Cours de C++

Programmation orientée objets

Cécile Braunstein
cecile.braunstein@lip6.fr

Programmation oriented object (POO)

Advantages

- Re-use
- Modularity
- Maintainability

Language oriented object

Before :

- Data more or less well organised
- Functions and computation applied on these data
- A program is a following of affectation and computation

POO :

- Modules (*classes*) representing data and functions
- A program is a set of *objects* interacting by calling their own functions(*methods*),

Concepts

Objects

An object is a recognizable element characterized by its structure (*attributes*) and its behavior (*methods*)

➛ Object = Class instance

Class

Groups and creates objects with the same properties (method and attributes).

Class members :

- Attributes : define the domain of value
- Methods : define behavior ; set of function modifying the state of an object

A class has got at least one attribute and two methods (create and delete)

Information hiding

Purpose

Restrict access to a class by its interface

- Put constraints for the use and the interaction between objects.
- Programmer see only a part of the object corresponding to its behavior
- Help updates and changes for a class.

Class has two parts

- An interface : access for external users,
- Internal data and internal implementation.

Inheritance

Models the dependency between classes

- Allows re-use of class property by specialization
- Programming by incremental refinement

B derives from A

B has got at least all A's members.

- All B object are also A object,

```
A x;
B y;
x = y; // ok, y is of type B so of type A
y = x; // ko x is not of type B
```

- All A's members are members of members of B without declaration or implementation
- B may add new functionality, it's a specialization of A

Defining new types in C++

Create our types

```
struct Student_info{
  std::string name;
  double partiel, final;
  std::vector<double>
      homework;
};
```

```
struct Student_info{
  std::string name;
  double partiel, final;
  std::vector<double>
      homework;

  std::istream& read(std::
      istream&);
  double grade() const;
};
```

Usually written in a header file.

Create interface

Our Goal :

- Hiding implementation details
- Users can access only through functions

Member functions

read

```
istream& Strudent_info::read(istream& in)
{
   in >> name >> partiel >>final;
   read_hw(in, homework);
}
```

Particularities

- The name of the function `Strudent_info::read`
- No object `Strudent_info` in parameters list
- Direct access to data elements of our object

Member functions

grade

```
double Student_info::grade() const
{
  return ::grade(partiel,final,homework);
}
```

What's new ?

- `grade` is a member of `Student_info` : implicit reference to the object
- `::grade` : insists that we use a function that is not a member of anything
- and `const` ?

Const member function

```
double Student_info::grade() const {...} //new
double grade(const Student_info&) {...} //old
```

Const

- In the old version we ensure that the grade function do not change the parameter
- In the new version, the function is qualified as `const`
- `grade` can be applied to a `const` or no`const` object
- `read` cannot be call by a `const` object

Protection

```
class Student_info{
public:
  //interface
  double grade() const;
  std::istream& read(std::istream&);

private:
  //implementation
std::string name;
double partiel,final;
std::vector<double> homework;
};
```

Protection label

Each protection label defines the accessibility of all members that follow the label.

labels

They can appear in any order

- `private` : Inaccessible members
- `public` : accessible members

`struct` or `class` ?

There is no difference except :

- default protection : private for a class ; public for struct.
- by convention : struct for simple data structure

Constructor

Definition

- Special member functions that defines how object are initialized.
- If no constructor are defined the compiler will synthesized one for us.
- They have the same name as the name of the class itself
- They have no return type

```
class Student_info{
Student_info(); //construct an empty object
Student_info(std::istream&); // construct by reading a
    stream as before
};
```

The default constructor

The one without argument.

```
Student_info::Student_info():partiel(0),final(0) {}
```

Constructor initializer

When we create a new class object :

1. The implementation allocate memory to hold the object
2. It initializes the object using initial values as specified in an initializer list
3. It executes the constructor body

Destructor

```
class Student_info{
 ~Student_info();
};
```

Definition

- Free the allocated memory
- Only one in a class
- Can be synthesized if it doesn't exist