Technical University of Moldova
Inginerical department S.A.

Report

№: 5

OOP

Subject: Multiple inheritance

Author: Terman Emil FAF161
Prof: M. Kulev

Chisinau 2017

1 Objectives

- studierea regulilor de determinare a moştenirii multiple.
- studierea avantajelor şi neajunsurilor moştenirii multiple.
- probleme legate de utilizarea moştenirii multiple.

2 Main notions of theory and used methods

Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes.

The diamond problem occurs when two superclasses of a class have a common base class. For example, in the following diagram, the TA class gets two copies of all attributes of Person class, this causes ambiguities.

3 Task

Să se creeze, o ierarhie de moştenire: om - student, colaborator - practicant

4 Data analysis

- the *Human* class has a *name* string.
- the *Student* and *Collaborator* virtualy inherit from *Humane*, because we have the **diamond** problem and we need to resolve the problem whith the Human constructor.
- both *Student* and *Collaborator* have a *grade* field. Because they both have something with the same name, when *Intern* inherits from both of them, we must also specify the class we are refering to.

5 The actual code

CPP 1: Human.hpp

```cpp
#ifndef HUMAN_HPP
# define HUMAN_HPP

# include <iostream>
# include <string>

class Human
{
public:
    std::string Name() const;

    Human(std::string name);
private:
    std::string _name;
};

#endif
```

CPP 2: Student.hpp

```cpp
#ifndef STUDENT_HPP
# define STUDENT_HPP

# include "Human.hpp"
# include <iostream>
# include <string>

class Student : virtual public Human
{
public:
    std::string University() const;
    float Grade() const;
    void SetGrade(float newGrade);

    Student(std::string name, std::string university);

private:
    std::string _university;
    float _grade;
};

#endif
```

CPP 3: Collaborator.hpp

```cpp
#ifndef COLLABORATOR_HPP
# define COLLABORATOR_HPP

# include <iostream>
# include <string>
# include "Human.hpp"

class Collaborator : public virtual Human
{
public:
    float Grade() const;
    void SetGrade(float newGrade);

    Collaborator(std::string name);

private:
    float _grade;
};

#endif
```

CPP 4: Intern.hpp

```cpp
#ifndef INTER_HPP
# define INTER_HPP

# include <iostream>
# include <string>
# include "Student.hpp"
# include "Collaborator.hpp"

class Intern : public Student, public Collaborator
{
public:
    Intern(std::string name, std::string university);
};

#endif
```

CPP 5: Intern.cpp

```cpp
#include "Intern.hpp"

Intern::Intern(std::string name, std::string university) :
    Human(name),
    Student(name, university),
    Collaborator(name)
{
}
```

CPP 6: main.cpp

```cpp
#include "Human.hpp"
#include "Student.hpp"
#include "Collaborator.hpp"
#include "Intern.hpp"

int main()
{
    Intern intern = Intern("emil", "UTM");

    std::cout << intern.Name() << std::endl;
    std::cout << intern.University() << std::endl;

    intern.Student::SetGrade(10);
    intern.Collaborator::SetGrade(11);

    std::cout << intern.Student::Grade() << std::endl;
    std::cout << intern.Collaborator::Grade() << std::endl;
    return 0;
}
```

6 Analysis of the results and conclusions

- because multiple inheritance makes more problems than it solves, it is eliminated in more advanced languages, so it's not a necessary feature.
- it is useful in some particular cases, but mostly, we can workaround them by using composition.
- despite that, I think, it would be better if modern languages had this feature, because with it, we may find a more elegant solution to our problems.