Objec Oriented Design

Master 1 of Informatics I.E.I. - V.N.U.

2020-2021

Marie Beurton-Aimar

Université de Bordeaux

Introduction about Software Designing

- Object Oriented design and programming.
- Large programs for real cases.
- Maintenance is time and money consuming.
- Creating new programs is not the common task now.

Introduction about Software Designing

- Object Oriented design and programming.
- Large programs for real cases.
- Maintenance is time and money consuming.
- Creating new programs is not the common task now.
- Several ways to answer : conception level, programming level, data level ...

Data Analysis

- Real applications often imply complex data.
- Object Oriented paradigm is based on the analysis of data firstly.
- Does it exist some methods to help us to analyze these data?

Data Analysis

- Real applications often imply complex data.
- Object Oriented paradigm is based on the analysis of data firstly.
- Does it exist some methods to help us to analyze these data?
- It exists **methodology** and many **recipes**.

Find the good classes

- De-composition / Composition: an object is an individual element, real or abstract with a well defined role in the problem domain.
- Find the right objects and the size of each object.
- Decide where to attach behaviors.
- Create modules and links between them.

Basic Principles

- **Objects** : base units organized in classes and sharing common traits (attributes or procedures). They can be entities from real world, concepts of the application or of the concerned domain.
- **Encapsulation** :
  - Data structures and implementation details are hidden to the other system objects.
  - The only way to access to the object state is to send to it a message which activates the execution of one of its methods.
- **Abstraction** : is the way to apply encapsulation and encapsulation make abstraction useful.

Basic Principles

- **Polymorphism** : possibility to use the same expression for different operations. Inheritance is a specific form of polymorphism typical to the object oriented systems.
- **Modularity** : program partition which creates well-defined (and documented) frontiers into the program in the goal to reduce complexity(Meyers). The choice of a good set of modules for a given problem is as difficult as the choice of a good set of abstractions.

Object Programming

Object Oriented Languages

- The old ones : C++, SmallTalk, C
- A lot of another ones : Java, Python, Ruby, Ada, ObjectiveC, PHP ...
- In fact at this time, pretty all languages have a object layer.

Usage

- Graphic application : Swing, JFX, GTK ..
- Web : JavaScript, Nodejs
- Statistics : R, python library

Java: A coffee story

- Written by Sun team in the 90's.
- Develop by the world community in collaboration.
- Address all the application domain :
  - Economy, bank applications,
  - Graphical interfaces,
  - Web applications with Applet (no link with JavaScript).
  - DataBase programming,
  - Network and systems application

Java - Environment

Characteristics

- Byte Code
- Virtual Machine
- Tools:
  - Java SE - JDK plateform.
  - Current version 15.0.1 (11 is also possible).
  - Running environnement
  - Compiler to produce byte code.

Java Language

- Everything is an object.
- A program is a set of objects which send messages together.
- Each object as a type (instance of class). All objects of the same type can receive the same message.
- A class describes a set of objects which share common features (data) and behaviors (methods).
- Inheritance between classes allows to organize data type in hierarchy.
- Each instance of a class inherits of the features (attributes and methods) of its own class and of the super-class.

Writting Rules

- Class names have to begin by an upper case letter.
- Method and variable names have to begin by a lower case letter.
- Everything, all pieces of code, has to be in a class.
- A class has to contain the `main` method.

```
public static void main(String []args) {
.....your code ....
}
```

Primitive Types

Types	Characteristiques
boolean	`true` ou `false`
char	charactere 16 bits Unicode
byte	integer 8 bits signés
short	integer 16 bits signés
int	integer 32 bits signés
long	integer 64 bits signés
float	real number with floating point 32-bits
double	real number with floating point 64-bits

Operators

- **Arithmetics** :
  - $+$, $-$, $*$, $/$, %,
  - $+=$, $-=$, $*=$, $/=$, $++$, $--$,
- **Boolean** :
  - $==$, $!=$, $=<$,$=>$, $||$, &&, ? :.
- **Control structures** :
  - `if`, `for` , `while`, `switch`,

"wrapper" Classes

- Primitive types can be encapsulated in another classes :
  - Integer, Byte, Long,
  - Double, Float,
  - Character, Void.
- Example :

```
int num=Integer.parseInt(word);
double size=Double.parserDouble(chain);
```

String class

- Create a String : `String word = "abc";`

  ```
  char data[] = {'a', 'b', 'c'};
  String newWord = new String(data);
  ```

- Test equality : `word.equals(``bcd'');`

  Take care with ==.

- Convert an `int` or a `double` to a `String`

  `String name=String.valueOf(num)`

- Get information about a String : `length()`, `charAt()` ...

Create an array

- Array type : `[]`
- Declare an array :
  - `int []myArray;`
- Declaration and memory allocation :
  - `int []myArray=new int [10];`
  - `Person []anotherArray = new Person[MAX];`
- Access to the box array : `int num=myArray[2];`
- Size of the array : `int size =myArray.length`

Un petit program !

```
class Hello{
      static public void main(String []args){
        System.out.println(``Hello World'');
      }
}
```

Define a class

```
public class Person{
  private String name;
  private int age;
  public void display()
  {
   System.out.println("this person is " + name);
   System.out.println
                    ("she is " + age + "year old");
  }
}
```

Define a constructor

```
public class Person{
  private String name;
  private int age;
  public Person(String chain, int val)
  {
    name=chain;
    age=val;
  }
  public void display()
  {
   System.out.println("this person is " + name);
   System.out.println
                    ("she is " + age + "year old");
  }
}
```

Take care of the default constructor `Person()`

Overloading the constructor

```
public class Person{
  private String name;
  private int age;

  public Person(String chain, int val)
  {
    name=chain;
    age=val;
  }
  public Person(String chain)
  {
    name=chain;
    age=0;
  }
  public void display()
  {
   System.out.println("this person is " + name);
   System.out.println
                  ("she is " + age + "year old");
```

Using This

- Design the instance attribute:

```
public Person(String name,int age){
    this.name=name;
    this.age=age;
 }
```

Using a class

- Create a variable of the type `Person`

```
public void main (String []arg){
    Person item;
    Person item2=new Person(``Kee'');
    Person item3=new Person(``Lila'',2);
```

Define methods

- Using instance attributes/variables :

```
public String getName(){
     return (name);
 }
public int getAge(){
     return(age);
 }
public void setAge(int val){
     age=val;
 }
```

Calling methods

- Call a method linked to an instance :

```
Person item2=new Person(``Hien'');
item2.setAge(2);
String name=item2.getName();
item2.display();
```