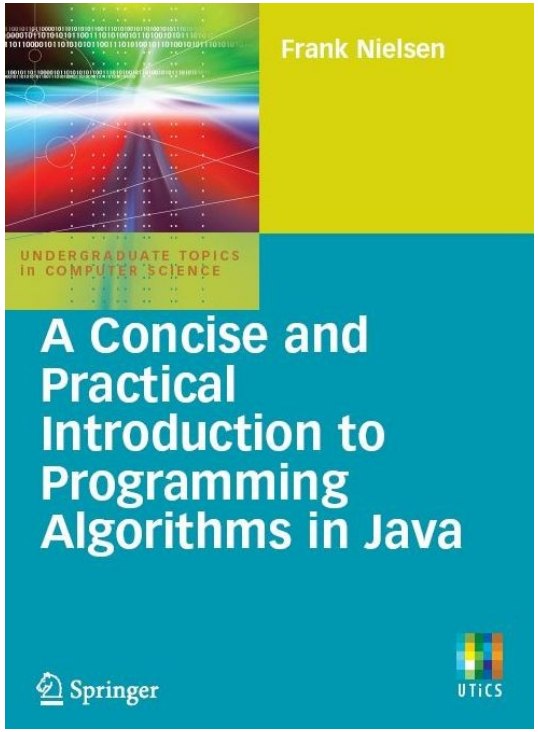
A Concise and Practical Introduction to Programming Algorithms in Java



Chapter 2: Conditional structures and loops

Frank NIELSEN

nielsen@lix.polytechnique.fr

Upper case versus Lower case

Java **distinguishes** between uppercases (A..Z) and lowercases (a..z)

- Unix *differentiates* upper/lower case filenames

```
class UpperLowerCase
{
   public static void main (String arguments[])
   {
   int MyVar;
      // this variable is different from MyVar
   int myvar;

   // Generate a syntax error at compile time:
   // cannot find symbol variable myVar
   System.out.println(myVar);
   }
}
```

Reserved keywords

You *cannot* choose reserved keywords for variable names:

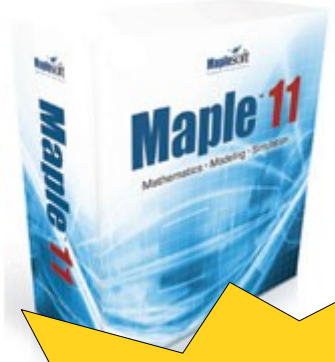
```
class ReservedKeyword
{public static void main (String arg[]){
   double x,y;
   // Generate a syntax error:
   // "not a statement"
   int import;
   }
}
```

Reserved keywords in Java:

abstract	default	if	private	throw
boolean	do	implements	protected	throws
break	double	import	public	transient
byte	else	instanceof	return	try
case	extends	int	short	void
catch	final	interface	static	volatile
char	finally	long	super	while
class	float	native	switch	
const	for	new	synchronized	
continue	goto	package	this	

Displaying versus Computing

- You need to display if you'd like to see the result of evaluating an expression
- `System.out.println` displays on the console *with* a return carriage
- `System.out.print` displays on the console *without* a return carriage



$$> \; 2 + 2;$$

$$4$$

$$> \; ifactor(203490);$$

$$(2)\,(3)^2\,(5)\,(7)\,(17)\,(19)$$

Confusing in Maple:
Interpreter: compute:display

Java is not Maple nor SciLab!

Output: Displaying values & messages

- **`System.out.println(stringname)`**: displays a string with a return carriage
- **`System.out.print(stringname)`**: displays a string *without* return line
- **`System.out.println(value)`**: converts (cast) numerical value into a string and displays it
- **`System.out.println(''The value of x is ''+x)`**:
  Converts the numerical value of x into a string and concatenate that string with the constant string " The value of x is "

println.java

```java
class println
{

    public static void main (String[] args)
    {
        double x=Math.E;
        int i=23;
        int a=25;
        int b=1024;

        System.out.println(x);
        System.out.println(i);

        System.out.print(a); System.out.print(b);

        System.out.print("\n");

        System.out.print("The value of x is "+x);


    }

}
```

C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe

```
2.718281828459045
23
251024
The value of x is 2.718281828459045Press any key to continue...
```

More on `System.out.print[ln]`

Equivalences of *stream* output:

```
System.out.print(''\n'') = System.out.println('''');
System.out.print(''Hello INF311 \n'') = System.out.println(''INF311'');
```

stream.java

```
1
2  class stream
3  {
4
5      public static void main (String[] args)
6      {
7
8      System.out.print("Hello INF311! \n");
9
10     System.out.println("Hello INF311!");
11
12     // Now a question... showing the difference betweem math and languages
13
14     int i=23;
15
16     System.out.println("The value of i+1 is "+i+1);
17
18     // Beware of + operator for strings and numbers !!!
19     }
20
21 }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE
Hello INF311!
Hello INF311!
The value of i+1 is 231
Press any key to continue...
```

Priority order+casting operations...

Display: String concatenations...

Cumbersome to type several `System.out.println` and `System.out.print`

Shortcut: **String concatenations** « + »...

Operator (expression)

```
int a=1, b=-2;
     System.out.print("a="); System.out.print(a);
     System.out.print(" b="); System.out.println(b);

     System.out.println("a="+a+" b="+b);

     String s1="Lecture in", s2=" Java";
     String s=s1+s2;// string concatenation
     System.out.println(s);
```

```
a=1 b=-2
a=1 b=-2
Lecture in Java
```

Declaring constants

```
/* Declare a constant (not a variable)
 to bypass using Math.PI */

  final double PI = 3.14; // constant
```

Numeric bug in predicate !

```
// Constant
   final double PI = 3.14;

   int a=1;
   double b=a+PI;

   if (b==4.14) // Equality test are dangerous!!!
    System.out.println("Correct result");
    else
    {System.out.println("Incorrect result");
    System.out.println("a="+a+" b="+b+" PI="+PI);
    }
   }
```

```
Incorrect result
a=1 b=4.140000000000001 PI=3.14
```

Syntax and compilation

Syntax errors are easy program bugs (mistyping?)
...But *syntaxically correct* program may be difficult to understand

```
int i=3;
// syntax below is valid!
int var=i+++i;
```

What is the value of var?

Protecting Java Source with Code obsfucation
Avoid reverse engineering of applications

Program: Data,computations + workflow

The ***control structures*** define the set of instructions being executed (aiguillage des trains)

For example, a *branching condition*:

In Java, we do not use the word `then`

```
if (a<b)  [then]
      c=a; // Do-something-1
    else
      c=b; // Do-something-2
```

There are two potential instructions paths depending on the predicate:

- a<b, -> c=a;

or

- a>=b, -> c=b;

c *is selected* as the **minimum** of a and b

Controling program workflow

Two kinds:

- Branching tests: (if else, switch)
- Repeat structure: Loop (while, for)

```java
class branchingprog
{

    public static void main (String[] args){

        int a,b,c;


        a=3;
        b=15;


        if (a<b)   // Predicate
            c=a;   // Block true
            else
            c=b; // Block false


        System.out.println("c has value:"+c);



        }



    }
```

Predicate: true or false
if there is a numerical error at that stage we take the wrong flow and this yields a bug, with potential desastrous effects.

```
c has value:3
Press any key to continue...
```

Key difference with maths.

Annotating programs: comments!

Writing comments is good for (1) yourself and for (2) others to proofread and debug your code

D. Knuth

In fact, there are some *paradigms* to write in a *single file* both

- the clean documentations, and
- the code.

Exempla gratia (e.g.) cweb, Literate programming, etc.

http://www.literateprogramming.com/

In INF311, we rather write short programs, so we consider the *standard comments*:

```
// This is a single line comment

/* I can also write comments on
   several lines
     by using these delimiters */
```

Comments: single versus multiple lines (Jcreator IDE)

```java
class comments{


    // This is a comment of my program stored in filename comments.java


    // This is the *** magic formula that we will explain later on ***

    public static void main (String[] args)
    {
    double a,b;

    double x,y;


    /* The equation of a non vertical line is y=ax+b
      If, I need vertical lines too, I rather choose
      to write the equation as ax+by+c=0 as the equation
      with homogeneous coordinates (a,b,c)
     */

     x=3;
     a=1;
     b=-2;

     y=a*x+b;

     /* My editor in Java
      * just add the * in from of any newline automatically so that
      * comments look prettier
      */

     System.out.println("ax+b=y="+y+" for x="+x+" with a="+a+" b="+b);

        }



}
```

```
ax+b=y=1.0 for x=3.0 with a=1.0 b=-2.0
Press any key to continue...
```

Comments... with errors!

```
1  class commentserror
2  {
3
4      public static void main (String[] args)
5      {
6
7          // I like to write /* many programs
8          sometimes it is disturbing for the compiler */
9
10
11         /*
12           This comment further shows that we cannot imbricate
13          other complex comments in a several-line comment structure
14           /* it does not work,
15           it yields a syntaxical error
16          */
17          */
18     }
19 }
```

The compiler is verbose: Try to fix the first error first (greedy approach)

```
--------------------Configuration: <Default>--------------------
D:\Enseignements\INF311\Lectures2008\prog-inf311.2\commmentserror.java:8: ';' expected
                sometimes it is disturbing for the compiler */
                            ^

D:\Enseignements\INF311\Lectures2008\prog-inf311.2\commmentserror.java:8: ';' expected
                sometimes it is disturbing for the compiler */
                                         ^

D:\Enseignements\INF311\Lectures2008\prog-inf311.2\commmentserror.java:8: '(' expected
                sometimes it is disturbing for the compiler */
                                           ^

D:\Enseignements\INF311\Lectures2008\prog-inf311.2\commmentserror.java:8: ';' expected
                sometimes it is disturbing for the compiler */
                                                        ^

D:\Enseignements\INF311\Lectures2008\prog-inf311.2\commmentserror.java:8: illegal start of expression
                sometimes it is disturbing for the compiler */
                                                         ^

D:\Enseignements\INF311\Lectures2008\prog-inf311.2\commmentserror.java:17: illegal start of expression
                 */
                 ^

D:\Enseignements\INF311\Lectures2008\prog-inf311.2\commmentserror.java:17: illegal start of expression
                 */
                  ^

D:\Enseignements\INF311\Lectures2008\prog-inf311.2\commmentserror.java:18: illegal start of expression
        }
        ^

D:\Enseignements\INF311\Lectures2008\prog-inf311.2\commmentserror.java:19: illegal start of expression
}
Process completed.
```

Comments...
repaired... = *Syntaxically correct program*

commmentnoerror.java

```
1  class commentnoerror
2  {
3
4      public static void main (String[] args)
5      {
6
7          // I like to write
8          /* many programs
9          sometimes it is disturbing for the compiler */
10
11
12         /*
13          This comment further shows that we cannot imbricate
14         other complex comments in a several-line comment structure
15         */
16         // it does not work,
17         // it yields a syntaxical error
18
19     }
20 }
```

Do not forget:
Writing good comments is as important as writing source code

You will thank yourself for doing this once you look back at your programs **months later**

Structures of Java programs

- Comments // or /* */
- Constants (Math.PI, etc.)
- Variables (typed) with valid identifiers (not reserved keyword)
- Operators +,-,/,%, etc. for expressions
- Reserved language keywords: if, for, while, etc.

A **set of instructions** is called a **block**
Blocks can be delimited by *parenthesis* **{Block}**

```
{ // This is a block
// (non control structure inside it)
var1=Expression1;
var2=Expression2;
var3=Expression3;
...
}
```

Structures of java programs: Blocks

blocks.java

```java
class blocks{


    public static void main (String[] arguments)
    {
        // This is the main block for the procedure called "main"
        // observe the parenthesis that delimit the block

        double a,b,c;

        a=2;
        b=3;
        c=5;


        if (a+b==c)
                {
                    // This is a block delimited by parenthesis
                    System.out.println("a+b=c");

                }
                else

                {

                //  This is another block
                System.out.println("a+b is not equal to c");
                }


        }


}
```

Block 1

Block 2

Principal block

```
a+b=c
Press any key to continue...
```

Conditional structures:

if (predicate) Block1 else Block2

Essential control structure for executing a (block of) operations if a condition is `true` (or false for the else-Block)

```
if (condition) Instruction

if (condition) {
    Instruction1;
    Instruction2;
    Instruction3;
    ... ;
}

if (condition)
    BlockForTrueCase
    else
    BlockForFalseCase
```

predicate

Block=sequence of instructions

Conditional structures: Compact form
if (predicate) Inst1 else Inst2

Conditional instruction for singe instruction block can be called using the ternary operator (3 operands) « ? : »

BoolOperand1 ? TypeOperandTrue2 : TypeOperandFalse3

```
double x1=Math.PI;
double x2=Math.E;

double min=(x1>x2)? x2 : x1; // min value
double diff= (x1>x2)? x1-x2 : x2-x1; // absolute val.
System.out.println(min+" difference with max="+diff);
```

2.718281828459045 difference with max=0.423310825130748

Instructions and conditions

Instructions always terminate with a semi colon ;
(except potentially the last one in a block)

A set of instructions encapsulated in { } is a block
The block has the same syntax as an instruction

Variables *can be declared* in a block

A condition is a boolean expression
that returns either true or false :
= **A predicate**

Variables and blocks

scope.java

```java
class scope
{
public static void main (String[] args)
{
    int i=3;
    int j=4;

    System.out.println(i);

    {
    int ii=4;

    System.out.println(ii);

    j=6;
    }

        System.out.println(j);


}

}
```

C:\PROGRA~1\XINOXS~1\JCREAT~1\GE

```
3
4
6
Press any key to continue...
```

Very different from C++!
(Java better controls the syntax of programs, better semantic)

We cannot declare twice a same variable in encapsulated block

Variables and blocks: Scopes

scope.java

```java
class scope
{
public static void main (String[] args)
{
    int i=3;
    int j=4;

    System.out.println(i);

    {
    int ii=4;

    System.out.println(ii);

    j=6;

    int l=3;
    }

        System.out.println(j);

            System.out.println(l);


}

}
```

Error!!! Variable l is not defined in the block it here

```
--------------------Configuration: <Default>--------------------
D:\Enseignements\INF311\Lectures2008\prog-inf311.2\scope.java:22: cannot find symbol
symbol  : variable l
location: class scope
                        System.out.println(l);
                                           ^
1 error

Process completed.
```

Boolean operators for comparisons

a==b Test of equality (for basic types)
a!=b Test for difference [equivalent to **! (a==b)**]

Inequalities:
a<b True if and only if (iff.) a<b
a<=b True iff. a<b or a=b

a>b True iff. a>b
a>=b True iff. a>b or a=b

Beware: a=b is *assignment* not test (test of equality is ==)
Typing helps you avoid this mistake:

```
int a=3;
if (a=3) System.out.println("THEN-PART");
 else System.out.println("ELSE-PART");
```

incompatible types found : int
required: boolean

Boolean operators for comparisons

eqineq.java

```java
class eqineq
{

    public static void main (String[] args)
    {
    int a=2008;
    int b=2007;
    int c=2008;

    boolean test1, test2;


    if (a==b) System.out.println("Equality of a=b");
        else
            System.out.println("a is different from b");


        if (a==c) System.out.println("Equality of a=c");
        else
            System.out.println("a is different from c");


            test1=(a==b);
            test2=(!(a!=b));


        System.out.println("test1 "+test1+" should be equivalent to test2:"+test2);



        if (c>=a)   System.out.println("c>=a");
        if (b<=a) System.out.println("b<=a");

        System.out.println("That is all folks!");



    }

}
```

```
a is different from b
Equality of a=c
test1 false should be equivalent to test2:false
c>=a
b<=a
That is all folks!
Press any key to continue...
```

Boolean operators for comparisons

Boolean comparisons are of type `boolean`

```
class Boolean{
  public static void main(String[] args)
{
  boolean b1  = (6-2) == 4;
  boolean b2 =  22/7 == 3+1/7.0 ;
  boolean b3 = 22/7 == 3+ 1/7;

  System.out.println(b1); // true
  System.out.println(b2); // false
  System.out.println(b3); // true
  }
}
```

(6-2) == 4 evalutes to true but 22/7 == 3+1.0/7 evaluates to false

More on boolean operators: Tables

Unary operator:
NOT !

!	
true	false
false	true

Binary **connector operators**:
AND &&

&&	true	false
true	true	false
false	false	false

OR ||

||	true	false
true	true	true
false	true	false

Priority order for boolean expressions

Lazy evaluation of boolean binary operators:

- If a is **false** we do *not need* to evaluate b in a && b
- If a is **true** we do *not need* either to evaluate b in a || b

lazyevaluation.java

```java
class lazyeval{
    public static void main (String[] args)
    {
        double x=3.14, y=0.0;
        boolean test1, test2;



        // Here division by zero yields a problem
        // But this is prevented in the && by first checking whether the denominator is
        // zero or not
        if ((y!=0.0) && (x/y>2.0))
                    {// Do nothing
                     ; }
                    else
            {// Block

            test1=(y!=0.0);
            test2=(x/y>2.0);

            System.out.println("Test1:"+test1+" Test2:"+test2);
            System.out.println("We did not evaluate x/y that is equal to "+(x/y));
                }


        // Here, again we do not compute x/y since the first term is true
        if ((y==0.0) || (x/y>2.0))
            {// Block
            System.out.println("Actually, again, we did not evaluate x/y that is equal to "+(x/y));
                }



    }


}
```

C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe

```
Test1:false Test2:true
We did not evaluate x/y that is equal to Infinity
Actually, again, we did not evaluate x/y that is equal to Infinity
Press any key to continue...
```

Few remarks

Key difference between **assignment (=)** and **logical test ==**

Do not forget the semi-colon at the end of **Instructions;**

✓ Indent your code and structure it into blocks for clarity

Think of **nested if** control structures

```
if (condition1)
        {BlockT1;}
    else
        {
            if (condition2)
                {BlockT2;}
            else {BlockF2;}
            }
```

Nested if

Nested conditionals (nested if)

nestedif.java

```
1  class nestedif
2  {
3
4  public static void main(String[] arg)
5  {
6  boolean condition1=false;
7  boolean condition2=false;
8  boolean condition3=false;
9
10
11     if(condition1){
12
13                     }
14                         else if(condition2)
15                         {
16
17                         }
18                         else if (condition3)
19                         {
20                         }
21 else
22 {System.out.println("Else here!");}
23
24 }
25
26 }
27
28
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE
Else here!
Press any key to continue...
```

Set *curly brackets* { } to increase code readibility

Loops: While/do for iterations

Structure for iterating

- Process a **single instruction** or a **block** until the given boolean expression is **true** (thus may loop forever... and program may not terminate)
- **Boolean expression is re-evaluated at each round**
- We can exit the loop at any time using the keyword **break;**

```
while (boolean_expression)
    single_instruction;
```

```
while (boolean_expression)
    { block_instruction;}
```

```
do
    { block_instruction;}
while (boolean_expression);
```

At least, the loop is executed once.

Loops: Euclid' GCD algorithm

Greatest common divisor of two integers a and b

While

Do

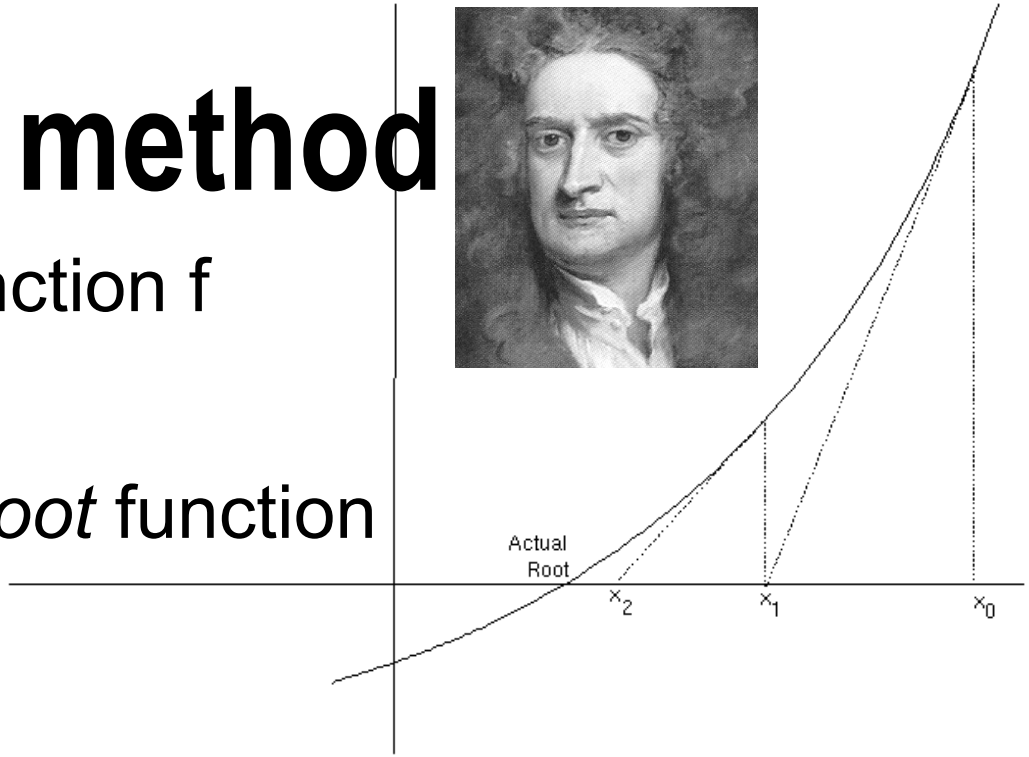
gcd2.java

```java
class gcd2{


    public static void main(String[] arg)
    {

        int a= 231232*4*5*11;
        int b= 123*4*5*11;

        while (a!=b)
    {
        if (a>b) a=a-b;
            else b=b-a;
    }


        System.out.println(a);

        a= 231232*4*5*11;
        b= 123*4*5*11;

        // Now use the do expression in java to perform the same task


        do
            {if (a>b) a=a-b;
            else b=b-a;}
        while (a!=b);

        System.out.println(a);
    }


    }

```

C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe

```
220
220
Press any key to continue...
```

Loops: Newton's method



Converge to a root of the function f

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Use to calculate the *square root* function f(x)=x*x-a

```
double a = 2.0, x, xold;

    x = a;
    do{
        xold = x;
        // compute one iteration
        x = (xold+a/xold)/2;
        System.out.println(x);
    } while(Math.abs(x-xold) > 1e-10);
```

Setting
y=f'(xprev)(x-xprev)+f(xprev)=0
give new value for x

1.5
1.4166666666666665
1.4142156862745097
1.4142135623746899
1.4142135623730949

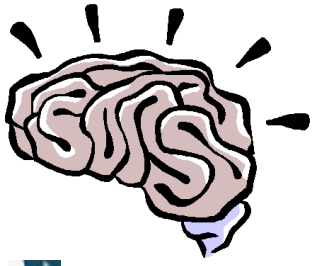
Loops:

```
do
    { block_instruction;}
while (boolean_expression);
```

Syracuse and termination conjecture

Replace x by x/2 (for x odd) and x by 3x+1 (for x even)
Start from any given x, does the replacing alway terminate (x=1)

```
do{
    if ((n%2)==0)
            n/=2;// divide by 2
        else
            n=3*n+1;
  } while (n>1);
```



Nobody knows whether this programs stops for any given input (open problem)
No counter example from simulation so far but no termination proof too!

Loops: Perpetual movements...

Easy to do when programming.... ESC key or Control-C to escape!

Always ensure that loops terminate when programming

```
int i=0;
while (true)
  i++;

for(i=0;i>=0;i++)
   ; // common mistyping error

for(i=0;i>=0;i++)
   { }
```

Loops: Breaking the loop with `break`

Read a sequence of **non-negative natural integers** and compute the cumulative sum of the sequence.

readn.java

```java
class readn{


    public static void main(String[] args)
    {
    int n,sum=0;

    // A forever loop here
    while(true)
    {
        System.out.print("integer ?");
        n=TC.lireInt();  // call function lireInt on class TC

        if (n<0) break;
        sum+=n;  //equivalent to sum+sum+n;

        System.out.println("So far, the cumulative sum is:"+sum);
    }

        System.out.println("The final cumulative sum is:"+sum);

    }
}
```

C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe

```
integer ?4
So far, the cumulative sum is:4
integer ?23
So far, the cumulative sum is:27
integer ?12
So far, the cumulative sum is:39
integer ?71
So far, the cumulative sum is:110
integer ?34
So far, the cumulative sum is:144
integer ?23
So far, the cumulative sum is:167
integer ?76
So far, the cumulative sum is:243
integer ?-5
The final cumulative sum is:243
Press any key to continue...
```

Observe the shortcut:
sum+=n; that is equivalent to assignment **sum=sum+n;**

Loops: For... iterations

- Allows one to execute a block of instructions, and
- Increment the counter at each round
- Semantically equivalent to a while loop
- Often used for program readibility (e.g., for a range of integers)

```
for(instruction1; boolean_condition; instruction2)
block_instruction3;
```

Equivalence with While construction

```
instruction1;
while (boolean_condition)
{block_instruction3;
instruction2;}
```

Loops: For... iterations

for(instruction1; boolean_condition; instruction2)
block_instruction3;

```
class ForLoop
{
   public static void main(String args[])
   {
   int i, n=10;
   int cumulLoop=0;

   for(i=0;i<n;i++) cumulLoop+=i;

   int cumul=(n*(n-1))/2; // closed-form solution
   System.out.println(cumulLoop+" closed-form:"+cumul);
   }
}
```

We get 45

Loops: For... iterations (unlooping)

```
int cumulLoop=0;
for(i=0;i<n;i++) cumulLoop+=i;
```

Unlooping...

```
int cumulLoop=0;
i=0; // Initialization
cumulLoop+=i;
i++; // i=1 now
// i<n so we continue...
cumulLoop+=i;
i++; // i=2 now
// i<n so we continue...
cumulLoop+=i;
...
cumulLoop+=i; // i=n-1
i++; // i=n now
// i is not i<n so we stop...
```

Examples of `for` loop: IsPrime

Program that determines whether a given integer is prime or not.

isprime.java

```java
class isprime{
    public static void main(String[] args)
    {
        System.out.print("Enter an integer please:");
        long k=0,n=TC.lireLong(); // reads a long number
        boolean prime=true;

        if ((n==1) || (n>2 && n%2 ==0) || (n>3 && n%3==0))
                prime=false;
                else
                {
                     k=(long)(Math.sqrt(n)+1);

                    for(long i=5; i<k;i=i+6)
                    if ((n%i==0) || n%(i+2)==0)
                            {
                            prime=false;
                            System.out.println("Exit the loop with k="+k);
                            break;}

                }



        // Output result to console

        if (prime)
        System.out.println("Number "+n+" is prime");
        else
        System.out.println("Number "+n+" is NOT prime.");

        }
}
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.
Enter an integer please:23121971
Number 23121971 is prime
Press any key to continue...
```

```
Enter an integer please:209
Exit the loop with k=15
Number 209 is NOT prime.
Press any key to continue...
```

209=11x19

Multiple choices: `switch`

Avoid nested if-else structures for multiple choices

```
class ProgSwitch
{public static void main(String arg[]){
     System.out.print("Input a digit in [0..9]:");
     int n=TC.lireInt();

     switch(n)
     {
     case 0: System.out.println("zero"); break;
     case 1: System.out.println("one"); break;
     case 2: System.out.println("two"); break;
     case 3: System.out.println("three"); break;
     default: System.out.println("Above three!");
             break;
     }}}
```

Natural integers and int

Difference between mathematics (infinite precision) and computing.

Computing: ***discrete algorithms*** working on *finite representations* of numbers

Source of many bugs !!!

Typically, an algorithm can be correct but its implementation buggy because of *numerical errors*.

int: maximum machine-representable int is 2^31-1
(in the old days, written as 2**31-1)

long: maximum machine-representable long is 2^63-1

Overflow problems...

A toy example 

overint.java

```java
class overint{

    public static void main(String[] args)
    {

        int n=64;

        long s=1;
        int i;

        for(i=1; i<=n; i=i+1)
        { s=2*s;
         System.out.println(i+":"+s);
        }
    }


}
```

C:\PROGRA~1\XINOXS~1\JCREAT~1\G

```
41:2199023255552
42:4398046511104
43:8796093022208
44:17592186044416
45:35184372088832
46:70368744177664
47:140737488355328
48:281474976710656
49:562949953421312
50:1125899906842624
51:2251799813685248
52:4503599627370496
53:9007199254740992
54:18014398509481984
55:36028797018963968
56:72057594037927936
57:144115188075855872
58:288230376151711744
59:576460752303423488
60:1152921504606846976
61:2305843009213693952
62:4611686018427387904
63:-9223372036854775808
64:0
Press any key to continue...
```

Computes 2^s, but at some point 2^64 cannot fit 64-bit, we get first
- negative number (leading bit set to 1 by the arithmetic logic unit - ALU)
- then zero!!!!

Overflow problems: revisited

Increment:
i++
or
++i

overint2.java

```
1  class overint2{
2
3      public static void main(String[] args)
4      {
5
6          int n=64;
7          long s=1;
8
9
10         for(int i=1; i<=n; i++)
11         {
12         s=s<<1; // bit operation
13         System.out.println(i+":"+s);
14         }
15     }
16
17
18
19 }
```

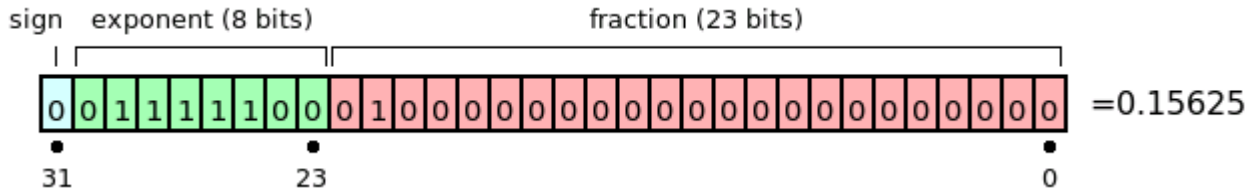
Declaration inside the For

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE
41:2199023255552
42:4398046511104
43:8796093022208
44:17592186044416
45:35184372088832
46:70368744177664
47:140737488355328
48:281474976710656
49:562949953421312
50:1125899906842624
51:2251799813685248
52:4503599627370496
53:9007199254740992
54:18014398509481984
55:36028797018963968
56:72057594037927936
57:144115188075855872
58:288230376151711744
59:576460752303423488
60:1152921504606846976
61:2305843009213693952
62:4611686018427387904
63:-9223372036854775808
64:0
Press any key to continue...
```

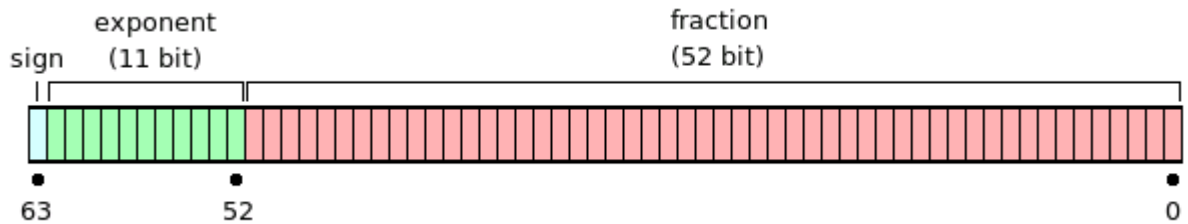
Multiplication is just a *bit shift* << to the right

Floating points & numerical precisions

32-bit IEEE 754 floating point.



64-bit IEEE 754 floating point.



- **float** (32-bit) or **double** (64-bit) have sign, exponent and matissa parts
- Examples: float a=0.3; float b=2e-8 (scientific engineering); float c=1.1f;
- Math class contains important "constants": Math.PI, Math.E, etc.
  and transcendental functions: Math.log(), Math.exp(), Math.sin(), Math.cos()

mathfunc.java

```
1  class transfunc{
2
3      public static void main (String[] args)
4      {
5
6          System.out.println(Math.log(Math.E));
7          System.out.println(Math.cos(Math.PI));
8
9      }
10
11 }
```

C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe

```
1.0
-1.0
Press any key to continue...
```

http://en.wikipedia.org/wiki/IEEE_floating-point_standard

Loosing numerical precision...

A bug

$$ax^2 + bx + c = 0, \qquad x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

doubleprecision.java

```java
class doublepres{

    public static void main (String[] args)
    {
        double a,b,c,d;

        a=0.3;
        b=2.1;
        c=3.675;


        // determinant:
        d=b*b-4.0*a*c;

        System.out.println("Discriminant:"+d);

        if (d==0.0) System.out.println("Correct computation: double roots (discriminant is zero)");
        else System.out.println("I did some numerical roundings somewhere and got the wrong result");


    }
}
```

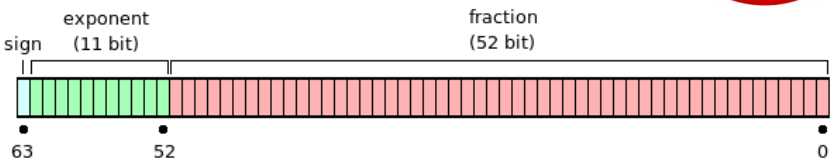
$$\Delta = b^2 - 4ac,$$

C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe

```
Discriminant:8.881784197001252E-16
I did some numerical roundings somewhere and got the wrong result
Press any key to continue...
```

Usually, difficult to
test for the zero
(use threshold or better analysis)

Loosing associativity rule



Rounding & truncations to fit the standard yields the loss of associativity

looseassoc.java

```
1  class looseassoc{
2
3      public static void main (String[] args)
4      {
5          double x,y,z;
6          double u,v;
7
8          x=1.0e35; y=-1.0e35; z=-1.0;
9
10         u=x+(y+z);    // overflow
11         v=(x+y)+z; // does not refit the mantissa
12
13         System.out.println("Value of u is:"+u);
14         System.out.println("Value of v is:"+v);
15     }
16
17 }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
Value of u is:0.0
Value of v is:-1.0
Press any key to continue...
```

Better to add numbers having already the same exponent decomposition...

Computing Euler-Mascheroni 's constant

$$\gamma = \lim_{n\to\infty}\left[\left(\sum_{k=1}^{n}\frac{1}{k}\right)-\log(n)\right] = \int_1^\infty \left(\frac{1}{\lfloor x\rfloor}-\frac{1}{x}\right)dx.$$

$$\lim_{n\to\infty}(H_n - \ln n),$$

euler.java

```java
class euler
{
    public static void main (String[] args)
    {
        double cumul=0.0;
        int n=1000000;
        double ti;

        for(int i=1;i<=n;i++)
        {
            ti=(double) i; // cast: We change format of numbers!!!
            cumul=cumul+1.0/ti;
        }

        double gamma=cumul-Math.log(n);

        System.out.println("Euler's constant:"+gamma);
    }

}
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
Euler's constant:0.5772161649007153
Press any key to continue...
```

Number of known decimal digits of γ

Date	Decimal digits	Computation performed by
1734	5	Leonhard Euler
1736	15	Leonhard Euler
December 8, 2006	116,580,041	Alexander J. Yee[5]
July 15, 2007	5,000,000,000	Shigeru Kondo (claimed)[6]

Types and conversions: Cast operator

- All variables and constants are typed: Math.PI is a (static) double
- Variables should be declared according to their type: double x; int i; etc
- The type of variable **determines** the operator and meaning:
  Exempla gratia, 3.0+2.1 (double) or "Hello "+" INF311" (String)
- The expression is also typed (by the compiler)
- For **assignment =**, the left-hand side (variable) and right-hand side (expression) should have the **same type**.

Casting types
with parenthesis (type):

double x=3.14;
int i=(int)x;

double x=(double)i;

casting.java

```
1  class casting{
2
3      public static void main(String[] args)
4      {
5          double x=3.14;
6          int i=(int)x;
7
8          System.out.println("x="+x);
9          System.out.println("i="+i);
10
11         // Does not round but truncate
12         double y=2.71;
13         int j=(int)y;
14
15         System.out.println("y="+y);
16         System.out.println("j="+j);
17
18     }
19
20 }
```

C:\PROGRA~1\XINOXS~1\JCREAT~1

```
x=3.14
i=3
y=2.71
j=2
Press any key to continue...
```

Converting strings to numbers...

Use Type.parseType(String stringtoparse)

convertstring.java

```java
class convertstring{

    public static void main (String[] args){

        String s1="23122008";
        String s2="1234567890123456";  // if it is too long, it will produce an error!
        String s3="6.02214179E-23";

        // Parse strings to number according to selected type
        int i=Integer.parseInt(s1);
        long j=Long.parseLong(s2);
        double x=Double.parseDouble(s3);

        // Here, we do the converse: numbers to strings
        System.out.println(i);
        System.out.println(j);
        System.out.println(x);

    }
}
```

C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe

```
23122008
1234567890123456
6.02214179E-23
Press any key to continue...
```

A glimpse at functions

Declaring functions in Java

```
class INF311{

public static  typeF F(type1 arg1, ..., typeN argN)
            {
            // Description
            Block of instructions;
            }
        }
```

- This kind of function is also called a **static method**
- Functions must be defined **inside classes**
- A function not returning a result has type **void**
  (also known as a procedure)

Defining the body of a function in Java

```
class INF311{

public static  typeF F(type1 arg1, ..., typeN argN)
                 {
                 // Description
                 Block of instructions;
                 }
           }
```

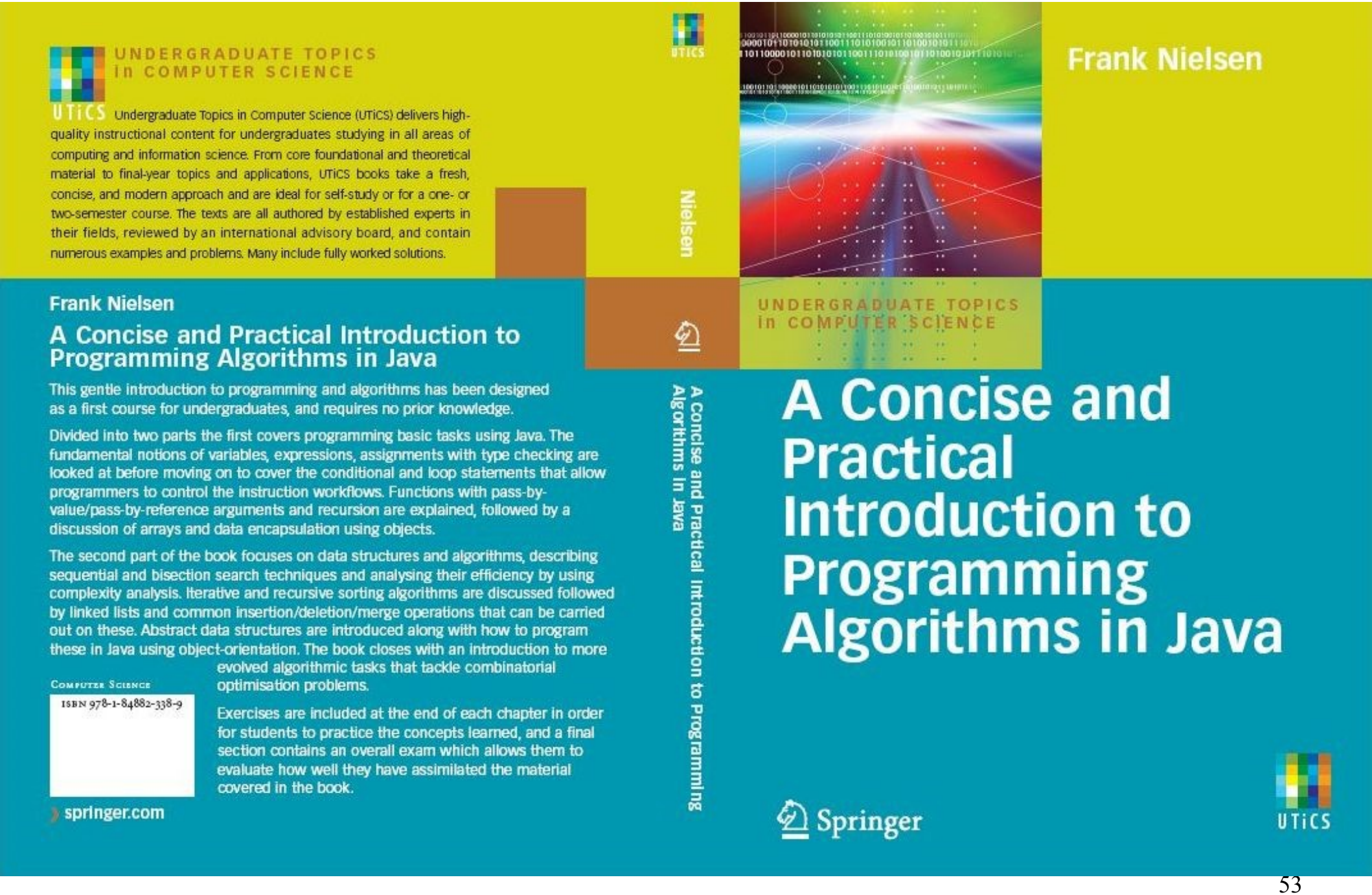
Body of a function

Body should contain an instruction return to indicate the result
If branching structures are used (if or switch) ,
then a return should be written for all different branches.

Otherwise we get a compiler error! (why? => not type safe!)

A few examples of basic functions

```
class FuncDecl{
   public static int square(int x)
                    {return x*x;}

   public static boolean isOdd(int p)
                    {if ((p%2)==0) return false; else return true;}

   public static double distance(double x, double y)
                    {if (x>y) return x-y; else return y-x;}

   public static void display(double x, double y)
                        {System.out.println("("+x+","+y+")");
                         return; // return void
                        }

   public static void main (String[] args)
   {
   display(square(2),distance(5,9));

   int p=123124345;
   if (isOdd(p)) System.out.println("p is odd");
   else   System.out.println("p is even");
   }
}
```


UNDERGRADUATE TOPICS
in COMPUTER SCIENCE
UTiCS Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.
Frank Nielsen
A Concise and Practical Introduction to Programming Algorithms in Java
This gentle introduction to programming and algorithms has been designed as a first course for undergraduates, and requires no prior knowledge.
Divided into two parts the first covers programming basic tasks using Java. The fundamental notions of variables, expressions, assignments with type checking are looked at before moving on to cover the conditional and loop statements that allow programmers to control the instruction workflows. Functions with pass-by-value/pass-by-reference arguments and recursion are explained, followed by a discussion of arrays and data encapsulation using objects.
The second part of the book focuses on data structures and algorithms, describing sequential and bisection search techniques and analysing their efficiency by using complexity analysis. Iterative and recursive sorting algorithms are discussed followed by linked lists and common insertion/deletion/merge operations that can be carried out on these. Abstract data structures are introduced along with how to program these in Java using object-orientation. The book closes with an introduction to more evolved algorithmic tasks that tackle combinatorial optimisation problems.
Exercises are included at the end of each chapter in order for students to practice the concepts learned, and a final section contains an overall exam which allows them to evaluate how well they have assimilated the material covered in the book.
Computer Science
ISBN 978-1-84882-338-9
springer.com
UTiCS
Nielsen
A Concise and Practical Introduction to Programming Algorithms in Java
Frank Nielsen
UNDERGRADUATE TOPICS
in COMPUTER SCIENCE
A Concise and Practical Introduction to Programming Algorithms in Java
Springer
UTiCS