

MSc/ICY Software Workshop  
Type Casting, Syntactic Sugar  
Conditionals, Loops, Loop Invariants  
Arrays, ArrayLists

Manfred Kerber [www.cs.bham.ac.uk/~mmk](http://www.cs.bham.ac.uk/~mmk)

Tuesday 15 October 2019

©Manfred Kerber

1 / 27

## Characters and Strings

```
String s;  
s = "Hello, Java";  
With s.length() you get the length of string s.  
With s.charAt(4) the 4th character in the string s. (Careful, we  
start to count with zero!)  
s.substring(0,4) returns a substring of s from the 0th element  
(inclusively) to the 4th element (exclusively).
```

©Manfred Kerber

3 / 27

## Type Casting (Cont'd)

```
d = 123.856778;  
i = (int) d;  
  
i = 123;  
c = (char) i;  
  
c = 'A';  
i = (int) c;
```

©Manfred Kerber

5 / 27

## Some syntactic sugar (Cont'd)

```
(a == b) // tests for pointer equality.  
  
// != stands for not equal  
System.out.println("! (a == b) ==> " + !(a == b));  
System.out.println("(a != b) ==> " + (a != b));
```

For non-base types use `equals` to test for equality, e.g.  
`"Text".equals("Te"+"xt")`

©Manfred Kerber

7 / 27

## Overview

- ❶ Pocket calculator computations, base types, simple strings, variables, static methods, JavaDoc  
Wed/Thu/Fri: 1st Lab Lecture (login, editor, javac, javadoc)
- ❷ Classes, objects, methods, JUnit tests  
Wed/Thu/Fri: 2nd Lab Lecture (Eclipse)
- ❸ **Conditionals, 'for' Loops, arrays, ArrayList**
- ❹ Exceptions, I/O (Input/Output)
- ❺ Functions, interfaces
- ❻ Sub-classes, inheritance, abstract classes
- ❼ Inheritance (Cont'd), packages
- ❽ Revision
- ❾ Graphics
- ❿ Graphical User Interfaces
- ⓫ Graphical User Interfaces (Cont'd)

Changes possible

©Manfred Kerber

2 / 27

## Type Casting

Some types can be converted, some not. Examples are: byte b;  
short s;  
int i;  
long l;

```
float f;  
double d;
```

```
char c;  
l = Long.MAX_VALUE / 48;  
i = (int) l;  
s = (short) i;  
b = (byte) s;  
f = (float) i;
```

©Manfred Kerber

4 / 27

## Some syntactic sugar

```
int a,b;  
a = 10;  
  
a += 5; // += adds the value of the  
        // term to the right to the variable.  
        // It does the same as a = a+5;  
  
a -= 10; // -= subtracts the value of the  
        // term to the right from the variable.  
        // It does the same as a = a-10;  
  
a++;    // increases the value of the variable by 1.  
        // It does the same as a = a + 1;  
  
a--;    // decreases the value of the variable by 1.  
        // It does the same as a = a - 1;
```

©Manfred Kerber

6 / 27

## static, final, public, private

When defining a class we typically initialize objects with variables, e.g. when we define a `BankAccount` we may have an account number and an initial balance. Since the account number never changes, we can enforce this by declaring:  
`final int ACCOUNT_NUMBER = acNumber;`  
Obviously the account numbers should be **different** for different accounts.

©Manfred Kerber

8 / 27

In contrast when we speak about variables which are the same for all objects in a class – e.g., the interest rate could be the same for all accounts of a particular type – we call these variables **static**. E.g., if we want to have a verbose and non-verbose mode for all objects in a class C then we can define this by

```
public static boolean verbose;

```

and access it as

```
C.verbose = false;

```

## Conditionals – if

Remember BankAccount: We want to allow withdrawals only maximally until the balance is zero. How?

*If the condition is true the body is evaluated*

```
if (<cond>){
    <command>*
}
```

In case of one command also

```
if (<cond>)
    <command>
```

Example:

```
if (x < 0){
    x = -x;
}
```

## Nested if statements

```
String str = "evening";
if (str.equals("morning"))
    System.out.println("Have a good day");
else if (str.equals("noon"))
    System.out.println("Enjoy your lunch.");
else if (str.equals("afternoon"))
    System.out.println("Good afternoon, see you soon");
else if (str.equals("evening"))
    System.out.println("See you tomorrow");
else
    System.out.println("Oops. I don't understand.");
```

## while loop

In a **while** loop we distinguish the condition (included in round brackets) and the body of the loop (included in curly brackets). If the condition is true the body is executed repeatedly until the condition is false after executing the body in full.

Syntax

```
while (<cond>) {
    <command>*
}
```

Example:

```
int i = 0 ;
while (i < 7) {
    i = i+1;
    System.out.print(i + " ");
}
```

If something is **static** and **final** we can declare this by

```
static final double CM_PER_INCH = 2.54;

```

If **public** or **private** in addition:

```
public static final double CM_PER_INCH = 2.54;

```

or

```
private static final double CM_PER_INCH = 2.54;

```

## Conditionals – if else

*If the condition is true the 'then' part is evaluated else the 'else' part*

```
if (<cond>) {
    <command>*
} else {
    <command>*
}
```

Example:

```
if (x >= 0) {
    x = Math.sqrt(x);
} else {
    x = Math.sqrt(-x);
}
```

## switch statements

**switch** provides a convenient way to select between different elements of types such as byte, short, char, int, and String data types with the syntax:

```
switch(var){
    case value1: ... break;
    case value2: ... break;
    ...
    default: ... break;
}
```

If the default is missing and none of the cases occurs then the switch statement does nothing.

## for loops

A **for** loop is similar to a while loop, however, in the round brackets we declare and initialize an iteration variable, separate by a semicolon the termination condition and again by a semicolon the update expression for the iteration variable.

Syntax e.g.,

```
for (int i = 0 ; i <= n ; i++) {
    System.out.print(i + " ");
}
```

## A loop

```
int a,b,x,y,u,v;
a = 48;
b = 36;
x = Math.abs(a); y = Math.abs(b);
u = Math.abs(a); v = Math.abs(b);

while (x>y || y>x) {
    if (x>y) {
        x = x - y; u = u + v;
    }
    else if (y>x) {
        y = y - x; v = v + u;
    }
}
```

What does it do?  
Does it terminate?

©Manfred Kerber

17 / 27

## Arrays

'An array is a data structure for storing a collection of values of the same type' [Horstmann, Cornell, Core Java, p.90]  
E.g.,

```
int length = 100;
int[] a = new int[length];
for (int i=0; i < length; i++){
    a[i] = i*i;
}
```

Careful: The lowest index i is 0 the biggest length-1, that is, in this case 99. It is easy to make mistakes involving array bounds. Hence, there should be test cases which check them.

©Manfred Kerber

19 / 27

## Initialization of Arrays

```
int [] c = {2, 5, 24, 6};

for (int element : c){
    System.out.print(element + " ");
}
```

©Manfred Kerber

21 / 27

## Two-dimensional arrays (Cont'd)

Example: TicTacToe taken from [Horstmann, Big Java, p. 288]

```
int rows = 3;
int columns = 3;
String[][] board = new String[rows][columns];

/* board[0][0] board[0][1] board[0][2]
   board[1][0] board[1][1] board[1][2]
   board[2][0] board[2][1] board[3][2] */

for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
        board[i][j] = " ";
    }
}
board[1][1] = "x";
board[2][1] = "o";
```

©Manfred Kerber

23 / 27

## Loop Invariant

Need good documentation of loops, in particular, a loop invariant in order to understand the loops

INVARIANT:  $\text{gcd}(a,b) == \text{gcd}(x,y) \ \&\& \ 2*a*b == x * v + y * u$   
TERMINATION: either x or y is decreased in each run of the loop, x and y will always be positive (assumed we start with positive a and b).

RESULT: if  $x == y$  then  $\text{gcd}(a,b) == (x+y)/2 == x == y$ .  
Hence  $(u+v)/2 == a*b/\text{gcd}(a,b)$ ,  
hence  $(u+v)/2 == \text{lcm}(a,b)$ .

©Manfred Kerber

18 / 27

## Loops on Arrays

There are different possibilities:

```
for (int i=0; i < a.length; i++){
    System.out.print(a[i] + " ");
}
```

Better: write

```
for (int element : a){
    System.out.print(element + " ");
}
```

©Manfred Kerber

20 / 27

## Two-dimensional arrays

Taken from [Deitel and Deitel, Java, 8th ed., 7.9, p. 274]  
A two-dimensional array is an array of one-dimensional arrays.  
`a[row][column]` declared and initialized e.g., by

```
int[][] a = {{1, 2, 3, 4},
             {5, 6, 7, 8},
             {9, 10, 11, 12}};
```

	Column 0	Column 1	Column 2	Column 3
Row 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

with `a[2][1] == 10`

©Manfred Kerber

22 / 27

## Two-dimensional arrays (Cont'd)

Example multiplication table:

```
int length = 12;
int[][] multi = new int[length][length];
for (int i=0; i < length; i++){
    for (int j=0; j<length; j++){
        multi[i][j] = i*j;
    }
}

for (int[] element : multi){
    for (int el : element){
        System.out.print(el + " ");
    }
    System.out.println();
}
```

©Manfred Kerber

24 / 27

Two-dimensional arrays can be initialized in an easy way as shown in the example.

```
System.out.println("INIT FOR A TWO-DIM. ARRAY");
int [][] d = {{ 2, 3, 4},
              { 5, 6, 7},
              {24,25,26},
              { 6, 7, 8}};

for (int[] element : d){
    for (int el : element){
        System.out.printf("%4d ",el);
    }
    System.out.println();
}
// (see PrintfTest.java)
```

ArrayList is not a basic construct, a particular library has to be loaded. This is done by writing at the top of the class file `import java.util.ArrayList;` More information can be found from the API (Application Programming Interface). The API documentation is part of the JDK (Java Development Kit) from Oracle's Java pages.

Comparable to arrays, but without fixed size.

Only on objects. E.g.,

- `ArrayList<String> items = new ArrayList<String>();`
- `ArrayList<String> items = new ArrayList<String>(1000);`
- `items.add("newString");`
- `item.size()` corresponds to length of an array.
- `void trimToSize();` reduce storage size.