OOP - Encapsulation

Chunting Gu, 2011/05/04

An example from Annotation library

```
class Node {
public:
    typedef std::map<std::string, std::string> attributes_t;
    std::string getAttribute(const std::string& key) const;
    void setAttribute(const std::string& key, const std::string&
value);
    const attributes_t& getAttributes() const; // ???
private:
    attributes_t attributes;
};
```

Method ‘getAttributes’ is very generous (慷慨) because it:

1. returns a (const) reference to the member variable;
2. exposes the details of the implementation (i.e., std::map).

To avoid returning reference

```
attributes_t getAttributes() const;
```

Return a copy, less efficient.

To avoid exposing implementation details

We find the only place to call ‘getAttributes’ is to serialize all attributes:

```
foreach(..., node->getAttributes()) {
    // Create a XML element for this attribute.
}
```

So, what about to provide a method like this?

```
struct AttributeHandler {
    void operator()(const string& k, const string& v) {
        visit(k, v);
    }
private:
    virtual void visit(const string& k, const string& v) = 0;
};
void foreachAttribute(AttributeHandler& ah);
```

`And why not public member variable?`

```
class Node {
public:
    attributes_t attributes;
};
```

So that we don't have to provide any methods for attributes.

Don’t do it!

Hybrids (混血, 杂交) are the worst of both worlds

Hybrid: Half object and half data structure.

> “Public variables” tempts other external functions to use those variables the way a procedural program would use a data structure.

Hybrids make it hard to add new functions but also make it hard to add new data structures.

- It’s hard to add new data structures to procedural code because all the functions must change.
- It’s hard to add new functions to OO code because all the classes must change.

Back to the beginning,

`Why shall we keep variables private?`

> We don’t want anyone else to depend on them.
>
> We want to keep the freedom to change their type or implementation on a whim or an impulse.

Why, then, do so many programmers automatically add getters and setters to their objects, exposing their private variables as if they were public?

A Case About Encapsulation

> 取自《冒号课堂》，见 References

```
class Person {
    private Date birthday;
    private boolean sex;
    private Person[] children;
}
```

Get birthday

```
public Date getBirthday() { // ???
    return birthday;
}
```

Method ‘getBirthday’ is dangerous because it returns the *reference* to a *mutable* member variable.

It’s the same as the ‘getAttributes’ as we saw before:

```
attributes_t& getAttributes();
```

Make ‘getBirthday’ return a clone of birthday:

```
public Date getBirthday() {
    return birthday == null ? null : new Date(birthday.getTime());
}
```

Set birthday

```
public Person(Date birthday, boolean sex) { // ???
        this.birthday = birthday;
        this.sex = sex;
    }

public void setBirthday(Date birthday) { // ???
    this.birthday = birthday;
}
```

Get children

```
public Person[] getChildren() {
    return children;
}
```

Array is also mutable. So, clone the children?

```
public Person[] getChildren() {
    if (children == null || children.length == 0) { return null; }
    Person[] childrenCopy = new Person[children.length];
    System.arraycopy(children, 0, childrenCopy, 0, children.length);
    return childrenCopy;
}
```

Get children (cont.)

Or replace ‘getChildren’ with:

```
getChild(int index);
getChildCount();
getFirstChild();
getLastChild();
findChildByXXX();
...
```

Set children

Method ‘setChildren’ is too generous (太慷慨了):

```
public void setChildren(Person[] children) {
    this.children = children;
}
```

Consider to replace with:

```
addChild(Person child);
removeChild(Person child);
clearChild();
```

Get sex

```
public boolean getSex() {
    return sex;
}
```

- Change *boolean* to *Boolean* so that we have an extra *null* value for sex? Or
- Use *int*, *char*, *string*?
- Use enumerated type?
- ...

Get sex (cont.)

What about this?

```
public boolean isMale();
public boolean isFemale();
```

Set sex

```
public void setSex(<T> sex) {
    this.sex = sex;
}
```

Get age

```java
public int computeAge() { // ???
    // compute age by birthday.
    // ...
}
```

The ‘compute’ in the name exposes the implementation details.

Consider to change to:

```java
public int getAge() {
    // ...
}
```

References

Robert C. Matin, Clean Code - A Handbook of Agile Software Craftsmanship.

郑晖, 《冒号课堂-编程范式与 OOP 思想》
http://book.douban.com/subject/4031906/

Wm. Paul Rogers, Encapsulation is not information hiding.
http://www.javaworld.com/javaworld/jw-05-2001/jw-0518-encapsulation.html