SFU CMPT-212 2008-1 Topic: Has-a Relationship

Ján Maňuch

E-mail: jmanuch@sfu.ca

Friday 22nd February, 2008

Has-a relationship

- *public inheritence* models is-a relationship
  Example: `Student` is a `Person`
- *Q.* how can we model has-a relationship?
  Example: `Student` has "grades" (`intArray`)
  2 different ways:
  - *containment* — the owner class contains member objects of another class
    Example:

```
1   class Student {
2     char name[20];
3     intArray grades;
4     ...
5   };
```

- *private inheritence*

  Example:

```
1    class Student : private intArray {
2      ...
3    };
```

- In both cases the owner class (`Student`) inherits the *implementation*, but not the *interface* of the contained/base class:
  - class methods can use public interface of the contained/base class;
  - public interface of the contained/base class will not become part of public interface of the owner class.

Recall declaration of class `intArray`:

```
1  class intArray {
2    int *data;
3    long size;
4   public:
5    intArray(long s);
6    intArray(const intArray &); // copy constructor
7    virtual ~intArray();
8    intArray & operator=(const intArray & a);
9    // assignment operator
10   int get(long index) const; // get a value
11   void set(long index, int value); // set a value
12   friend ostream & operator<<(ostream &os,
13                               const intArray &a);
14 };
```

Containment

Example: student1.cpp

```
1   class Student {
2     char name[20];
3     intArray grades;
4    public:
5     Student(const char *n, int ngrades);
6   };
```

- *Q.* how to initialize grades to the array of size `ngrades`?
  - before executing the body of constructor, C++ creates all data members: *for member objects, it calls the default constructor*
  - but `intArray` has no default constructor!
- *A.* similarly as data members of the base class, member objects can be initialized with **non-default** constructors through *member initializer list*

Example:

```
1  Student::Student(const char *n, int ngrades)
2    : grades(ngrades)
3  {
4    strcpy(name,n);
5  }
```

Notes:

- instead of class name we use directly the name of the member object
- any data members can be initialized in the same way:

```
1   class Point {
2     int x,y;
3    public:
4     Point(int xx, int yy) : x(xx), y(yy) {}
5   };
```

- *Q.* how to use interface of class `intArray`?
  - as `grades` is a private data member of class `Student`, it cannot be accessed (as well as its interface) from outside of the class
  - but it can be accessed in the methods of the class, hence class `Student` can make it available to "the world":

```
1  class Student { ...
2    int getGrade(long index) const
3      { return grades.get(index); }
4    void setGrade(long index, int value)
5      { grades.set(index,value); }
6    friend ostream & operator<<(ostream &os,
7                                const Student &s);
8  };
9  ostream & operator<<(ostream &os,const Student &s) {
10   os << "Student name: " << s.name << endl;
11   os << "Grades: " << s.grades <<endl;
12   return os; }
```

Private inheritence

Example: student2.cpp

```
1  class Student : private intArray
2    char name[20];
3   public:
4    Student(const char *n, int ngrades);
5  };
```

Notes:

- `public` and `protected` members of the base class (`intArray`) become `private` members of the derived class (`Student`)
- models the has-a relationship
- very similar to *containment method*, but now the "*subobject*" `grades` has no name!

- *Q.* how to initialize `intArray` to the size `ngrades`?
  - as with public inheritance, inherited subobjects can be initialized using **non-default** constructors through *member initializer list*

  Example:

```
1   Student::Student(const char *n, int ngrades)
2     : intArray(ngrades)
3   {
4     strcpy(name,n);
5   }
```

- now, we use the name the class of the contained "subobject"

- *Q.* how to use interface of class `intArray`?
  - as public/protected members of `intArray` are private members of class `Student`, they cannot be accessed from outside of the class `Student`
  - we have to make them available to "the world":

```
1 class Student { ...
2   int getGrade(long index) const
3     { return get(index); }
4   void setGrade(long index, int value)
5     { set(index,value); }
6 };
```

Notes:

- sometimes it's desirable that the function of the base class which we are *making available publicly* has the same name as in the base class:

```
1 class Student { ...
2   int get(long index) const
3     { return intArray::get(index); }
4 };
```

— we have to use the scope resolution operator `::`, otherwise we end up with infinite recursive call

- there is another way how to make function of the base class "world-wide" available: *changing access using* `using`*-declaration*:

```
1 class Student { ...
2   using intArray::get;
3   using intArray::set;
4 };
```

— note: no parameters!

- *Q.* what about `friend` functions? how to invoke a `friend` function from the base class?
  - we *cannot* use the membership operator ("subobject" has no name)
  - we *cannot* use the scope resolution operator because a `friend` function does not belong to the class!

  A. type cast the `Student` argument to the base class:

```
1 ostream & operator<<(ostream &os,const Student &s) {
2   os << "Student name: " << s.name << endl;
3   os << "Grades: " << (const intArray &) s <<endl;
4   return os;
5 }
```

Remark: "no upcasting"

- a pointer (reference) to *privately* derived class is not automatically converted to a pointer (reference) to the base class

Containment vs. private inheritance

Advantages:

- *containment:*
  - named subobjects are easier to follow
  - can include more subobjects of the same class
  - easier to resolve identical names
- *private inheritance:*
  - can access protected members of the included subobject
  - can redefine virtual functions
- *in general*: use *containment* unless you need any of the two features provided by *private inheritence*

Protected inheritance

- a variation of *private inheritance*, models the *has-a* relationship
- public and protected members of the base class become protected members of the derived class (this will affect only classes derived from the derived class)

Sample questions

- What are the differences between public and private inheritance. What do they have in common?
- What is member initializer list? What can it contain? In which situations it is necessary to use the member initializer list?
- What's the problem with the following code?

```
1  class A {
2  ...
3  };
4  class B : private A {
5  ...
6  };
7  ...
8  B object;
9  A *p = &object;
```

- What are the two methods for modeling *has-a* relationship? What are advantages of each method?