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

Manfred Kerber www.cs.bham.ac.uk/~mmk

12 October 2016

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).

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;
```

Type Casting (Cont'd)

```
d = 123.856778;
i = (int) d;

i = 123;
c = (char) i;

c = 'A';
i = (int) c;
```

Some syntactic sugar

```
int a,b;
a = 10;
b = 10;

a += 5;  // += adds the value of the
         // term to the right to the variable
b = b+5;

a -= 10; // -= subtracts the value of the
         // term to the right from the variable
b = b-10;
```

Some syntactic sugar (Cont'd)

```
a--;    // decreases the value of the term by 1
b = b - 1;

a++;    // increases the value of the term by 1
b = b + 1;

(a == b) // tests for pointer equality

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

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.

static, final, public, private (Cont'd)

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;
```

static, final, public, private (Cont'd)

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`

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

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;
}
```

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);
}
```

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.");
```

switch statements

`switch` provides a convenient way to select between different elements of `byte`, `short`, `char`, and `int` 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.

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 + " ");
}
```

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?

Loop Invariant

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

```
INVARIANT: gcd(a,b) == 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 gcd(a,b) == (x+y)/2 == x == y.
        Hence (u+v)/2 == a*b/gcd(a,b),
        hence (u+v)/2 == lcm(a,b).
```

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.

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 + " ");
}
```

Initialization of Arrays

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

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

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	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

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

Two-dimensional arrays (Cont'd)

Example: TitTacToe 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";
```

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();
}
```

System.out.printf
Two-dimensional arrays (Cont'd)

Two-dimensional arrays can be 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

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.

ArrayList (Cont'd)

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.