Introductory Programming
Imperative Programming I, sections 2.0 - 2.9

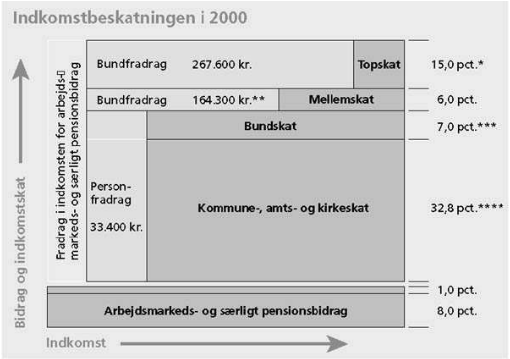
Anne Haxthausen[a]

IMM, DTU

1. Values and types (e.g. **char, boolean, int, double**) (section 2.4)
2. Variables and constants (section 2.3)
3. Statements (e.g. assignments) (section 2.3)
4. Expressions (e.g. arithmetic) (section 2.4)
5. Data conversion (section 2.4)
6. Objects and classes (e.g. `String`) (sections 2.0, 2.1, 2.2, 2.6, 2.7)
7. Output to screen (sections 2.1, 2.9)
8. Input from keyboard (section 2.8)

a. Parts of this material are inspired by/originate from a course at ITU developed by Niels Hallenberg and Peter Sestoft on the basis of a course at KVL developed by Morten Larsen and Peter Sestoft.

Example: Tax year 2000



* 15,0 pct. før evt. reduktion for skråt skatteloft. (13,9 pct. i en gennemsnitskommune i 2000).

** Mellemskattegrænsen forhøjes med 8.000 kr. årligt i 2001 og 2002 ud over den normale regulering.

*** Bundskatteprocenten sænkes yderligere til 6,25 pct. i 2001 og til 5,5 pct. fra og med 2002.

**** Skattesatsen i en gennemsnitskommune i 2000. Skattesatsen varierer fra 27,6 pct. i den billigste til 35,85 pct. i landets dyreste kommune.

Source: *FORSTÅ Skatten...*, Skatteministeriet

Example: Calculation of AMBI and special pension payment

```
public class Tax1 {
  public static void main(String[] args) {
    int income = 120000;
    double ambi, pension;

    ambi = income * 8.0 / 100.0;
    pension = income * 1.0 / 100.0;

    System.out.print("AMBI: ");
    System.out.println(ambi);
    System.out.print("Special pension payment: ");
    System.out.println(pension);
  }
}
```


Output from the Tax1 program

```
AMBI: 9600.0
Special pension payment: 1200.0
```

Basic concepts: values and types

A (data) *value* can for instance be

- an integer (heltal): `120000`
- a floating point number (kommatal): `8.0`
- a character (tegn): `'!'`
- a Boolean (sandhedsværdi): `false` or `true`
- a character string (tegnstreng): `"Bundskat: "`

A *(data) type* is a family of values and operations on these.

Types in Java

`byte, short, int, long` are types of integers: ..., -2, -1, 0, 1, 2, . . .

`float, double` are types of floating point numbers: e.g. -32.3, 1.0, 42.456, , 4.5E6, . . .

`char` is the type of characters: `'a', ..., '1', ..., '!', ..., '\n', ...`

`boolean` is the type of Booleans: `true`, `false`

`String` is a type of strings of characters: `"Bundskat: "`, `"Peter"`,,...

`String` is not a primitive type, but a so-called class. More about that later.

Numeric values

In a computer every numeric value is represented as a binary number, i.e. combinations of bits (0 and 1).
Example: the numeric value 4 is represented as the binary number 100.

The various kinds of numeric values differ by the amount of memory space used to store a value of that type. E.g. the type **int** uses 32 bit. See figure 2.4 of the book.

Literals:
An integer like `17` is of type **int**.
An integer followed by 'L', e.g. 14084591234L, is of type **long**.
A floating point number like `17.2` is of type **double**.
A floating point number followed by 'F', e.g. `17.2F`, is of type **float**.

Characters

In a computer every character is represented as an integer, e.g. 'A' is represented as 65.

A *character set* is a 'translation' from code (integers in the computer) to characters (graphics on a screen or paper).

Standard character sets: ASCII, ISO Latin1, Unicode.
Java uses Unicode.

Exercise: what are the types of the following values?

Value	Type
`58`	
`true`	
`-23`	
`"afd "`	
`42.0`	
`'$'`	
`"42.0"`	
`"true"`	
`'7'`	

Basic concepts: variable, declaration and assignment

A *variable* has a name and a location in the computer memory that can contain a value of a given type.

A *declaration* gives a name and a type for one or more variables:

```
double ambi, pension;
```

The declaration also allocates space for the variables in the memory.

A declaration can contain an *initial value*:

```
int income = 120000;
```

An *assignment (tildelingssætning)* assigns a value to a variable by storing the value in the memory location belonging to the variable:

```
ambi = income * 8.0 / 100.0;
```


Basic concept: constants

A *constant* is similar to a variable, but it has the same value all the time.

Example of declaration:

```
final double AMBI_PROCENTAGE = 8.0;
```

Example of use:

```
ambi = income * AMBI_PROCENTAGE / 100.0;
```


Basic concept: statements

The *state* consists of the contents of the computer memory, output (on the screen), etc.

A *statement* changes the state.

Example 1: an assignment can change the value of a variable:

```
pension = income * 1.0 / 100.0;
```

Example 2: a print statement can write on the screen:

```
System.out.print("Special pension payment: ");
```

Basic concept: expressions

An *expression* denotes (Danish: angiver) a value.

Example:

```
income * 8.0 / 100
```

Kinds of expressions:

- arithmetic expressions (denote numeric values)
- Boolean expressions (denote Boolean values)
- string expressions (denote string values)

Arithmetic expressions

Are combinations of arithmetic *operators* and *operands*.

Operator	Meaning	Examples	
`*`	Multiplication	`1.5 * 60.0`	`24 * 60`
`/`	Division	`13.0 / 2.0`	`13 / 2`
`%`	Remainder	`13.0 % 2.0`	`13 % 2`
`+`	Addition	`1.1 + 60.0`	`14 + 60`
`-`	Subtraction	`1.1 - 60.0`	`134 - 60`

Result type is `int` if both arguments are `int`, otherwise `double`.

Read more on operator precedence and evaluation order in the book.

Boolean expressions

More about that in chapter 3 (next lecture).

String expressions

Like you can write arithmetic expressions representing numeric values, you can write expressions representing strings of characters.

An important string operator:

- string concatenation (+)

Example:

- `"This is " + "course 02100"`
  (represents the string `"This is course 02100"`)

Types of expressions

Each expression has a type.

Examples:

- $1 + 2$ has type **int**
- `"This is "` + `"course 02100"` has type `String`

Basic concept: data conversion

Values of one type can (in certain cases) be converted to another type:

1. Conversions between numerical types:
   (a) "widening": from a smaller type to a larger type (e.g. from **int** to **double**)
   (b) "narrowing": from a larger type to a smaller type (e.g. from **double** to **int**)
2. Conversion from any type to `String` type

In Java 1a and 2 is done *automatically*, but 1b can only be done *explicitly* by *casting*.

Examples of implicit widening conversion

```
double money; int dollars;

//assignment conversion:
dollars = 25;
money = dollars; //now: money == 25.0

//illegal assignment:
dollars = money;

//Second argument (2) of / is converted to a double (2.0):
money  = money / 2; //now: money == 12.5

//Here 2 is not converted:
money = 25.0;
money  = (int) money / 2; //now: money == 12.0
```


Examples of narrowing conversion by casting

```
money = 84.69;
// casting, that throws decimals away:
dollars = (int) money; //now: dollars == 84
```

Example of conversion to a string

```
int x = 7;
System.out.println("the contents of x is: " + x);
```

Second argument of + is automatically converted to the string `"7"`, before it is concatenated with the first argument.

Output: `the contents of x is: 7`

Generally it holds for `s + v` and `v + s`, where `s` is a string and `v` an expression of another type, that `v` is automatically converted to its string representation.

Exercise: which values and types do the following expressions have?

Expression	Value of expression	Type of expression
`1.5 * 60.0`		
`1.5 * 60`		
`24 * 60`		
`1.1 + 60 - 1`		
`150.0 / 60`		
`150 / 60`		
`134.0 % 60`		
`134 % 60`		
`"02199"`		
`"x is equal to "+ "0"`		
`"x is equal to "+ 0`		

Short introduction to classes and objects

Informally

- A *class* represents a concept: time, appointment, car, cow, person, ...
- An *object* represents a thing, an instance of a concept: a particular time, a particular car, a particular cow, a particular person, ...
- A class has a collection of *methods*: those operations (functions) that can be applied to its objects.

In Java

- A *class* corresponds to a type, like `int`, `double`, `boolean`, ...
- An *object* corresponds to a value, like `17`, `18.01`, `false`, ...
- A *method* corresponds to an operation, like $+$, $-$, ...

Three steps in using classes, objects and methods

1. Define a *class* (incl. its methods).
2. Create objects of the class.
3. Use the methods of the objects.

Some classes (e.g. `String`) are already defined in a class library. In that case you can skip step one.

If a class has *static* methods, these can be used without step 2.

Definition of classes

Later you will learn how to *define* a class that contains declarations of:

- *data* (constants and variables)
- *methods* (each method is given a name and a sequence of statements, that have to be executed when the method is *invoked*.)

For now we will only *use* classes from libraries.
For each class there is an *interface* informally describing the methods of the class:
name, argument types and a result type, and what happens when it is *invoked*.

Example: Extract of interface for `String`

int `length()` returns the number of characters in the string

char `charAt(int index)` returns the character at the specified `index`

Creations of objects

An object is an instance of a class. It has data (e.g. a string) and methods as described by the class.

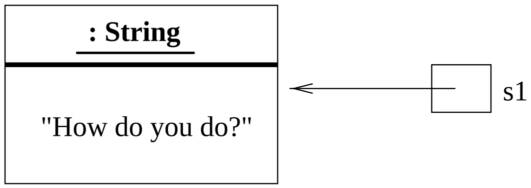
A variable contains either

- a primitive value, or
- a *reference (henvisning)* to an object.

Example: a variable containing a reference to an object of class String

```
String s1 = "How do you do?";
```


Example: an object of class String



The variable s1 is a location in the memory containing a reference to the object containing the string

Use of methods

Invocation (kald) of the methods of an object is done with the *dot operator*.

Example: invocation of a `String` method

```
s1.length();
```

returns the value `14`
(when the object, referenced to by `s1`, has the value `"How do you do?"`).

Class libraries and packages

A *class library* is a collection of classes.

The classes are organized into various *packages*.

The most popular standard packages are:

Package	Supports	Classes
`java.lang`	General stuff; *automatically* imported	`Math`, `String`, `System`
`java.io`	Input and output (e.g. to/from files)	...
`java.util`	General aux. classes	`Random`, ...

To use a class of a package, you must either qualify the class name with the package name, or use an import *statement*, e.g.:

```
import java.util.*; // all classes in the package
import java.util.Random; // only the Random class
```


Output to screen

The methods

- `System.out.print`
- `System.out.println`

can be used to print text on the screen.

The `NumberFormat` and `DecimalFormat` classes provide methods for formatting the output nicely. See section 2.9 of the book.

Input from the keyboard

The `Keyboard` class has methods for reading data from the keyboard

```
static boolean readBoolean()
static byte    readByte()
static char    readChar()
static double  readDouble()
static float   readFloat()
static int     readInt()
static long    readLong()
static short   readShort()
static String  readString()
```

static means that methods can be invoked via their class name, e.g.

`Keyboard.readBoolean()`

The class can be copied from the cd of the book.

Example of input (and output)

```
import cs1.Keyboard;

public class Question
{
  public static void main (String[] args)
  {
    String name; int cars;

    System.out.print("What is your name? ");
    name = Keyboard.readString();

    System.out.print("How many cars do you own, " + name + "? ");
    cars = Keyboard.readInt();

    if (cars>1)
       System.out.println(name + " owns many cars!");
} }
```

Execution of Question

```
> java Question
What is your name? Anne
How many cars do you own, Anne? 1

> java Question
What is your name? Henrik
How many cars do you own, Henrik? 2
Henrik owns many cars!
```