *trying to open a non existent file (IO)*
  - *…*

  a program will never crash if all exceptions are handled correctly
- for some operations, like opening files for reading or writing,
  exception handling is **mandatory**
  exception handling is **optional** for `RuntimeException` and subclasses
- exceptions are handled by wrapping the dangerous operation with a
  `try { .... } catch () {}` bracket

example: exception handling

- **`Integer.parseInt(`***`String`***`))`** member function of the **`class Integer`** converting a string into integers, if possible

```
 1. public class TryCatchFinally {
 2. /** This function throws an artimetic runtime exception
 3.   * whenever a division by zero is perfomed.
 4.   */
 5.   public static int devide (int dividend, int divisor)
 6.          throws ArithmeticException{
 7.     if (divisor == 0)
 8.       {
 9.       throw new ArithmeticException("division by zero");
10.       } else
11.     return dividend / divisor;
12.   }  // end of TryCatchFinally.divide()
13.
14. /** Entry point.
15.   */
16.   public static void main(String[] args) {
```

```
17.     try{
18.        int result = TryCatchFinally.devide(Integer.parseInt(args[0]),
19.                                            Integer.parseInt(args[1]));
20.        System.out.println(result);
21.        } catch(ArithmeticException exception)
22.          {           // thrown by TryCatchFinally.devide()
23.          System.out.println("Division by zero!");
24.          } catch(ArrayIndexOutOfBoundsException exception)
25.            {         // thrown by calling args[]
26.            System.out.println("Provide two parameters!");
27.            System.out.println(exception.toString());
28.            } catch(NumberFormatException exception)
29.              {       // thrown by Integer.parseInt()
30.              System.out.println("Provide only integer numbers!");
31.              } finally
32.                {     // always executed
33.                System.out.println("Done with or without error.");
34.                }
35.   }  // end of TryCatchFinally.main()
36. }    // end of class TryCatchFinally()
```

exercise: number game

The computer chooses a random integer number of a given interval.
The user tries to guess the number while the computer gives the feedbacks "too high" or "too low".

- get random number:

```
1. int random = (int) Math.round(Math.random() * 100);
```

- get input:

```
1. Scanner scanner = new Scanner(System.in).useDelimiter("\n");
2. String input = scanner.next();
```

- parse integer:

```
1. int number = Integer.parseInt(input);
```

solution: number game

```
 1. import java.util.Scanner;
 2. public class NumberGame {
 3.
 4.   public static void main(String[] args) {
 5.     Scanner scanner = new Scanner(System.in).useDelimiter("\n");
 6.     int min = 0, max = 100, input = 0, numberOfGuesses = 0;
 7.     int random = (int) Math.round(Math.random() * (max - min)) + min;
 8.
 9.     System.out.println("Guess the random integer number between " + min + " and " + max + "!")
10.     do {
11.       numberOfGuesses++;
12.       while (true) {
13.         System.out.print("guess " + numberOfGuesses + ": ");
14.         try {
15.           input = Integer.parseInt(scanner.next());
16.           break;
17.         } catch (NumberFormatException exception) {
18.           System.out.println("Integer number requested!");
19.         }
```

```
20.       }
21.       if (input < random) {
22.         System.out.println(input + " is too low.");
23.       } else if (input > random) {
24.         System.out.println(input + " is too high.");
25.       }
26.     } while (input != random);
27.     System.out.println(input + " is correct!");
28.   }
29. }
```

exercise: factorial (iterative)

The computer has to calculate the factorial of a non-negative integer number given by the user using iteration.

- $0! \coloneqq 1$
- $\forall\, n \in \mathbb{N} \setminus \{0\}: \quad n! \coloneqq \prod_{k \in [1,\, n]} k \quad = \quad 1 \cdot 2 \cdot \ldots \cdot n$
- create a large number:

```
1. BigInteger number = BigInteger.valueOf(15);
```

- multiply numbers:

```
1. BigInteger product = number1.multiply(number2);
```

factorials via iteration

- **BigInteger**: integers with arbitrary length

```
 1. import java.math.BigInteger;         // arbitrary length integers
 2.
 3. public class IterativeFactorial {
 4. /** Factorial using BigIntegers.
 5.   */
 6.   public static BigInteger factorial(int arg) {
 7.     BigInteger result = BigInteger.valueOf(1);
 8.     for (int i = 2; i <= arg; i++) {
 9.       result = result.multiply(BigInteger.valueOf(i)); // ℤ
10.     }
11.     return result; // only correct if arg ≥ 0
12.   }
13.
14. /** Factorial using Long integers.
15.   */
16.   public static long factorial(byte arg) {
17.     long result = 1;
```

```
18.     for (byte i = 2; i <= arg; i++) {
19.       result *= i; // [-9,223,372,036,854,775,808, 9,223,372,036,854,775,807]
20.     }
21.     return result; // only correct if arg ∈ [0, 20]
22.   }
23.
24.   public static void main(String[] args) {
25.     try {
26.
27.       int intNumber = Integer.parseInt(args[0]); // [-2,147,483,648, 2,147,483,647]
28.       BigInteger bigIntegerFactorial = IterativeFactorial.factorial(intNumber);
29.
30.       System.out.println(intNumber + "! = " + bigIntegerFactorial);
31.
32.       byte byteNumber = Byte.parseByte(args[0]); // [-128, 127]
33.       long longFactorial = IterativeFactorial.factorial(byteNumber);
34.
35.       System.out.println(byteNumber + "! = " + longFactorial);
36.     } catch (Exception exception) {
37.       System.out.println("An error occured!");
38.       System.out.println("usage: java IterativeFactorial ");
39.     }
40.   }
```

```
41. }
```

factorials via recursion

```
 1. public class Recursion {
 2.
 3.   public static void down(int i) {
 4.     System.out.println(i);
 5.     if (i > 0)
 6.       Recursion.down(i - 1);
 7.   }
 8.
 9.   public static int sum(int i){
10.     if(i > 0){
11.       return i + sum(i - 1);
12.     }else{
13.       return i;
14.     }
15.   }
16.
17.   public static void main(String[] args) {
18.     Recursion.down(10);
19.     System.out.println("100*101/2 = " + Recursion.sum(100));
```

```
20.   }
21. }
```

recursion: Fibonacci series

training suggestion

write a code for evaluating the n-th Fibonacci number

- $f_0 \coloneqq 0, \quad f_1 \coloneqq 1$
- $\forall\, n \geq 2: \quad f_n \coloneqq f_{n-1} + f_{n+2}$

possible solution

```
 1. public class Fibonacci {
 2.
 3.   public static long fibonacci(int arg) {
 4.     switch (arg) {
 5.     case 0:
 6.       return 0; // f_0 := 0
 7.     case 1:
 8.       return 1; // f_1 := 1
 9.     default:
10.       return Fibonacci.fibonacci(arg - 1) + Fibonacci.fibonacci(arg - 2);
```

```
11.                 // f_n := f_(n-1) + f_(n-2)
12.     }
13.   }
14.
15.   public static void main(String[] args) {
16.     try {
17.       int number = Integer.parseInt(args[0]);
18.       long fibonacci = Fibonacci.fibonacci(number);
19.       System.out.println("f_" + number + " = " + fibonacci);
20.     } catch (Exception exception) {
21.       System.out.println("An error occured!");
22.       System.out.println("usage: java Fibonacci ");
23.     }
24.   }
25. }
```

loops and sums

- **training suggestion**
  calculate results first and store them in vectors,
  then one loop for the output
- what happens?

```
 1. public class TwoSum {
 2.
 3. public static void main(String args[]) {
 4.    double sum, smallNumber;
 5.    double bigNumber = 1.0;
 6.    int n = 100;
 7. // ***
 8. // *** with not too small number first
 9. // ***
10.    for (int exponent=4;exponent<19;exponent=exponent+2)
11.      {
12.      System.out.printf("exponent: %d\n",exponent);
13.      smallNumber = Math.pow(10.0,-exponent);
```

```
14.       sum = n*smallNumber + bigNumber;  // sum of smallNumber first
15.       System.out.printf("adding sum of small numbers to big number: %20.18f\n",
16.                         sum);
17. //
18.       sum = bigNumber;
19.       for (int i=0;i<n;i++)
20.         sum += smallNumber;
21.       System.out.printf("       adding small numbers to big number: %20.18f\n",
22.                         sum);
23.       System.out.println(" ");
24. //
25.       } // end of loop over exponents
26.    }    // end of TwoSum.main()
27. }       // end of TwoSum
```

numerics: summation

large number of small numbers

- $\sum_{n=1}^{N} \frac{1}{n^2}$
- small terms first!

```
 1. import java.lang.Math;   // standard import, always useful
 2. /** Series summation may depend on the order.
 3.   * Summing 1/k^2 upwards and downwards.
 4.   */
 5. public class SeriesSummation {
 6.
 7. public static void main(String[] args) {
 8.   double exact = Math.PI*Math.PI/6.0;  // infinite series
 9.   double resultUp   = 0.0;
10.   double resultDown = 0.0;
11.   int maxN = 0;
12.   for (int ii=4; ii<21; ii+=4)
13.     {
```

```
14.     maxN = ii*10000000;
15.     resultUp   = SeriesSummation.summationUp(maxN);  // both calls
16.     resultDown = summationDown(maxN);                // possible
17.     System.out.printf("%20.16f %20.16f %20.16f\n",
18.                       exact,resultUp,resultDown);
19.     }
20.   System.out.println(" ");
21.   System.out.printf("exact result : %20.16f\n",exact);
22.   System.out.printf("summing down : %20.16f\n",resultDown);
23.   System.out.printf("  summing up : %20.16f\n",resultUp);
24.   System.out.printf("       error :            ^^\n");
25.   System.out.printf("        maxN : %d\n",maxN);
26. }
27.
28. /** Summing upwards.
29.   */
30. public static double summationUp(int maxN) {
31.   double result = 0.0;
32.   for (int i=1; i<=maxN; i++)
33.     result += 1.0/(1.0*i*i);
34.   return result;
35.   }    //  end of summationUP
36.
```

```
37. /** Summing downwards.
38.   */
39. public static double summationDown(int maxN) {
40.   double result = 0.0;
41.   for (int i=maxN; i>0; i--)
42.     result += 1.0/(1.0*i*i);
43.   return result;
44.   }    //  end of summationUP
45.
46. }    //  end of class seriesSummation
```

- $N = 20 \cdot 10^7 = 2 \cdot 10^8$
- $1/N^2 = 0.25 \cdot 10^{-16}$
- error: $\approx 10^{-8}$ why?

```
  1.6449340668482264   1.6449340418154346   1.6449340418482268
  1.6449340668482264   1.6449340545247193   1.6449340543482265
  1.6449340668482264   1.6449340578345750   1.6449340585148930
  1.6449340668482264   1.6449340578345750   1.6449340605982263
  1.6449340668482264   1.6449340578345750   1.6449340618482264

exact result :   1.6449340668482264
summing down :   1.6449340618482264
  summing up :   1.6449340578345750
```

```
error :            ^^
 maxN : 200000000
```