Computer Science for Engineers
Exercise 2

Introduction to Java

Prof. Dr. Dr.-Ing. J. Ovtcharova
Dipl. Wi.-Ing. Dan Gutu
21st of November 2008
Basic principle of object orientation

- **Mapping of parts of the real world into a program:**
  - The real problem is illustrated through a program
  - The illustrated elements of the program are related to each other and can interact with each other
• An **object** is a theoretic or real entity of the environment or in the software. Objects are characterized by [Schn98]:
  - *Identity*: specification, resp. instance of a class
  - *Data aspect*: Attributes, resp. properties
  - *Activity aspect*: behavior, given by the methods of the object

• A class is a **entity**, by which the **properties** and the **behavior** of the objects is described. Classes can be in a **class hierarchy**. [Schn98]

• The **attribute** is a property of an entity provided with a name. Usually, its meaning is given by its name [Schn98].
• A **method** is a program fragment, where a received message is processed and the inner state of the addressee is modified, accordingly. Methods can be compared with the procedures of the procedural programming languages and are assigned to an object. [Schn98]

• A **message** is an entity, from which all **interactions between objects** are derived. It is composed of the **specification of the addressee** (an object), a **method** that can be processed by the addressee, as well as the required **parameters** [Schn98].

• **Inheritance:** propagation of properties (attributes, methods) to another class or to another object [Schn98].

• A **class library** is a collection of reusable software components that use the constructs of the object oriented programming paradigm. [Schn98]
Object orientation example: Person

- Which are the properties of a person?
  - Hair colour
  - Height
  - Weight
  - Name
  - Age

- What can a person do?
  - Sleep
  - Eat
  - drink
  - createChildren
  - speak
Class in Java: Person

- Which are the properties of a person?
  - Name
  - Hair colour
  - Height
  - Adresse
- What can a person do?
  - Wave hand
  - speak
    - Say his name
• In a class are defined:
  - Attributes (properties)
  - Relations to other classes (by attributes that are objects of other classes)
  - Methods (behavior of the objects of one class)

  public class Person {
    // String for string of characters
    private String name;
    // int for integer number
    private int age;
    public String hair colour;
    // Method
    public String getName() {
      return name;
    }
  }
• By encapsulation, users of an object have access only to those methods and data elements that are relevant for them. The rest is hidden.

• The object encapsulates its data and allows access only over predefined interfaces (the methods).

• The attributes of an object can be accessed only by the methods (set the attributes value, or read it).

• This behavior is called **encapsulation** or **information hiding**.
Encapsulation in Java

• Visibleness and accessibility on the classes, attributes and methods is limited
  - public: visible from outside. Meaning, that the method/ attribute is visible and accessible from outside the class or the object
  - private: visible/accessible inside the class or the object

  - Example (see code):
    - Hair colour is public, it can be accessed from outside.
    - Name is private, not visible from outside. – Access methods must be written (ex., the method getName)
Classes in JAVA

- Classes are implemented completely in the source code file.
- Usually, a file contains only one class.
- Declaration of a class:

  ```java
  public class Classname {
  }
  ```

- Last row

  ```java
  }
  ```

- Inside this block, variables are declared and methods implemented. In the methods, assignment blocks are defined.
Variables:

- Variable = storage position where a value can be stored
- Variables have a data type, a name, and a value
- Example: `int anIntegerVariable boolean aLogicalVariable`

In JAVA: 3 different kinds of variables:

- **Local variables** (have local validity, e.g. in blocks, methods)
- **Instance variables** (belong to an instance ( = Object))
- **Class variables** (belong to a class)
• **Declaration** of a variable: definition of a variable’s data type and name

• Memory is allocated with the declaration of primitive data types

• **Each** variable has to be declared before its first use. Otherwise: compilation error.

• Syntax of a declaration:

```
Data type comma-separated list of variables’ names;
```

• **Examples:**

```c
int integerNumber1, integerNumber2;
char aCharacter, anotherCharacter;
boolean anAnswer, anAnswer2;
```
• Assignment of a value to an already declared variable

• Syntax of an assignment:

```plaintext
Variable name = term ;
```

• Term:

  • has to be of the same data type as the variable (left of “=”). Otherwise: compilation error.

  • Can be a complex instruction comprising other variables and operators

• Examples:

```plaintext
integerNumber1 = 16;
integerNumber2 = (integerNumber1+5) * 13;
anAnswer1 = false;
anAnswer2 = (!anAnswer1) | false;
```
Initialisation = **first** assignment of a value to an already defined variable

Initialisation has to be performed for each variable. Otherwise, compilation errors occur when the variable is used.

Syntax of an initialisation: like assignment.

Thus, frequently, declaration **and** initialisation are performed in **one** statement with the following syntax:

```
Data type Variable name = term ;
```

Attention: here, two very different things are happening within the same statement.
• Constant  = Variable with a fixed, invariable value
• Declaration + singular assignment in one statement
• Further assignments cause compilation errors
• Key word: final

Example:
```
final double DM_PER_EURO = 1.99583;
final char BIGA = 'A';
```
• There are 3 kinds of variables:

```java
public class Circle {
    private double x, y, z; // Coordinates of the center
    private double r; // Radius
    private static final double PI = 3.14159;

    // Method returns the area of the circle
    public double getArea() {
        double rSquare;
        rSquare = r * r;
        return rSquare * PI;
    }
}
```

Definition of Variables: *(visibility) Type Name*;
• Instance variables describe the properties of an object
• Each object of a class has the same properties, but with individual values
• Syntax of the declaration of instance variables

```
Access modifier  Data type  Name(s) of instance variable(s) ;
```

• Possible access modifiers in JAVA:
  - `private`: access only from methods of the same class
  - `protected`: (explained later on)
  - `nothing`: (explained later on)
  - `public`: access without restrictions
Access to instance variables is allowed or denied by access modifiers.

Syntax for the access to instance variables of a local declared object:

- Access to an object's instance variable from within the same instance:
  
  ```
  objectname . Name of instance variable
  ```

- Access to an object's instance variable from within the same instance:
  
  ```
  this.Name of instance variable , or
  ```

  ```
  Name of instance variable
  ```
Class Variables

• Class variables are variables that are valid for all instances of a class.

• Class variables are similar to global variables in other programming languages.

• The keyword "static" is used to declare a variable as class variable and not as instance variable.

• Class variables are also called static variables.

• There is only one copy of the variable which belongs to the class, instead of many copies, each of which belonging to each instance of the class.
• Class variables describe common properties of all objects of a class (attribute name + value)

• Every object can have class variables but there is only one version of the variable per class

• Syntax of the declaration of class variables

```
access modifier static data type Name(s) of class variable(s) ;
```

• Same access modifier like with instance variables possible

• Example:
  ```
  private String givenName, Surname;
  private static int bankCode;
  ```
• access to class variables is allowed or denied by access modifiers.

• Syntax for the access to a class variable

\[
\text{classname} \ . \ \text{Name of class variable}
\]

• Access to class variable from within the same class:

\[
\text{Name of class variable}
\]
public class Circle {
    // Class variable: how many circles were created
    static int numCircles = 0;
    // Instance variable: The center and radius
    public double x, y, r;
    // Constructors
    public Circle( double x, double y, double r ) {
        this.x = x; this.y = y; this.r = r;
        numCircles++;
    }
    public Circle() {
        x = 0.0; y = 0.0; r = 1.0;
        numCircles++;
    }
    // Methods
    public double circumference() { return 2 * 3.14159 * r; }
    public double area() { return 3.14159 * r * r; }
}

System.out.println("Number of circles created: "+Circle.numCircles);
public class Circle {
    static final double PI = 3.14159265358979323846;

    // Instance variables
    public double x, y, r;

    // Constructor
    public Circle() { x = 0.0; y = 0.0; r = 1.0; }

    // Methods
    public double circumference() { return 2 * PI * r; }
    public double area() { return PI * r * r; }
}

Methods

• There are 2 kinds of methods:
  - Object methods
  - Class methods
    ▪ Particular case *main method*

• Method definition:

  Access modifier (Visibility) return type Name (Parameter type1 Parameter name1, ...) {
    instruction1;
    instruction2;
    return aValue;
  }

  public double umfang() {
    return 2 * 3.14159 * r;
  }
  public void setRadius (double anotherR) {
    r = anotherR;
  }
  public static boolean isRound {
    return true;
  }
• Instance methods provide access to an object’s instance variables and implement the possible actions (operations) of an object.

• Declaration of an instance method:

```
access modifier  data type of return value  Method name

(  Parameterlist  )  {

}  

• End of an instance method:

```
```

• Parameter list:

```
data type  variable name  ,  ...  (or empty parameter list)
```

examples:

```java
public void showAccountBalance() {
    // here method body with statements
}
public boolean deposit() {
    // here method body with statements
}
public void payout(double amount) {
    // here method body with statements
}
public double calculationMethod(double error, int actualIteration) {
    // here method body with statements
}
```
public String aMethod (int number1, int number2) {
    String output;
    if (number1 > number2) {
        output = "the first number is bigger";
    } else {
        output = "the second number is bigger";
    }
    return output;
}
Calling an instance method: only possible for a certain object

Syntax of method call:

- **Object name**
  
- **Method name**
  
- **Parameter list**

Syntax of method call from within the same instance:

- **This**.
  
- **Method name**
  
- **Parameter list**

, or

- **Method name**
  
- **Parameter list**

, or
• Class methods are methods that are valid for all instances of a class.

• The keyword "static" is used to declare a method as class method and not as instance method.

• Class methods are also called static methods.

• Class methods have no implicit this-reference.

• Class methods do belong to no instance of the class.

• Class methods can access no instance variables or call instance methods.
• Class methods implement actions independently from any objects of the class. Consequence: **no access to instance variables**

• Declaration of a class method:

  ```
  access modifier static data type of return value void Method name (Parameter (list)) { } 
  ```

• End of a class method:

  ```
  } 
  ```

• Parameter list

  ```
  data type variable name , ... (or empty parameter list) 
  ```
• Calling class methods only with class reference
• Syntax for calling class methods

\[
\text{classname} \ . \ \text{methodname} \ (\text{Parameter} \ (\text{list})) \ ;
\]

• Syntax for calling methods within other class methods of the same class

\[
\text{methodname} \ (\text{Parameter} \ (\text{list})) \ ;
\]
At method declaration: return type is not `void`, but an existing data type (class possible, too)

In order to allow for returning results: return statement as final statement of the method

Syntax:

```
return term ;
```

„term“ must be of the same data type that was declared in the method‘s declaration

Example:

```java
public double square(double aNumber) {
    double numberSquared = aNumber*aNumber;
    return numberSquared;
}
```
• In one class, only one `main`-method can be declared and implemented

• When a class is executed with `java`, only its `main`-method is executed

• Head of the `main` method within a class

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

• Within the `main` method: sequence of statements

• Last row of the `main` method:

```
}
```
public class Circle {
    //... class definition

    public static void main(String[] args) {
        // declaration of a variable for objects of the class "Circle":
        Circle a;

        // creation of a new Circle-object:
        a = new Circle();
        // the variable "a" references to a new object

        // call of the method "circumference" of the object "a":
        double cf = a.circumference();

        // a lot of things to do........

        // program end
    }
}
Differences between Classic/Object-Oriented Programming

Classic
- Calls
- Functions
- Data
- Access
- Relations

OO
- Message exchange and launch of actions
- Objects
- Inherit specifications (variables, methods) and relations
### Collection of common attributes of objects from one class

<table>
<thead>
<tr>
<th>Name</th>
<th>Attribute</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle</td>
<td>x-position: double</td>
<td>circumference(): double</td>
</tr>
<tr>
<td></td>
<td>y-position: double</td>
<td>area(): double</td>
</tr>
<tr>
<td></td>
<td>radius: double</td>
<td></td>
</tr>
</tbody>
</table>
Coherence between Objects and Classes

Classes:
A class represents a pattern for the creation of objects of the same type. Methods and variables existing in a class, will be transmitted to new objects (instances of a class).

Objects:
An object contains encapsulated data and methods to manipulate them. The data specify the state of an object, the methods specify its behavior - the reactions to external influences.
• A class corresponds to the data type of a variable.
  - `String charstring` is a variable of the (class)-type `String`.

• In a class are defined:
  - Attributes (properties)
  - Relations to other classes (by attributes that are objects of other classes)
  - Methods defined (behavior of the objects of one class)

```java
public class Person {
    private String name;
    private int age;
    public String haircolour;
}
```

- The variables can have objects of their data type as values
The class as datatype

- A class corresponds to the data type of a variable.
  - Attributes can have as data type a class.
- Ex: A **Person** has the attribute of type **Adress**
  - Person contains a variable that is an object of the class address.

```java
public class Person {
    private Adress adress;
    private int age;
    // …
}
```

```java
public class Adress {
    private String street;
    private int housenumber;
    // …
}
```

The classes are linked together:
An object of the class Adress belongs to an object of the class Person
• The relations between classes are represented in class diagrams

• A Person instance has an attribute of type Adresse
  - This relation is directed: a Person has an Adresse, but the Adresse does not know to which Person it is assigned (the class Address has no attribute of type Person)
  - A Person has only one Address.