Precourse Computer Science

Programming

FaRaFin

August 29, 2024

FAKULTÄT FÜR INFORMATIK

Why Programming?

Why Programming?

- To give instructions to computers.
- To make abstract concepts understandable to the PC.
- To communicate algorithms.

experience

What are your experiences with programming?

History

- Ada Lovelace (1815-1852) is considered the first programmer
- Alan Turing (1912-1954) is considered the inventor of the computer
- C (1972) is a programming language that is still widely used today
- IBM Personal Computer (1987) is considered the first personal computer
- Java (1996) is the programming language we learn and is also widely used.

Programming Constructs

Variables

- A variable is a reserved memory area.
- All variables have a fixed data type.
- Variables can be assigned a data value.
- Examples of simple data types are:
  - integers, floating-point numbers, boolean values, and characters.

Primitive Data Types

- `boolean` for boolean values
- `int` for integers
- `float` for floating-point numbers
- `char` for characters
- `String` for strings/text

Programming Constructs

- Programmes consist of combinations of the following basic constructs:
  - Sequence
  - Conditional execution / case differentiation
  - Repetition
- This can be used to form *every* programme.

Notations

- Various notations (writing forms) are:
    - Colloquial
    - Pseudo-Code

Sequence

- Successive execution of several instructions

```
Execute statement 1,
then instruction 2,
Execute the last statement N.
```

Sequence

Pseudo-Code

```
Instruction_1
Instruction_2
Instruction_N
```

Instruction

- Instructions are individual commands.
- These include, for example:
  - Output to the console
  - Individual calculation operations.
  - Function calls.

Instruction block

- A statement block is a sequence of one or more statements.
- It is represented in Java by curly brackets

  ```
  {
    statement 1;
    statement 2;
  }
  .
  ```
- This block is also called scope.
- An instruction can also be an instruction block.

Case Distinction: One-sided

- Execution depending on a condition

colloquially

If`condition B` applies,
then execute `statement A`

Case distinction: One-sided

Pseudo-Code

```
If condition B then
    statement A
endif
```

Condition

- A condition is a variable or a function call that returns a variable that can only have two states. (boolean, Wahrheitswert)

TRUE oder FALSE.

- These include, for example::
  - `variable A` is even. (A % 2 == 0)
  - `text T` has 7 characters. (T.length() == 7)
  - `True value W` is TRUE/True.

Case distinction: Two-sided

- Execution also dependent on a condition
- If condition is false, another instruction is executed.

Umgangssprachlich

If `condition` applies,
execute `statement A`,
otherwise execute `statement B`

Case differentiation: Two-sided

Pseudo-Code

```
if condition then
    statement_A
else
    statement_B
endif
```

Case distinction: Multipage

- One of several cases is determined using a selector and the corresponding instruction is executed.
- "‘default"’ is for all other cases that are not queried via the selector
- selector is a variable

Case distinction: Multipage

colloquial

If the `selector S` has the value x_0,
execute `statement for` x_0.
If the `selector S` has the value x_1,
execute `statement for` x_1.
For all other cases, execute `statement K`.

Case differentiation: Multipage

Pseudo-Code

```
if variable_1:
    1:
        statement_1
    2:
        statement_2
else
    statement_n
endif
```

Repetition

- Repetition of instructions
  - Depending on a condition

colloquially

As long as`condition B` *holds, repeat* `statement A`.

while-Loop

- Header-controlled loop.
- Check the condition before starting the loop..

while-Loop

Pseudo-Code

```
while condition_B do
    statement_1
    statement_2
    ...
    statement_n
endwhile
```

do-while-Loop

- Foot-controlled loop.
- Checks the condition after the first run.
- Is executed at least once

do-while-Loop

Pseudo-Code

```
repeat
    Instructions_A
until condition_B
```

for-Loop

- Abbreviation for a while loop
- Common use case: For each value between a and b

colloquially

Execute the statement A
N times.

for-Loop

Pseudo-Code

```
for i := start value to end value do
    statement_1
endfor
```

Case Distinction: One-sided

- Execution depending on a condition

colloquially

If`condition B` applies,
then execute `statement A`

Case distinction: One-sided

Pseudo-Code

```
If condition B then
    statement A
endif
```

Condition

- A condition is a variable or a function call that returns a variable that can only have two states. (boolean, Wahrheitswert)

  TRUE oder FALSE.

- These include, for example::
  - `variable A` is even. (A % 2 == 0)
  - `text T` has 7 characters. (T.length() == 7)
  - `True value W` is TRUE/True.

Case distinction: Two-sided

- Execution also dependent on a condition
- If condition is false, another instruction is executed.

Umgangssprachlich

If `condition` applies,
execute `statement A`,
otherwise execute `statement B`

Case differentiation: Two-sided

Pseudo-Code

```
if condition then
    statement_A
else
    statement_B
endif
```

Case distinction: Multipage

- One of several cases is determined using a selector and the corresponding instruction is executed.
- "‘default"’ is for all other cases that are not queried via the selector
- selector is a variable

Case distinction: Multipage

colloquial

If the `selector S` has the value x_0,
execute `statement for` x_0.
If the `selector S` has the value x_1,
execute `statement for` x_1.
For all other cases, execute `statement K`.

Case differentiation: Multipage

Pseudo-Code

```
if variable_1:
    1:
        statement_1
    2:
        statement_2
else
    statement_n
endif
```

Introduction to Java

What is Java

- Object-oriented programming language
- Developed by Sun since 1993
- Sun was taken over by Oracle in 2010.
- Advantages of Java:
  - Platform-independent program code
  - Simple programming by dispensing with more complex constructs (e.g. pointers)
  - Despite this, no restricted usage options
  - Extensively documented

Basic framework – program structure

- Program example: *HelloWorld.java*

```java
public class HelloWorld {
    public static void main(String[] args) {
        //This program outputs "Hello World
            !".
        System.out.println("Hello World!");
    }
}
```

- Outputs "Hello World!" to the console.

Basic framework – public class

```java
public class HelloWorld {
```

- Every executable program consists of at least one public class (here: *HelloWorld*).
- Class name = file name
  - public class HelloWorld ⇔ HelloWorld.java

Basic framework – Main method

```
public static void main(String[] args) {
```

- jump-in point of the Java interpreter
- Each executable program has exactly one main method
- *args* contains the command line parameters

Basic device – Keywords

- Keyword:
  - Words reserved by Java that control the program flow
  - Key words must never be used as variable or method names
  - e.g: *if*, *else*, *public*, *private*, *static*, *void* . . .

Basic framework – Statements

- Statements:

```
System.out.println("Hello World!");
```

  - Statements in Java are ended with "‘;"’.
  - This statement outputs "Hello World!"’ on the console.

Basic framework – Comments

- Comments:

```
// This program outputs "Hello World!"
```

  - Comments are ignored by the compiler
  - Single-line comments are introduced with //
  - Multiline comments:

```
/* This program outputs "Hello World!" */
```

Variables and data types

- Variable = other name for memory area → a container for a specific data value
- The content of a variable can be changed.
- Variables must be declared!
  - Syntax:

```
type variable name;
```

  - e.g.:

```
int variable;
```

- Cannot be used without prior declaration!

```
// int variable;
variable = 2; // Compilation error
```

Basics of the Statements

- An Statement is a single step that the computer can execute.
- In Java this is:
  - Variable declaration (possibly with value assignment)

```java
int variable;
float variable2 = 13.5f;
```

  - a value assignment

```java
variable = 13;
```

  - a method call

```java
System.out.println("Output text");
```

Value assignment of a variable

A value assignment of a variable can consist of:

- fixed value

```
variable = 13;
```

- Method call

```
variable = Math.pow(2,5);
```

- Other Variables

```
variable = variable2;
```

- Calculation

```
variable = (variable2 + Math.pow(2,5)) * 13;
```

The calculation is based on dot-before-dash calculation, bracketing is possible.

Arithmetic operations

The following arithmetic operations are possible:

operation	**example**	**comment**
Addition	$a + b$	only with numbers
Subtraction	$a - b$	only with numbers
Multiplication	$a * b$	only with numbers
Division	$a \ / \ b$	is integer division if a and b are integers
Modulo	$a \ \% \ b$	calculates the remainder of the division a/b

Strings

- Object of the class String
- is used to store character strings
- Creation of a new string:
  - Assign literal

```
String str = "Hello";
```

Operations on strings

- Concatenation (stringing strings together) with $+$

```java
String a = "This is " + "a test";
String b = a + " with strings";
```

Truth Values

- A truth value is something that is either true or false.
- In Java, a truth value is a value of the data type "boolean".
- A boolean value can be created by:
  - fixed value

```
boolean b1 = true;
```

  - a comparison

```
boolean b2 = variable == 2;
```

Comparisons

- The following comparisons for primitive data types are possible:

Comparison	Example	Comments
Equality	$a == b$	a and b must be of the same type
Unequality	$a \mathrel{!=} b$	a and b must be of the same type
Greater	$a > b$	a and b must be numbers
Smaller	$a < b$	a and b must be numbers
Greater than or equal to	$a >= b$	a and b must be numbers
Less than or equal to	$a <= b$	a and b must be numbers

Logical Connectives

- Logical operations can only take place between boolean values.
- Nesting and parenthesis is possible.
- The following connectives are possible:

Linking	**Code**	**Condition**
NOT	$!a$	if a is untrue
AND	$a\&\&b$	if a and b are true
OR	$a\|\|b$	if a or b is true

if

- One-sided case differentiation

```
if (Bedingung) {
    Statements;
}
```

- E.g.

```
if (x == 3) {
    System.out.println("x is 3");
}
```

```
boolean condition = true;
if (condition) {
    System.out.println("condition is true");

}
```

if-else

- Two-sided case differentiation:

```
if(Condition) {
    Statements;
} else {
    other statements;
}
```

- z.B.

```
if(x == 3) {
    System.out.println("x is 3");
} else {
    System.out.println("x is not 3");
}
```

switch-case

- Multilateral case differentiation:
- Compare a variable with constant values.

```
switch(variable) {
    case Constant_Value_1: Statements;
        break;
    case Constant_Value_2: Statements;
        break;
    ...
    default: Statements;
        break;
}
```

- Expression is numeric, not logical!

switch-case

```java
int x = 1;
switch(x) {
    case 0:
        System.out.println("x is 0");
        break;
    case 1:
        System.out.println("x is 1");
        break;
    default:
        System.out.println("x is was anderes");
        break;
}
```

Shortened Operators

- Shorter notation for variable operators.

```
int a = 1;
int b = 1;
a += 5; // a = a + 5;
b -= a; // b = b - a;
a *= b; // a = a * b;
b /= 5; // b = b / 5;

a++;    // a = a + 1;
b--;    // b = b - 1;
```

while-loop

- As long as condition is fulfilled do statement.

```
while (Condition) {
    Statements;
}
```

```
int n = 0;
while (n <= 10) {
    System.out.println("2 high" + n + "is"
    + Math.pow(2,n));
    n = n+1;
}
```

do-while-Loop

- do-while-:
- when the statements are to be executed at least once.

```
do {
     Statements;
} while (condition);
```

```
int n = 0;
do {
     System.out.println("2 high "+n+" is "
     + Math.pow(2,n));
     n=n+1;
} while (n <= 10);
```

for-Loop

- for-Loops
- When the number of runs is known beforehand.

```
for(Initialisation; condition; Change) {
     Statements;
}
```

```
for(int n=0; n<=10; n=n+1) {
     System.out.println("2 high "+n+" is "
     + Math.pow(2,n));
}
```

for- as while-Loop

- Every for loop can also be expressed as a while loop.
- Therefore:

```
for(Initialisation; condition; Change) {
    Statements;
}
```

- can be expressed as:

```
Initialisation;
while(condition) {
    Statements;
    Change;
}
```

Beware of Infinite Loops

- Attention: With loops, you must ensure that the termination condition is met!
- E.g.:

```
int n=0;
while(n <= 10) {
     n=n-1;
}
```

- Since n runs against $-\infty$, the cancellation condition n > 10 is never reached $\rightarrow$ Endless loop.
- So always check iteration step and termination condition.

User input via scanner

- To be able to make user input from the console, we use an object of the Scanner class.
- To use the Scanner class, we need to include the system source java.util.Scanner.
- We can address the console via System.in.
- The documentation of the Scanner class can be found here: `https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html`

Methods of Scanner class

```java
Scanner scn = new Scanner(System.in);
String string1 = scn.next();
//reads the input up to the next space


String string2 = scn.nextLine();
//reads the input up to the next Enter

int a = scn.nextInt(); //reads the integer

long b = scn.nextLong(); // read in long

double c = scn.nextDouble(); //Read in double

scn.close(); //Release the input stream
```

Example 1: Reading of strings

```java
import java.util.Scanner;

public class ScannerDemo {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        System.out.println("Enter a string");
        String text = scn.nextLine();
        System.out.println("Enter another string
            ");
        String text2 = scn.nextLine();

        scn.close(); //Release the input stream
    }

}
```

Example 2: Reading of whole numbers

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    //enter several integers in one line.
    System.out.println("enter 2 integers " +
    "at once");
    int a = scn.nextInt();
    int b = scn.nextInt();

    scn.close();  //Release the input stream
}
```

- Attention: This code does not check whether the user really enters whole numbers!

Java Continuation

Data type sizes

type	size (bit)	range
boolean	1	$\{true; false\}$
byte	8	-128 to 127
short	16	-32.768 to 32.767
int	32	$-2,147,483,648$ to $2,147,483,647$
long	64	$-9,223,372,036,854,775,808$ to $9,223,372,036,854,775,807$
char	16	'\u0000' to '\uFFFF'
float	32	$-3,40282347 \times 10^{38}$ to $3,40282347 \times 10^{38}$
double	64	$-1,79769313486231570 \times 10^{308}$ to $1,79769313486231570 \times 10^{308}$

Overflow or Underflow

- "Overflow or Underflow
  - Variables have a fixed value range.
  - If the value exceeds or falls below this range, the new value can no longer be saved correctly.
    $\rightarrow$ "Overflow or Underflow.
  - Instead, the process continues at the other end of the value range.
  - Java does not recognise an error.
  - Example

```java
int x = 2147483647; //maximum possible
    value
x = x + 1; //overflow x = -2147483648
```

Casting

- converts basic data types into each other, e.g. double to int.
- only works with basic data types → no strings, objects or complex data types
- Example int to float:

```
int intValue1 = 5;
int intValue2 = 4;
float dValue;
dValue = (float) intValue1 / (float)
    intValue2;
//Floating comma instead of integer division
```

- Example float to int:

```
float dValue = 3.893926;
int iValue;
iValue = (int) dValue; //iValue = 3;
```

Example: Creating random numbers

- The following programme creates integer random numbers between 50 and 100.
- Math.random(); generates a random number x, where 0 $<=$ x $<$ 1.

```
int smallestNumber = 50;
int numberofpossiblenumbers = 51;
int random = (int)(Math.random() *
numberofpossiblenumbers) + smallestNumber;
```

Arrays

- An array consists of several variables of the same type, e.g.

int	int	int	int	int	int	int

- The size of a field is defined during initialisation and is fixed. In the example above, the length is seven.

Creation of Arrays

- creation
  - syntax

```
type[] name = new type[number_elements];
```

  - E.g. a field with twelve double values:

```
double[] field = new double[12];
```

  - Alternatively with initialisation list:

```
int[] field = {1,1,2,3,5,8,13};
```

Access to Arrays

- access:
  The elements of an array are numbered, starting with 0.

```
int[] numbers = new int[6];
```

```
numbers[0] = 13;
numbers[1] = 4;
numbers[5] = 42;
```

- Array size:

```
int a = numbers.length; // a = 6
```

Iterative access to arrays

- access:
  The elements of an array are numbered, starting with 0.

```
Scanner scn = new Scanner( System.in);
int[] numbers = new int[6];
for( int i = 0; i < numbers.length; i++) {
    numbers[i] = scn.nextInt();
}
scn.close();
```

foreach Loop

- The following foreach loop outputs all elements of the array *myArray*:

```
int[] myArray = { 3, 7, 13, 44, 54 };
for (int element : myArray) {
    System.out.println(element);
}
```

- **No assignments can be made here!**
- for comparison the output with a normal for loop:

```
int[] myArray = { 3, 7, 13, 44, 54 };
for(int i=0;i<myArray.length;i++){
    System.out.println(myArray[i]);
}
```

break-statment

- Through a `break` you can exit **a** loop prematurely without executing the rest of the statements in the loop body. The programme continues after the loop.

```java
for (int i=0; i<500; i++) {
    if (i==20){
        break; //exits the loop if i==20
    }
    System.out.println(i);
} // End of the for loop
//After the loop, it continues normally
```

continue-Statment

- `continue` skips the execution of the current loop pass and starts the next loop pass.
- Example: The following code outputs the numbers from 1 to 100 without the number 42.

```java
for (int i=1; i<=100; i++) {
    if (i==42){
        continue;
    }
    System.out.println(i);
}
```

Methods

- Small tasks and algorithms that are often required are written in separate functions (called "methods" in Java).
- e.g. calculate powers or compare strings

Methods

- Analogy to mathematics:
  - $y = f(x)$
    x $\rightarrow$ Parameter
    y $\rightarrow$ Return value
    f $\rightarrow$ Method
  - $f(x) = x^2$

```java
public static double f(double x) {
    return x*x;
}
```

Methods

- Basic structure:

```
Visibility [static] Return type Name(
    Parameter) {
     //source text
     return return;
}
```

- *visibility* can be either **private**, **public** or **protected**.
- `static` can be set, but does not have to be.
  Rule: only static methods can be called directly in static methods (such as main).

Methods

- *return type* can be a class, array, primitive data type or `void`. `void` means "‘no return"’.
- *name* is the name of the method.
- A function can have any number of *parameters*. They are already initialised variables within the function.
- The method is ended by **return**. At the same time, the variable whose value is returned must be specified. Nothing must be specified for **void**.

Conventions for method names

- Method names should be
  - describe what the method does
  - should start with lowercase letters
  - in CamelCase (first letter in lower case, further words in upper case)
- meaningful characters
  - a-z, A-Z, numbers
  - if possible do not use special characters, umlauts and ß
- Examples of good method names:
  - toString(); getNextInputValue(); calculateMean(); isEqual();

Example 1

- A method that checks whether a number is even:

```
public static boolean isEven(int number) {
    if (number % 2 == 0) {
        return true;
    } else {
        return false;
    }
}
```

- shorter is also possible:

```
public static boolean isEven(int number) {
    return number % 2 == 0;
}
```

Example 2

- Power function for positive exponents

```
public static int potenzBerechnen(int base,
int exponent) {
    int potenz=1;
    for(int i = 1; i <= exponent; i++){
        potenz= potenz * base;
    }
    return potenz;
}
```

Call

- Calling the method

```
public class Methods {
    public static void main(String[] args) {
        int result = potenzBerechnen( 2, 10);
        System.out.println("2 to the power of
            10 is equal" + result);
    }

    public static int potenzBerechnen(int
        base,
    int exponent) {...}
}
```

Overloading methods – Example 1

When a method is overloaded, the same method name is used but with more or fewer parameters.

```
1  public static void main(String[] args) {
2      int x1 = 3;
3      int x2 = 1;
4      output(x1); //jumps to line 8
5      output(x1,x2); //jumps to line 12
6  }
7
8  private static void output(int x1) {
9      System.out.println("x1 = " + x1);
10 }
11
12 private static void output(int x1, int x2) {
13     System.out.println("x_n = " + x1 + "\t" + x2
           );
14 }
```

What is a scope?

- A scope is the area in which a variable is visible.
- It is characterised by:

```
{
    Variable;
}
```

- A new memory area is reserved on the stack for each scope.
- Each variable is only visible in its and the subordinate scopes.

- Each name can only be used once in a scope and sub-scopes.
- When the scope is closed, all variables are deleted. (Memory area is removed from the stack)
- A scope is executed as a statement sequence

Examples for Scopes

- methods

```
public static void main () {
     \\ ^ this is a Scope
} \\ <-- it ends here
```

- loops

```
while( true) { \\ <-- this is also a Scope

} \\ <-- this one ends here
```

Scope of a for-loop

```
public static void main( String[] args) {
    for( int i = 0; i < 10; i++) {
        \\ The variable i exists in this scope
    }

    \\ Here again no more
    \\ This means that in a new scope
    \\ the variable i can be used again.

    for( int i = 0; i < 10; i++) {
        \\ There is a new variable i here
    }

}
```

Clean Code

Code Conventions...

... improve the readability of the code and therefore allow a faster and better familiarisation with the code.

Line breaks

- no line longer than 80 characters
- Line breaks:
  1. after commas
  2. before operators
  3. prefer higher-value bracket pairs
  4. align new line to the height of the previous expression
  5. should point 4 lead to unreadable code, indent the following line indent by 1 tab

Line breaks (e.g. 1/3)

- Example for 1. (commas) and 4. (same height)

```java
private static void aMethod(int aParameter,
int nextParameter
) {

}
```

- Example for 2. (before operators) and 4. (same height)

```java
int sumOfFollowingNumbers = number1 + number2
+ number3 + number4;
```

Line breaks (e.g. 2/3)

- Example for 3 (higher-value bracket pairs)

```
float aFloatingNumber = num1 / (num2
- (num3 + num4));
```

as follows **NOT**:

```
float aFloatingNumber = num1 / (num2 - (num3
+ num4));
```

Line breaks (e.g. 3/3)

- Example for 5 (illegible code)

```
if(isThisTrue && isThisTrueToo
|| ThisOneIsTrue) {
  makingThingsHappen();
}
```

as follows **NOT**:

```
if(isThisTrue && isThisTrueToo
|| ThisOneIsTrue) {
  makingThingsHappen();
}
```

Declarations

A declaration introduces a variable, method or class in the source code in the source code. This is necessary so that the compiler knows that a variable with a certain a certain type is used in the code block.

1. one declaration per line
2. if possible initialise variable at its declaration
3. declarations at the beginning of a code block
4. no hidden declarations
5. further hints for the declaration of methods

Declarations (example 1/3)

Example for 1. (one declaration per line) and 2. (initialise immediately):

```java
int aNumber = 0; //nice comment
String aText = "Some text..."; //also with comment
```

as follows **NOT**:

```java
int aNumber, nextNumber; //cannot be nicely
//comment ...
//also, the variables are not initialised
```

Declarations (example 2/3)

Example for:

- 2nd (initialise immediately)
- 3 (declaration at the beginning)
- 4 (no hidden declarations)

```
void aMethod() {
  int counter1 = 0;
  if(somethingHappens) {
    int counter2 = 0;
    // ... some code
  }
  //... and some more code
}
```

hint: `counter1` and `counter2` could also be named the same, but this would violate Note 4.

Declarations (example 3/3)

Example for 5 (further notes)

```
int aMethod(int firstValue, int secondValue) {
  // no space between method name and '('
  // '{' immediately after the parameter list
     separated by
    // separated by a space
    // '}' comes in an extra line
}

// leave a blank line between the methods

void emptyMethod() {} //empty method
//with empty methods the '{' '}' equal
//write one behind the other
```

Java Code Conventions

The *Java Code Conventions* describe the use and spelling of other expressions. Only the most important ones are listed below.

Java Code Conventions

- simple instructions

```java
aNumber += aNumber;
nextNumber += aNumber;
```

As with declarations, you should also ensure that there is only one statement per line.

- if-Statement

```java
if(somethingTrue) {
    doThat();
}
```

Also use curly brackets for single-line `if` blocks!

Java Code Conventions

naming conventions

- Class identifiers should be nouns (`MyObject`, `Date`, `Storage`)
- Method identifiers should be verbs (`run`, `runFast`, `getObject`, `isObjectBlue`)
- Variable identifiers start small and describe the stored value (`tableWidth`, `position`, `myObject`, `date`, `storage`)
- Constants are capitalised and words are separated by an underscore (`MAX_WIDTH`, `MAX_AMOUNT_OF_OBJECTS`, `STD_POSITION`, `BLUE`)

Principle

"‘Good source code needs no comments!"’

(This does **not** mean that you may or should not use any).

Bad code

```
    private void doIt(int a) {int[] b={4,11}; if (a==0)
        {System.out.println("No connection"); c.set(b);c.reload();}else}
    if(a==1){System.out.println("Connection
        is established"); c.set(b);c.reload();}else if(a==2)
    {System.out.println("Connection established");
        int[] b={1,2,3,5,6,7,8,9,15,16,14,13};
        c.set(b); c.beep(); c.reload();} else
    { System.out.println ("INTERNAL ERROR! "
        + "STATUS not recognisable");c.set(b);c.reload();}
}
```

- What should the code do?
- Who will find the error?

Good code

Already better! But not really clean yet.

```
private void doIt(int a) {
    int[] b = {4,11};
    if (a == 0) {
        System.out.println("No connection");
        c.set(b);
        c.reload();
    }else if(a == 1) {
        System.out.println("Connection is established");
        c.set(b);
        c.reload();
    }else if(a == 2) {
        System.out.println("Connection established");
        int[] b={1, 2, 3, 5, 6, 7, 8, 9, 15, 16, 14, 13};
        c.set(b);
        c.beep();
        c.reload();
    } else {
        System.out.println ("INTERNAL ERROR! "
        + "STATUS not recognisable");
        c.set(b);
        c.reload();
    }
}
```

Fully elaborated code

```java
private final static int NOT_CONNECTED = 0;
private final static int TRY_TO_CONNECT = 1;
private final static int CONNECTED = 2;

/**
* Returns the status of the modem connection
* on the console and the modem display.
* (for space reasons, the definition of the display functions is missing).
* @param status - status of the modem connection
*/
private void showConnectionStatus(int status) {
    if (status == NOT_CONNECTED) {
        System.out.println("No connection");
        setDisplayOFF();
    }else if(status == TRY_TO_CONNECT) {
        System.out.println("Connection is being established");
        setDisplayOFF();
    }else if(status == CONNECTED) {
        System.out.println("Connection established");
        setDisplayON();
    } else {
        System.out.println("INTERNAL ERROR! "
        + "STATUS not recognisable");
        setDisplayOFF();
    }
}
```

Final remarks

Comments should be used to explain the task of a method or code section, but not to show its implementation (with the exception of complicated or very complex code sections). This makes it easier to familiarise yourself with the code and find any errors or improvements.

Recursion

Definition Recursion

Recursion, from the Latin *recur = to run back*, refers to the call or definition of a function or structure by itself.

source: http://www.uni-koeln.de/rrzk/kurse/unterlagen/java/spinfo/kap18/rekursiv.htm

Recursion

In recursion, a problem is reduced to sub problems of the same type that are easier to solve than the initial problem. This is continued until a problem is created that is trivial and can be solved with virtually no effort.

Terms

- Recursion descent: Function calls itself.
- Recursion termination: Subproblem is solved and partial result can be returned.
- Recursion ascent: Use partial result to solve larger problem.
- **If the cancellation condition is not met, a stack overflow occurs**

Number of rows

Max is sitting in the last row of a large lecture theatre.
Problem: *Maxwants to know how many rows the lecture theatre has*

- Because he is too lazy to stand up to count all the rows one by one, he asks the person in front of him how many rows there are.
- He in turn asks the person in front of him and so on.
- Until the person in the first row gives back that there are no more rows in front of them.
- The next person adds one more and passes this on.

Until it reaches Max and he knows how many rows the auditorium has.

Example: find file

- If you want to find a file on your computer, you must first search for it in your home directory.
- If you cannot find the file there, you must search through all folders in the home directory in the same way.
- You do this until you have found the file or you have searched through all folders.

Find file as recursion

- Find problem file on computer returned to Find problem in folder.
- Use the same solution scheme for subfolders.
- Cancel when file found or everything searched.

Fibonacci series

$$fib(x \leq 1) = 1$$
$$fib(x) = fib(x-1) + fib(x-2)$$

recursion termination The definition shows that fib of every number less than or equal to 1 is 1. This is therefore the recursion termination.

recursion step The nth Fibonacci number is the sum of the previous two. So this is our recursion step.

fib(x) - Sequence

$fib(3)$

$3 > 1$

- $fib(3) = fib(2) + fib(1)$ **recursive descent**

  $fib(2)$
  $2 > 1$

  - $fib(2) = fib(1) + fib(0)$ **recursive descent**

    $fib(1)$
    $1 = 1$

    - $fib(1) = 1$ **recursive descent**

    $fib(0)$
    $0 < 1$

    - $fib(0) = 1$ **recursion cancellation**

  - $fib(2) = 2$ **recursion cancellation**

- $fib(3) = 3$ **recursion cancellation**

Addition of two Numbers

$plus(x, y) = x + y$ with $x, y \in \mathbb{N}$ structure

recursion cancellation If one of the two values is 0, the other can be returned immediately, as a number added with 0 always results in itself.
Simplification: Recursion cancellation at $y = 0$.

- **plus(x,0) = x**

Recursive descent If $y \neq 0$, then the sum can be formed as follows:

- **plus(x,y) = plus(x+1, y-1)**

(e.g.: $plus(3, 3) = plus(4, 2)$)

plus(x,y) - Sequence

$plus(3,2)$

$2 \neq 0$

- $plus(3,2) = plus(4,1)$ **recursive descent**

  $plus(4,1)$

  $1 \neq 0$

  - $plus(4,1) = plus(5,0)$ **recursive descent**

    $plus(5,0)$

    $0 = 0$

    - $plus(5,0) = 5$ **recursion termination**

plus(x,y) - implementation in Java

```
public static int plus(int x, int y) {
    if (y = = 0) {
        return x;
    } else {
        return plus(x+1, y-1);
    }
}
```

Auxiliary functions

- In some cases the number of parameters for a recursion is too small
- In this case, a helper function is required

This can look like this:

```java
public static int mult(int x, int y) {
    return multHelp( 0, x, y);
}

private static int multHelp(int x, int toAdd,
    int times) {
    // The code for the actual recursion
}
```

Recursion as iteration

- Every recursive function can also be represented by an iterative function, i.e. a function made up of loops and statements.
- Once understood, recursion has the advantage of dividing large problems into small, manageable parts.
- However, it is also a bit slower in comparison to the iterative approach.

Debugging

Basics (1/2)

- Developing programmes is like writing a recipe, the required ingredients (variables) must be specified and also the order in which they are processed (method calls etc.).
- So the programmer swings pen and paper (alternatively keys) and passes the result on to the cook.
- However, the cook is Italian and doesn't even begin to understand what the programmer wants ...

Basics (2/2)

- Consequently, the recipe must be put into a form that the cook can execute.
- We are talking in the Java programming language, the cooking computer only understands machine code (or byte code).
- Compilers take care of translating one language into another. It is nice that it also finds grammatical errors and simplifies awkward formulations, but it is usually left to the programmer to correct his errors.

Compile Java programmes (1/2)

- Java programmes are not executed directly by the hardware, but by a software abstraction layer that ensures platform independence, among other things.
- **Java Virtual Machine (JVM) / Java Runtime Environment (JRE)**
- The format for executable programmes of the JVM is an intermediate between a machine programme and a programming language and can only be interpreted by the latter (e.g. under Windows, Linux, Mac OS X, ...).
- The **Java Development Kit (JDK)** is used for programming.

`http://java.sun.com/javase/downloads/`

Compile Java programmes (2/2)

- After we have written a small programme in the editor, we want to run it. To do this, we call the following command on the console (in the project directory) to "translate" a single file:
  $ javac HelloWorld.java
- The result is executed with:
  $ java HelloWorld
- Please note that although **javac** creates a *.class* file, the extension must not be specified when calling **java**.

Error types

- Syntactical errors
- Runtime errors
- Logical errors

Syntax errors

- also called grammatical errors
- prevents the successful compilation of the programme in question
- examples:
  - misspelled objects and methods (e.g. upper and lower case)
  - incorrectly set brackets
  - incorrect semicolons
  - incorrectly imported packages
- Syntax errors are generally not a problem, they are easy to find and correct.

Runtime errors(1/2)

- only occur when the programme is executed (at runtime)
- Examples:
  - missing but required start parameters (e.g. input file)
  - Read access to uninitialised variable/object
  - Access to non-existent memory locations
  - Illegal type conversion
  - Referencing through null pointer
  - invalid arithmetic operation
  - unreachable code

Runtime errors(2/2)

- If runtime errors occur, the Java runtime throws exceptions, e.g. :
  - **NullPointerException**
    when trying to access a reference that does not point to an object
  - **ArrayOutOfBoundException**
    on invalid field access (index too large or too small)
  - **BufferOverflowException**
    if the buffer limit is reached (endless recursion)
  - **ArithmeticException**
    division by 0 or root of a negative number, ...
- If no exception is thrown, the programme never terminates and is stuck in an infinite loop.

Logical errors(1/5)

- Logical errors are incorrect results or unexpected programme behaviour with correct code.
- Possible solutions:
  - Find out when the error occurs or for which test data the error occurs.
  - Read through or skim the programme and check whether there is a transposed number, an incorrect formula or a simple mental error.
- If the above variants fail, it's time for the actual debugging.

Logical errors(2/5)

- An example:
  Implement a method swap(int a, int b) that swaps the two values a and b with each other.
- First approach:

```
public void swap (int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

Logical errors(3/5)

- When testing the function, it is determined that the values outside the *swap* method are not swapped.
- The *swap* method only swaps the values with each other, but does not save them in the passed variables.
- If you access the original values outside the *swap* method, you get the unswapped values.
- **Java passes the value of an object as a function parameter (also "‘copy by reference"’).**

Logical errors(4/5)

- **"Passing the value**: A copy of the values of the arguments is passed to the parameter list.
- **"Passing by reference**: The memory address of the value of the argument is "passed. If the value is changed by the called routine (function), the change to the value is retained even after the routine is exited.

- Both variants are important!

Logical errors(5/5)

- a solution to the swap problem:
- The *swap* method must be realised with a data type that implements the transfer by reference.

```
public void goodSwap (int[] field) {
     int temp = field[0];
     field[0] = field[1];
     field[1] = temp;
}
```

- This solution uses a field (array).

Historical programm errors (1/2)

- **1962** A missing hyphen in a Fortran programme led to the loss of the Venus probe Mariner 1, which cost over 80 million US dollars.
- **1985-1987** There were several accidents with the Therac-25 medical irradiation device. As a result of an overdose caused by faulty programming and a lack of safety measures, organs had to be removed and three patients died.

Historical programm errors (2/2)

- **1999** NASA's Climate Orbiter probe missed its approach to Mars because the programmers used the wrong measurement system, yards instead of metres. NASA lost the probe, which cost several hundred million dollars, as a result.
- **2002** Siemens mobile phone switched off in April when the calendar function was called up. The error is also known as the April bug. As a result, the mobile phone manufacturer was convinced of the benefits of a user-controlled software update.

Debugging

- When debugging with a debugger, we also speak of debugging. The word bug, for "programming error", was coined by computer pioneer Grace Hopper. Bugfix refers to the correction of a programme error.
- There are various ways to do this:
  - System.out.println();
  - Break points and step-by-step scrolling

Debugging through breakpoints

- A breakpoint is a point defined by the user at which the programme interrupts or pauses execution.
- This gives the programmer the options:
  - to observe values of different variables
  - process the programme step by step or skip steps
  - test conditions

Object Orientation

Object orientation – a brief definition

- programming paradigm based on the concept of object orientation
- basic idea: align software structure with the basic structure of reality i.e. classes as „blueprints“, objects as finished „products“

General principle

- Classes form the abstract blueprint of an object (e.g. a car's blueprint)
- Objects are the realisation/instance of this blueprint (e.g. the finished car) In Java, you create an instance like this:

Properties and methods

- An object has certain properties (attributes) and methods.
- e.g. a car
  has a colour, a number of wheels and a number of doors = attributes and can also drive, flash its lights and do a lot more (object methods)

  This is the basic concept of a car, but it is not yet a specific car!

Confidentiality principle

- Attributes and methods should only be as public as necessary
- most attributes should be private and only accessible via getters and setters
  - Getter: only return the value of the attribute (read only)
  - Setter: change the value of an attribute
- getter and setter can be automatically generated in Eclipse

You already know this

- Every executable programme is a class, where the class name = programme name
- The data type string is a class and you can create a new string object, e.g. with String myString = new String();.
- You have also already met the Math class.
- Java offers many ready-made classes, but you still have to write your own.

Structure of a class (1/2)

- Basic structure:

```
Visibility class name {
  //Attributes
  //Constructor
  //Methods
}
```

```
public class Car{
}
```

Structure of a class (2/2)

- A class has a visibility that determines in which area an object of the class can be created
- Attributes and methods also have a certain visibility (slide Java continuation)

```
//Attribute
visibility [static] type variable name [= default value];

//Methods
visibility [static] return type method name(type parameter,....){
  //code
}
```

Constructor(1/2)

- The constructor „builds“ the object, i.e. it reserves and allocates memory for the object
- It takes over the initialisation of attributes that do not have a default value
- is automatically generated by the JVM if the programmer does not write one
- 2 types of constructors:
  - Default/standard constructor: no parameters
  - Constructors with (any number of) parameters

Constructor(2/2)

- Basic structure:

```
Visibility Class name(parameter){
  //code
}
```

- Example:

```
//Standard constructor
public Car(){
  //code
}
```

Java code example: Car class

```java
public class Car{
  //Attributes
  public String colour;
  public int numberOfDoors;

  //Constructors
  public Car(){
    this.colour = "Black";
    this.numberOfDoors = 4;
  }

  public Car(String c) {
    this.colour = c;
    this.numberOfDoors = 4;
  }

  //Method
  public void honk() {
    System.out.println("Honk!");
  }
}
```

static and non-static

- Methods and attributes can be declared as static
- static means that these attributes/methods can also be called without an object
- i.e. they are properties of an entire class and not of a specific object
- For example, you could use the methods of the Math class without creating a Math object

Examples static and non-static

- Instead of creating an object with Math, you could simply type the class name, a dot and the method you want to use.
- This is also called a class method/variable.

```
Math.pow(2,6);
Math.abs(-3);
```

- The scanner methods, on the other hand, are not static. You need an object to call the method.
- This is also called a member method/variable.

```
Scanner scan = new Scanner(System.in);
int a = scan.nextInt();
```

Creating and assigning objects

- An object is created using the keyword new and a constructor call.

```java
Car myNewCar = new Car();
Car myOtherCar = new Car("Red");

String aString = new String();
```

- A new object is always created with a new constructor call.

```java
//creates a new object of Car
Car myNewCar = new Car();
//does NOT create a new object of Car
Car myOtherCar = myNewCar;
```

Short digression: References

- Non-primitive data types in Java are generally created as references.
- Reference = link to an object
- A reference is therefore an alias for an existing object.
- In this case, myNewCar and myOtherCar are IDENTICAL, they refer to the same object.
- It is, so to speak, the same car with different names.

Accessing methods and attributes

- Access is made using the dot operator
  - if static: classname.method(), classname.attribute
  - if member variable/method: object.method(), object.attribute

Inheritance – General Principle

- Saving code by extending an existing class
- Classes can inherit their attributes and methods to other classes.
- This can be useful, for example, if the inheriting class is a special form of the original class.
- For example, a police car is a special kind of car.

Terms

- Parent class/super class/base class = class from which inheritance occurs (Car)
- Child class/subclass/inheriting class = class that inherits (Police Car)
- In English, we speak of parent and child class or base and derived class.

Syntax in Java

```
Visibility class Child class extends Parent
    class{
  //Class definition
}
```

- The keyword extends indicates inheritance.

Attributes and methods

- Attributes and methods are inherited from the parent class.
- Methods can be overwritten, but they do not have to be.
- @Override ensures that methods are overwritten and that a new method with the same name is not created by mistake (keyword: overloading).

```java
public class PoliceCar extends Car{
  @Override
  public void honk(){
    System.out.println("Ta Tue Ta Ta");
  }
}
```

Visibility in inherited classes

- Public and protected attributes and methods can be accessed from the inherited class without any problems.
- The situation is different for the visibility of private:
  - Child classes cannot directly access the private members of their parent class.
  - Nevertheless, they also indirectly possess these members, since they are also characteristics of the parent class.
  - This problem can be solved with public/protected getters and setters.

Mention: Multiple Inheritance

- Java does not allow direct multiple inheritance, i.e. only one class can be specified after extends.
- To inherit multiple „classes“ anyway, interfaces are used.

this Operator

- The keyword this establishes the reference to the object in which you are currently located.
- For example, a global variable (attribute) can have the same name as a local one (e.g. parameter).

```
public class Example{
  public int attribut;
  public Example(int attribut){
    this.attribut = attribut;
  }
}
```

super Operator

- With super, you access the methods/constructor of the parent class.

```java
public class ChildExample extends Example {
  public int anotherAttribut;
  public ChildExample(int attribut){
    super(attribut);
    anotherAttribut = 5;
  }
}
```

- The super constructor thus initialises all the variables that have been inherited.
- It must be called first in the constructor of the inheriting class.

Laboratory regulations

Laboratory regulations

- Eating and drinking in laboratories is prohibited
- Do not pull or change any cables
- Do not visit any illegal websites, do not make any illegal downloads
- Do not open any pages with anti-constitutional, pornographic or legally prohibited content.
- Do not hack
- Do not program any malware
- Do not send spam
- Close the laboratory door when leaving
- Do not switch off the computer

Linux and Solaris

Introduction

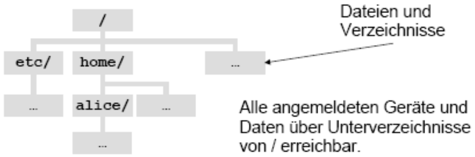
- Login with username and password
- User directories from all workstations can be used in the respective rooms
- Similar kernels in Linux and Solaris
  - Console commands for our needs are the same on both systems

Question for you:

- Why Linux?

Directory structure

- Root directory /
  - All other directories are sub-directories


/
etc/
home/
...
alice/
...
...
...
Dateien und Verzeichnisse
Alle angemeldeten Geräte und Daten über Unterverzeichnisse von / erreichbar.

- "case sensitive"
  - Under UNIX, a distinction is generally made between upper and lower case

Console – What is it?

- text-based user interface to Linux and Solaris
- full system administration and control
- work in current directory
- executes commands

Command Syntax

- Common command syntax:
  (exceptions in “man” or “info” entries):

  ```
  $ command [option] [parameter]
  ```

- Contents of the current directory:

  ```
  $ ls
  ```

- List of the current directory:

  ```
  $ ls -l
  ```

- List of a directory including hidden files:

  ```
  $ ls -la directory
  ```

Help!

- Information on a command by entering

  $ man *command*

  [q] - Ends the help

  or

  $ info *command*
- And if you don't know the name of the programm?

  $ apropos *keyword*
- Automatic completion of file and command names with the tab key

File management – Navigation

- Your own user directory

  /home/*username*

  (short form: ~/)
- Output current directory

  $ pwd
- Change to directory “dir”:

  $ cd *dir*
- Special references
  - **.** - current directory
  - **..** - parent directory

File management – creating & deleting (1/2)

- Creating a new file

  $ touch *file*
- Deleting (removing) a file

  $ rm *file*
- Warning: By default no confirmation[1] and no recovery possible

[1] `-i` turns on confirmation

File management – Creating & deleting (2/2)

- Creating a new directory

  ```
  $ mkdir dir
  ```
- Deleting an empty directory

  ```
  $ rmdir dir
  ```
- Deleting a directory with contents

  ```
  $ rm -r dir
  ```

File management – copying, moving, renaming

- Copying a file to a specific location

  $ cp *source target*
- Moving a file or directory to a specific location

  $ mv *source target*
- Rename a file or directory, where “target” is the new name

  $ mv *source target*

File management – access rights

- Files belong to an owner and a group
- Possible output of *$ ls -l*

```
drwxr-xr-x alice stud 24 2007-09-10 15:09 dir
-rw-r–r– alice stud 1035 2007-09-12 21:25 file.txt
```

  - **d** - directory
  - **rwx** - access rights of owner, group, others (read, write, execute)
  - name of owner and group
  - size, last modified date and file name

File management – access rights

- Changing the owner

  $ chown *user file*
- Changing the access rights

  $ chmod *changes file*
- Symbolic changes: **go+w** gives group and others write access
- Or numerically

  640 gives the owner write and read access, group may read, others nothing

File management – access rights

- Symbolic rights
  - **u** Owner, **g** Group, **o** Others, **a** All
  - + Grant rights, - Revoke rights
  - **r** Read, **w** Write, **x** Execute
- Numerical rights
  - Three digits each consisting of
    4 + 2 + 1
    read + write + execute

File management – text output

- Output the contents of all files one after the other

  $ cat *file file...*
- Output file, waits after each full page

  $ more *file*
- like “more”, back-scrolling possible

  $ less *file*
- Output “text”

  $ echo *text*

File management – Search & Find

- **find** searches for files with certain properties
  - All files in the directory, including sub directories

    $ find *dir*
  - All Java source code files in the directory

    $ find *dir* -name *.java
- **grep** searches for content in files
  - Output lines from Java files that contain the word “test”

    $ grep test *.java

Process management

- **CTRL-C** cancels the current process
- **STRG-Z** puts the currently runnign process to sleep
- **fg** continues the sleeping process
- **ps** shows currently running processes
- **top** opens the Linux process manager

Terminal login

- Connect to another computer via a network

  $ ssh *username*@*computername*

  - Password is requested
  - Terminal, works like a local console
  - Under Windows, you can access the console via ssh using Putty
  - If the username on the calling machine is the same as on the remote computer, *username@* can be omitted.

Transferring files

- Transferring files between computers

  ```
  $ sftp username@computername
  ```

  or

  ```
  $ gftp
  ```

  (graphical variant)
- sftp exists on Windows as well, as graphical variant there are many free programs (e.g. fileZilla)

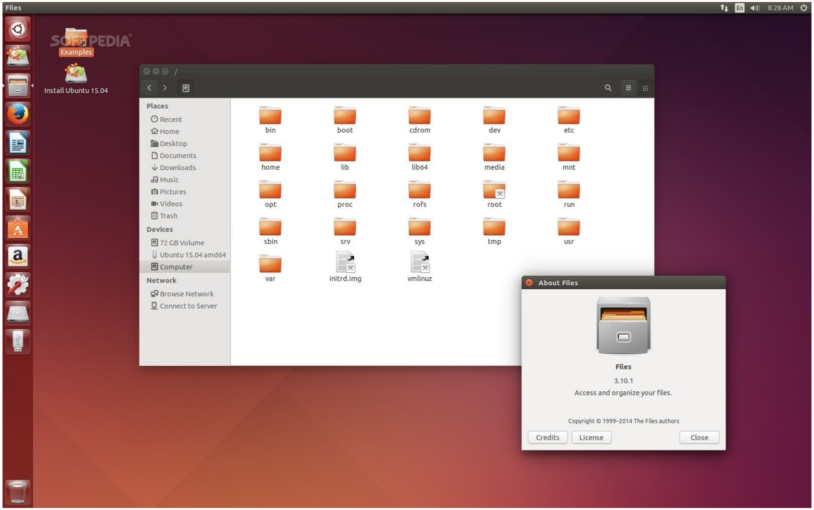
Useful settings

- Set access rights for private files
- Disable the browser cache (to prevent exceeding the memory limit)

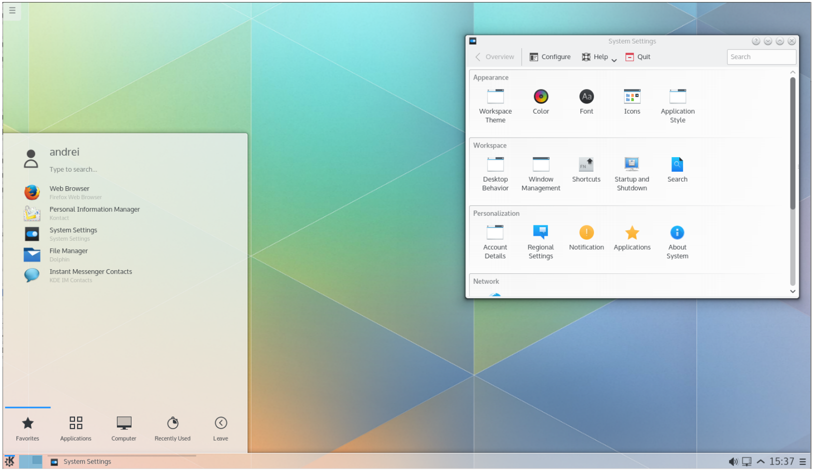
Your Linux!

- Widely used:
  - Ubuntu
  - Debian
  - Fedora
  - Kubuntu (similar to Windows)
  - Linux Mint (similar to Windows)
  - fast, stable, resource-efficient
  - many things are intuitive
  - you can always find help on the web
  - many tutorials contain all the necessary commands
  - easy to update (OS, drivers AND software)

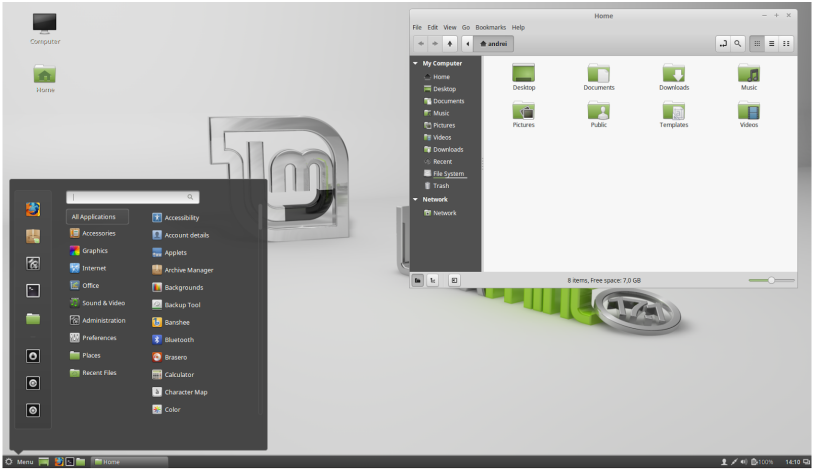
Ubuntu Unity Desktop



Kubuntu Plasma 5 Desktop



Linux Mint Cinnamon Desktop



Updating Ubuntu/Debian

- Admin rights: sudo | sudo -s

apt-get update	=	aptitude update (update package list)
apt-get upgrade	=	aptitude safe-upgrade (update packages)
apt-get install XYZ	=	aptitude install XYZ (install package)
apt-get autoclean	=	aptitude autoclean (clean cache)
apt-get (auto)remove	=	aptitude remove (remove package(s))

- Add a package source (not all are standard):
- sudo ppa-purge ppa:LP-USER/PPA-NAME
- Graphical alternative: Package Manager, Software Center

The following people contributed to these slides:

- Christian Braune
- Markus Durzinsky
- Benjamin Espe
- Kai Friedrich
- Gerhard Gossen
- Darija Grisanova
- Johannes Hintsch
- Matthias Kliche
- Stefanie Klum
- Markus Köppen
- Michael Kotzyba
- René Meye
- Michael Neike
- Twisha Parmar
- Sascha Peilicke
- Julia Preusse
- Michael Schiefer
- Hagen Schink
- Gerd Schmidt
- Martin Zettwitz
- Lydia Rohr
- Manuel Liebchen