Variables, primitive types, etc.

Chapter 2

Dr Jonas Lundberg, office B3024

Jonas.Lundberg@lnu.se

Slides and Java Examples are available in Moodle

28 oktober 2019

Chapter 2

- Strings and print-methods
- Variables and types
- Primitive types and their operations
- Type conversion
- The `Scanner` class

Readings: Chapter 2 (all of it)

Classes and Objects

`int`, `double`, `char` are simple types with simple values like `7`, `3.14`, `'x'`.
Classes are more complex types and their values are called **objects**.

- Example: The class `BankAccount` and three instances of the class

`class BankAccount`	`Object 1`	`Object 2`	`Object 3`
`Owner:`	`Jonas`	`Henrik`	`Nils`
`No:`	`4758-8696`	`3246-9744`	`5432-2347`
`Saldo:`	`34.345kr`	`8.456kr`	`97.654kr`

- $\Rightarrow$ The class defines the properties of one or more bank accounts.
- $\Rightarrow$ Each object represents one concrete bank account
- Think: One building blue-print can be used to many houses

- Each object belongs to (is defined by) a class
- A class decides the properties of a given type of objects
- We say that an object O_A is an **instance** of class A

Strings

- A **string** is a sequence of characters.
- character = letters, digits, white space, ... (All possible symbols)
- String Examples

```
"Hello World!"      "Sweden is in Europe."      "x"

"(GD&D   .,~-{]HG()B( SG-,.M*'PNI \\\n\t\n"
```

- Each string (created as `"..."`) is an object of class **String**
- The **String** class has many methods.

```
String s = "Hello World!";          // Create string object
int length = s.length();            // Ask object for its length
char first = s.charAt(0);           // ... and first character
...
```

We will look at these methods later on.

ReadySteady.java

```
public class ReadySteady {

    public static void main(String[] args) {
        System.out.print("Ready, ");
        System.out.print("Steady, ");
        System.out.println("Go!");              // breaking first line

        System.out.println("      ....      "); // the race is on

        System.out.println("Hurray! Sweden wins again!!");
    }
}
```

Print out:

```
Ready, Steady, Go!
     ....
Hurray! Sweden wins again!!
```

Print-outs using `println` and `print`

- Methods `println` and `print` print a string to the screen
- Difference: `println` breaks the line, `print` does not
- `System.out` is an object symbolizing the screen
- We call the methods `println` and `print` on the object `System.out`

```
System.out.println("Hello World!");
\_______/ \_____/ \_____________/
     |        |           |
  object    method      argument
```

- The argument (sometimes parameter) is input data to the method

Method Calls in General

- A class A might have methods $m(...)$, $n(..)$, $p(...)$, ...
- $\Rightarrow$ we can call these methods on all objects O_A of class A
- Method call syntax: $O_A.m(...)$

String Concatenation

```
System.out.println("This sentence gives a string that,
                    according to me, is too long to fit   // Error!
                    on a single line.");
```

- You can not break a line in the middle of a string.
- However, you can create a new string by adding two strings
  $\Rightarrow$ this is called **string concatenation**

```
System.out.println("This sentence gives a string that, "
                   +"according to me, is too long to fit "   // OK!
                   +"on a single line.");
```

- Notice: `string + string = new string`
- Also: `string + number = new string`

```
System.out.println("Number of Cars: " + 144);

Print-out:
   Number of Cars: 144
```

- However, it still holds that: `number + number = new number`

Concatenate.java

```
public class Concatenate {
    public static void main(String[] args) {
        // Concatenate five strings
        System.out.println("Do"+"Re"+"Mi"+"Fa"+"So"+"La");

        // Concatenate numbers
        System.out.println("Two integers: "+45+87);

        // Adding numbers
        System.out.println("One integer: "+ (45+87) );
    }
}
```

```
Print-out:
    DoReMiFaSoLa
    Two integers: 4587
    One integer: 132
```

Escape Sequences

```
System.out.println("And then she said: "Hello Darling"."); // Error!
```

- It is interpreted as: `"And then she said: "` followed by something strange $\Rightarrow$ Strings can not contain the character `"`.
- **Escape Sequences**: Characters representing another character
- Most frequently used escape sequences in Java

Escape	Represent
`\"`	"
`\'`	'
`\\`	\
`\n`	line break
`\t`	tab

- That is, we should have written

```
System.out.println("And then she said: \"Hello Darling\".");   /
```

- Linebreak `\n` and tab `\t` are often used to create formatted print-outs.

Escape.java

```
public class Escape {
    // indent (\t) and linebreak (\n)
    public static void main(String[] args) {
        System.out.println("One\n \tTwo\n \t\tThree\n");

        System.out.println("CS Teachers:\n"+"\tJesper\n"+"\tJonas\n"+"\tOla");
    }
}
```

```
Print-out:
One
     Two
          Three

CS Teachers:
     Jesper
     Jonas
     Ola
```

Variables.java

```
public class Variables {

    public static void main(String[] args) {
        String text = "Number of Students: ";   // string variable
        int students = 36;                      // integer variable
        System.out.println(text+students);

        text = "Pi: ";                          // reuse variable text
        double pi = 3.1415926536;               // float variable
        System.out.println(text+pi);
    }
}
```

A bit of vocabulary

- `int v;` ⇒ A variable with name `v` of type `int` is **declared**
- `int n = 7;` ⇒ The variable is declared and **initialized**
- `n = 66;` ⇒ The variable is **assigned** a new value
- `int p = n;` ⇒ We **use** (or **read**) the value in variable `n`

Variables

```
String name = "Jonas Lundberg";
int studentCount = 345;
```

Variables

- A **variable** has a **name** (e.g. `studentCount`) and a **type** (e.g. `int`);
- The name can be used to read (or reference) the current variable value
- A variable must be declared before it is used.
- A variable must be assigned a value before it can be read/referenced

Types

- The variable type decides what values it can be assigned
- The compiler recognizes type errors

```
int studentCount = "Jonas Lundberg";  // Error!
```

- A variable is a name on a memory slot
  - `int n;` $\Rightarrow$ a memory slot with name `n` is reserved (allocated)
  - The type (e.g. `int`) decides the size of the memory slot (e.g. 4 bytes)
  - Assignment (e.g. `n = 24`) $\Rightarrow$ memory slot gets a new binary value

Naming Conventions in Java

Rules ⇒ must be obeyed ⇒ checked by compiler

- Variable names are identifiers ⇒ They can only contain letters, digits, underscore and they can not start with a digit.
- Also, they can not be one of the reserved words (keywords).
  For example, variables named `final`, `class`, `int`, `2x`, `a/b` are not accepted.

Naming Convention ⇒ Recommended but not required

- Variable names should start with a lower case letter
- Separate words by introducing upper case letter for each new word.

```
int studentCount = 345;
int maxSize = 23;
String bestBeforeDate = "2015-12-15";
```

  This type of variable names is called *camel casing*.

Jonas' Additional Recommendations

- Use descriptive names! ⇒ makes your code easier to understand
- Use only the letters a-z, A-Z. Avoid non-standard letters like å,ä,ö.
- Use English if possible. It is very boring to translate all your variable names when you start to collaborate with an international partner.

Constants

Constants using the keyword `final`

```
final int MAX_SIZE = 1200;
int studentCount = 345;
final double PI = 3.1415926536;

studentCount = 256;      // Updating variable value ==> OK!
MAX_SIZE = 1800;         // Updating constant ==> Compile-time Error!
```

- `final int MAX_SIZE` $\Rightarrow$ A constant named `MAX_SIZE` is declared
- Constants can never be changed
- Constants must be assigned a value when they are declared

Naming Convention

- Use only upper case letters
- Separate words by underscore. Example, `MAX_SIZE`

Two types of types

Java divide their types into two categories:

- **Primitive types**: simple types like integer, float, and character
  - There are only 8 primitive types:
    - 4 integer types: `byte`, `short`, `int`, `long`. (Sizes 1-8 byte)
      $\Rightarrow$ The default type for integers are `int`
    - 2 real number types (floats): `float`, `double`. (Sizes 4-8 byte)
      $\Rightarrow$ The default type for floats are `double`
    - 1 character type: `char`. Size 2 byte $\Rightarrow$ $>$ 65000 different characters
    - 1 logical type: `boolean`. Size 1 byte, only two values, `true` or `false`
  - Use the default types (unless you have very good reasons).

- **Reference types**: Aggregated types with more complex properties
  - Each class defines a new reference type
  - An object O_A of class A is a value of type A
  - A string `"Hello"` is a value of type `String`.
  - More about classes and objects later on

Real Number Types

- A **floating point number** (or **float**) is an approximation of a real number
- A float is stored as $m \cdot 2^e$ where m is the **mantissa** and e the **exponent**.
- Think: mantissa gives significant digits and the exponent its size
- `double` $\Rightarrow$ 53 bits mantissa, 11 bits exponent. Both has a sign bit
- Possible `double` values: $\pm 3.4 \cdot 10^{\pm 308}$ with 15 significant digits
- Java uses *dot* (3.14), not a *comma* (3,14).
- **float operations**
  - `+` (Addition)
  - `-` (Subtraction)
  - `*` (Multiplication)
  - `/` (Divide)
- **Important:** Float operations are not exact. Small errors can occur at every operation
  $\Rightarrow$ Significant errors can occur after repeated operations

Integer Types

- We have four integer types with different sizes
- Small size $\Rightarrow$ limited set of possible values
  - `byte`, size 1 byte, values: ± 128
  - `short`, size 2 byte, values: ± 32000
  - `int`, size 4 byte, values: ± 2.1 billions
  - `long`, size 8 byte, values: $\pm 9.1 \cdot 10^{18}$ (Gigantic!!)
- As usual, use the default type `int` unless you have very good reasons.
- **Integer Operations** (Ordinary four + **modulus**)
  - `+` (Addition)
  - `-` (Subtraction)
  - `*` (Multiplication)
  - `/` (Divide)
  - `%` (Modulus)

Integer Division and Modulus

Integer Division $A/B \Rightarrow$ how many B fit inside A?

- Example

$$7/3 = 2, \quad 27/4 = 6, \quad 20/4 = 5, \quad 9/10 = 0, \quad -94/10 = -9$$

- **Exercise**: Compute a) $13/3$, b) $17/5$, c) $4/5$, d) $16/4$

Modulus $A\%B \Rightarrow$ left-overs of A in A/B. Formally

$$A\%B = A - \frac{A}{B} \cdot B$$

- Example

$$15\%4 = 15 - \frac{15}{4} \cdot 4 = 15 - 3 \cdot 4 = 3$$

- More examples

$$7\%3 = 1, \quad 27\%4 = 3, \quad 20\%4 = 0, \quad 5\%10 = 5$$

- **Exercise**: Compute a) $13\%3$, b) $15\%5$, c) $4\%5$, d) $77\%10$

Integer Division and Modulus

Integer Division $A/B \Rightarrow$ how many B fit inside A?

- Example

$$7/3 = 2, \quad 27/4 = 6, \quad 20/4 = 5, \quad 9/10 = 0, \quad -94/10 = -9$$

- **Exercise**: Compute a) $13/3$, b) $17/5$, c) $4/5$, d) $16/4$
- Answer: a) 4, b) 3, c) 0, d) 4

Modulus $A\%B \Rightarrow$ left-overs of A in A/B. Formally

$$A\%B = A - \frac{A}{B} \cdot B$$

- Example

$$15\%4 = 15 - \frac{15}{4} \cdot 4 = 15 - 3 \cdot 4 = 3$$

- More examples

$$7\%3 = 1, \quad 27\%4 = 3, \quad 20\%4 = 0, \quad 5\%10 = 5$$

- **Exercise**: Compute a) $13\%3$, b) $15\%5$, c) $4\%5$, d) $77\%10$

Integer Division and Modulus

Integer Division $A/B \Rightarrow$ how many B fit inside A?

- Example

$$7/3 = 2, \quad 27/4 = 6, \quad 20/4 = 5, \quad 9/10 = 0, \quad -94/10 = -9$$

- **Exercise**: Compute a) $13/3$, b) $17/5$, c) $4/5$, d) $16/4$
- Answer: a) 4, b) 3, c) 0, d) 4

Modulus $A\%B \Rightarrow$ left-overs of A in A/B. Formally

$$A\%B = A - \frac{A}{B} \cdot B$$

- Example

$$15\%4 = 15 - \frac{15}{4} \cdot 4 = 15 - 3 \cdot 4 = 3$$

- More examples

$$7\%3 = 1, \quad 27\%4 = 3, \quad 20\%4 = 0, \quad 5\%10 = 5$$

- **Exercise**: Compute a) $13\%3$, b) $15\%5$, c) $4\%5$, d) $77\%10$
- Answer: a) 1, b) 0, c) 4, d) 7

Integer Division and Modulus

Integer Division $A/B \Rightarrow$ how many B fit inside A?

- Example

$$7/3 = 2, \quad 27/4 = 6, \quad 20/4 = 5, \quad 9/10 = 0, \quad -94/10 = -9$$

- **Exercise**: Compute a) $13/3$, b) $17/5$, c) $4/5$, d) $16/4$
- Answer: a) 4, b) 3, c) 0, d) 4

Modulus $A\%B \Rightarrow$ left-overs of A in A/B. Formally

$$A\%B = A - \frac{A}{B} \cdot B$$

- Example

$$15\%4 = 15 - \frac{15}{4} \cdot 4 = 15 - 3 \cdot 4 = 3$$

- More examples

$$7\%3 = 1, \quad 27\%4 = 3, \quad 20\%4 = 0, \quad 5\%10 = 5$$

- **Exercise**: Compute a) $13\%3$, b) $15\%5$, c) $4\%5$, d) $77\%10$
- Answer: a) 1, b) 0, c) 4, d) 7

Hint: think in terms of distances.

Arithmetic.java

```
public class Arithmetic {
    public static void main(String[] args) {
        double x = 2.5;
        double y = x + 50.0;
        double z = x * y;
        System.out.println("X = "+x + ", Y = "+y+ ", Z = "+z);

        int m = 17;
        int n = 5;
        int div = m / n;
        int mod = m % n;
        System.out.println("\nDivide: "+div+", Modulus: "+mod);
    }
}
Print-out:
    X = 2.5, Y = 52.5, Z = 131.25

    Divide: 3, Modulus: 2
```

A 10 Minute Break

ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ

The Character Type `char`

- A variable of type `char` can store a character
- Remember: character = letter, digit, whitespace, etc.
  ⇒ all keys on the keyboard and many many more
- A character value is enclosed by the symbol '. For example
  `'x', '7', '+', '\n'`
- An example using the String method `charAt(int pos)`

```
public static void main(String[] args) {
    String myName = "Jonas Lundberg";
    char dot = '.';
    char first = myName.charAt(0);     // pick 1st character
    char second = myName.charAt(6);    // pick 7th character

    System.out.println(" Initials : "+first+dot+second+dot);
}
Print-out
    Initials : J.L.
```

- Each string is indexed from 0 till length-1
  ⇒ `myName.charAt(0)` picks 1st character in the string.

The Logical Type `boolean`

- A variable of type `boolean` can only take two values: `true` or `false`

```
boolean a = true;
boolean b = 5+7 < 4+6;   //  ==> b = false
boolean c = 12 <= 3*4    //  ==> c = true
boolean d = 12 == 3*4    //  ==> d = true
```

- The comparison operator == checks for equal values

A few tricky integer operations

- Increment: `n++;` `<==>` `n = n + 1;`
- Decrement: `n--;` `<==>` `n = n - 1;`
- Add assignment: `n += 5;` `<==>` `n = n + 5;`
- Subtr. assignment: `n -= 5;` `<==>` `n = n - 5;`
- Mult. assignment: `n *= 5;` `<==>` `n = n * 5;`
- Div. assignment: `n /= 5;` `<==>` `n = n / 5;`
- Mod. assignment: `n %= 5;` `<==>` `n = n % 5;`

Mixed Type Expressions

- Java is a **strongly typed** language
  $\Rightarrow$ we can not mix types in expressions anyway we like
- Mixing types is however sometimes necessary
- What is the type of a mixed type computation (expression)?

```
short s = 27;
int n = 124;
??? res = s + n;
```

```
double total = 23.47;
int count = 8;
??? average = total/count;
```

- Mixed types in a computation $\Rightarrow$ result is of the *larger* (or *wider*) type.
- **Type Size Comparision**
  - *byte* $<$ *short* $<$ *int* $<$ *long*
  - *float* $<$ *double*
  - *int* $<$ *double*
- Hence, *short* + *int* $\Rightarrow$ *int* and *double*/*int* $\Rightarrow$ *double*

Type Conversion

- Converting from one type to another is sometimes necessary
- We observe two different cases:
  1. Safe (or Widening) conversions where no information can get lost

     ```
     short s = ... ;     int n = ... ;    double total = 23.47;
     int n = s;          double d = n;    int count = 8;
                                          double average = total/count;
     ```

     These conversions are allowed since they *always* work
     $\Rightarrow$ it is based on the types involved, not their current values

  2. Unsafe (or Narrowing) conversions where information might get lost

     ```
     double d = 24.56;       int n = 54 ;          int n = 1234;
     int n = (int) d;        byte b = (byte) n;    byte b = (byte) n;

     ==> n = 24              ==> b = 54;           ==> b = -46 (Error)
     ```

     These conversions are dangerous and might fail. Java forces us here, e.g. by `(byte)`, to show that we are aware of these problems.
- The procedure `byte b = (byte) n` is a called **down casting**.

The `Scanner` Class

- We can read input from the keyboard using the `Scanner` class

```
import java.util.Scanner;    // Get Scanner from Java Library

public class ScannerIntro {
    public static void main(String[] args) {
        /* Create a Scanner object connected to the keyboard */
        Scanner sc = new Scanner(System.in);

        System.out.print("Type a line of text: ");   // user instruction
        String text = sc.nextLine();                 // read string
        System.out.println("Echo: "+text);           // print string

        sc.close();    // close scanner
    }
}
```

```
Print-out:
    Type a line of text: Java programming is fun!
    Echo: Java programming is fun!
```

The `Scanner` Class (cont.)

- The `Scanner` class is in the library package `java.util`
  $\Rightarrow$ We make it available by: `import java.util.Scanner;`
- We create a `Scanner` object connected to the keyboard

  ```
  Scanner scan = new Scanner(System.in);
  ```

- Operator `new` $\Rightarrow$ a new object is created
- The object `System.in` represents the keyboard
  (Remember: `System.out` represents the screen.)
- The method call `scan.nextLine()` ...
  - reads a line of text or ...
  - stops execution until one line is available.
- The methods `nextInt` and `nextDouble` can read other types of data.
- The program crashes if wrong type of data is provided.
- Done reading $\Rightarrow$ `sc.close()` (to avoid Eclipse warnings).

The `Scanner` class can do much more than we show here. For example, it can be connected to a file instead of the keyboard.

ScannerIntro.java

```
Scanner scan = new Scanner(System.in);

System.out.print("Type a float: ");
double d = scan.nextDouble();
System.out.println("Echo: "+d);

System.out.print("\nType three integers: ");
int sum = 0;
sum = sum + scan.nextInt();
sum = sum + scan.nextInt();
sum = sum + scan.nextInt();
System.out.println("Sum: "+sum);
scan.close();
```

```
Print-out:
    Type a float: 3.14159268
    Echo: 3.14159268

    Type three integers: 11 22 33
    Sum: 66
```

PickDigit.java

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

    /* Read 3-digit number */
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a 3-digit number: ");
    int n = sc.nextInt();                          // Example: n = 376

    /* Find 2nd digit */
    n = n / 10;                                    // Remove last digit ==> n = 37
    n = n % 10;                                    // Pick last digit   ==> n = 7

    System.out.println("The 2nd digit is: "+n);
    sc.close();
}
```

Print-out:

```
Enter a 3-digit number: 456
The 2nd digit is: 5
```

If time permits ... (1)

Exercise

Write a program `BMI.java` which computes the BMI (Body Mass Index) for a person. The program will read length and weight from the keyboard and then present the result as output. The BMI is computed as $weight/(length)^2$, where the length is given in meters and the weight in kilograms. An example of an execution:

```
Give your length in meters: 1,83
Give your weight in kilograms: 83

Your BMI is: 25
```

Notice

- BMI is always an integer.
- Hence, weight is an `int`, length is a `double`, BMI is an `int`
- How to round of a double to an integer?
- Also, I use `1,83` rather than `1.83` since my computer talks Swedish!

If time permits ...

Exercise

Create a program `Initiales.java` that reads a first name and a surname and prints the corresponding initiales. An execution might look like this:

```
First  Name: Jonas
Surname: Lundberg

Initiales : J.L.
```

Notice

- We use String method `charAt(int pos)` to pick a character from a string

Until Next Lecture

1. Read Chapter 2 in the book by Liang
2. Glance through my slides (as a check for understanding)
3. Start working on the exercises in Assignment 1
4. Show up at Practical Meeting to sort out problems/questions