Define Your Own Classes

HAN Xicun

xicun.han@gmail.com

3 Mars 2017

Table des matières

List of source codes

1 An Employee Class

The form for a class definition in Java is :

```java
class ClassName{
        field1;
        field2;
        ...

        constructor1
        constructor2
        ...

        method1
        method2
        ...
}
```

Listing 1: Example of Class Structure

Details plus look at the example in File : *Employee.java*

2 Use of Multiple Source Files

Importance : the order

1. looking for *.class
2. if can not find *.class, it will find *.java and compile it.
3. there is also a step of comparison with timestamp. if *.java has a timestamp newer than *.class, it will also compile it before using.

3 Dissecting the Employee Class

Dissecting : cut up and analyse.

— **Filed** : private
— **Method** : public

4 First Steps with Constructors

* A constructor has the same name as the class.
* A class can have more than one constructor.
* A constructor can take zero, one or more parameters.
* A constructor has no return value
* A constructor is always called with the new operator.

5 Implicit and Explicit Parameters

- **Implicit Parameters** : using *this* to refer to.
- Explicit Parameters : In the parentheses after the method name.

6 Benefits of Encapsulation

- *private data field*
- *public field accessors* : get method
- *public field mutator method* : set method

Attention to MUTABLE OBJECT : : if we need to return a reference to a mutable object, it is better to *Clone* it first.

```
Date d = harry.getHireDay();
double tenYearsInMilliSeconds = 10 * 365.25 * 24 * 60 * 60 * 1000;
d.setTime(d.getTime() - (long) tenYearsInMilliSeconds);
// let's give Harry ten years of added seniority
//In order to avoid this:

public Date getHireDay() {
return (Date) hireDay.clone(); // Ok
}
```

Listing 2: Do not return Object Directly

7 Class-based Access Privileges

```
class Employee {
...
public boolean equals(Employee other) {
        return name.equals(other.name);
}
}
// A typical call:
if (harry.equals(boss)) . . .
```

Listing 3: Privilege Access

This method accesses the private fields of harry, which is not surprising. It also accesses the private fields of boss. This is legal because boss is an object of type Employee, and a method of the Employee class is permitted to access the private fields of any object of type Employee.

8 Private Methods

Typically, these helper methods should not be part of the public interface.

9 Final Instance Fields

define an instance field as final : a field must be initialized when the object is constructed.

The final modifier is particularly useful for fields whose type is primitive or an immutable class.

immutable class : if none of its methods ever mutate its objects. For example String.

For mutable classes, the final modifier can be confusing. Consider an example :

```
private final StringBuilder evaluations;
```

that is initialized in the Employee constructor as :

```
evaluations = new StringBuilder();
```

The final keyword merely means that the object reference stored in the evaluations variable will **never again refer to a different StringBuilder object**. But the object can be mutated

```
public void giveGoldStar() {evaluations.append(LocalDate.now() + ": Gold star!\n");}
```