Grundlagen der Programmierung

Session II - Basics of Java Programming (1/2)

Romain PELISSE - Red Hat Gmbh

Humboldt Universität, Berlin

2012

Agenda

Session outline

- functions
- data conversion
- data structure (classe)
- error handling
- objects and methods

No session on the 15.06

No class on the 15.06

- postpone to 22.06 ?
- postpone to 06.07 ?

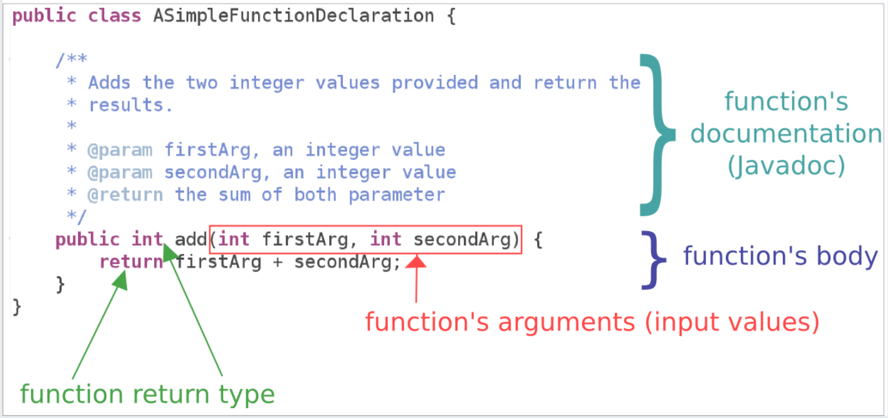
What is function ?

In computer science, a subroutine, also termed procedure, function, routine, method, or subprogram, is a part of source code within a larger computer program that performs a specific task and is relatively independent of the remaining code. - Wikipedia "Subroutine", accessed the 22.05.2012

What are their purpose ?

- reuse code easily
- make code easier to read
- breakdown code in different, meaningful, part
- hide complexity

Functions syntax



```
public class ASimpleFunctionDeclaration {

    /**
     * Adds the two integer values provided and return the
     * results.
     *
     * @param firstArg, an integer value
     * @param secondArg, an integer value
     * @return the sum of both parameter
     */
    public int add(int firstArg, int secondArg) {
        return firstArg + secondArg;
    }
}
```

function's documentation (Javadoc)
function's body
function's arguments (input values)
function return type

Exercice 0 : Create a simple function to add two integer

- Create a Java file in your project called FunctionByExample with a right-click on the folder containing the Java source files
- *Remark : Check the option "public static void main..." in the form*
- Add to the class the following function
  - `public static int add(int a, int b)`
  - implements the body of the function
- Remark : the function must be **within** the class definition
- in the **main** function add a call to the add function
  - `FunctionByExample.add(1,2)`

Exercice 1 : Reduce code duplication in using functions

- Import your Eclipse project the file called LenghtyProgram
- Follow the instructions and modify the code accordingly

Data conversion

- variables holds **typed** data
- need to convert one to an other :
  - an integer (3) to a float (3.0)
  - a char (c) to the
  - a String ("1.0") to a float (1.0)
- in Java, this can be achieved by some provided functions :
  - `int value = Integer.valueOf("1.0");`
  - `short value = Short.valueOf(1.0f);`
  - `float value = Float.valueOf(1);`

Constants

- some variable never changes
- can be prefixed by keyword `final`
- to prevent any - unwanted, changes later in the program
- convention is, in most languages, to use uppercase variable :
    - `final long EARTH_DIAMETER = 12,714;`
    - `final float PIE = 3. 1415926535;`

Exercice 2 : Increase code structuration using functions

- Import your Eclipse project the file called TiedlyCoupledBusinessCode
- Follow the instructions and modify the code accordingly

What to modelize complex data ?

- concept of **data structure**
- define a new **type** of variable
- that regroups all variables
- in Java, a data structure is called **class**

```
public class Steuerzahler {

  double id;
  short taxClass;
  long lastYearRevenue;
  // ...
}
```

How to use a data structure ?

- inner variables can be accessed directly
  - `steuerzahler.id = 120304;`
- structure can be passed around di
  - `public static void calculateTax(Steuerzahler steuerzahler) ...`

Designing a data structure

- Create a new **class** :
  - Right click on the folder containing your source code (inside Eclipse)
  - Select New...->Java Class
  - Name the new class Consultant
- a Consultant's data structure should regroup the following information :
  - an unique id value
  - years of experience
  - a country code (where he works) identified by two letter (DE, UK, FR...)
  - cost by day ratio (how much the consultant is sold by day)
  - a phone number
- implements the data structure
- add the following function to the class and implement it :
  - `public static int consultantCostFor(Consultant consultant, int nbDays);`

Structure to handle error in programming language

- do nothing
  - program crashes
  - no information on the root cause
- use "status code"
  - functions can't return value
  - leads to message such as "Error 400 happened"
  - needs to have an error database to translate the status code
- return a complete structure describing in length the error
  - ideal, but...
  - ...still remove the option of having returning value
- hence appeared the idea of **exception**
  - returns a complete structure describing the error
  - does not modify the return type of a function
  - can be explicitly catched or not
  - can be explicitly thrown or not

```java
public class ExceptionInJava {
  public static void functionsWithThrow()
      throws IllegalArgumentException {
      // explicitly
      throw new IllegalArgumentException("Error...");
  }

  public static void functionsWithSilentThrow() {
      //...
      throw new IllegalArgumentException("Error...");
  }

  public static void catchingException() {

      try {
          // code that may throw exception
          ExceptionInJava.functionsWithThrow();
      } catch ( IllegalArgumentException e ) {
          e.printStackTrace();
      }
      ExceptionInJava.functionsWithSilentThrow();
  }
}
```