COMP2012H

Access Control:
public, protected, private

Example: print

- Let's add a `print()` method to our U. Admin. classes.

```
class Person { public: void print() const; ... };
class Student : public Person { public: void print() const; ... };

void Person::print() const {
    cout << "--- Person details ---" << endl;
    cout << "Name: " << name << endl << "Addr: " << address << endl
      << "Dept: " << dept << endl;
}

void Student::print() const {
    cout << "--- Student details ---" << endl
      << "Name: " << name << endl << "Addr: " << address << endl
      << "Dept: " << dept << endl << "Enrolled in:" << endl;
    for (int i = 0; i < num_courses; ++i) {
      enrolled[i].print();                    // Assume a print function in the Course class
     }
}
```

Example: Doesn't Compile!

- The implementation of `Student::print()` given before doesn't work. It will cause a compilation error.

- `Student::print` cannot access `Student::name`, `Student::address`, or `Student::dept`.

  - Since `name` is a private data member of the base class, the derived class cannot access it.

  - Public inheritance does not change the access control of the data members of the base class: private members are still only available to its own methods, and not to any other classes including derived classes (except friends).

One Solution: Protected Data Members

```
class Person
{
    protected:
        string name;
        string address;
        Department dept;
    public:
        void print() const;
        ...
};
```

- By making `name`, `address`, `dept` protected, they are accessible to methods in the base class as well as methods in the derived classes.
- They should not be public though!

Member Access Control: public, protected, private

- There are 3 levels of member (data or methods) access control:
  - public: members can be used by itself and the whole world; any function can access them.
  - protected: methods (and friends) of itself and any derived class can use it.
  - private: members can only be used by its own methods (and its friends).
- Without inheritance, private and protected have exactly same meaning.
- The only difference is that methods of a derived class can access protected members of a base class, but cannot access private members of a base class.

protected vs. private

- So why not always use protected instead of private?
  - Because protected means that we have less encapsulation: Remember that all derived classes can access protected data members of the base class.
  - Assume that later you decided to change the implementation of the base class having the protected data members.
  - For example, we might want to represent address by a new class called `Address` instead of string. If the `address` data member is private, we can easily make this change. The class documentation does not need to be changed.
  - If it is protected, we have to go through all derived classes and change them. We also need to update the class documentation.

protected vs. private

- In general, it is preferable to have private members instead of protected members.
- Use protected only where it is really necessary. private is the only category ensuring full encapsulation.

- In our example, there is no reason at all to make `name`, `address`, `dept` protected, as we can access the name and address through the public member functions:

Example: print Using Public Functions Only

```
void Student::print() const
{
    cout << "--- Student details ---" << endl
         << "Name: " << get_name() << endl
         << "Addr: " << get_address() << endl
         << "Department: " << get_dept() << endl
         << "Enrolled in:" << endl;

    for (int i = 0; i < num_courses; ++i) {
         enrolled[i].print();
    }
}
```

Example Again

- Let's use the print method now:

```
Person mouse("Mickey", "Disney World", arts);
Teacher einstein("Albert Einstein", "USA", physics, professor);
Student plato("Plato", "Greece", philosophy);
plato.enroll_course("COMP151");

mouse.print();
einstein.print();
plato.print();
```

Example Again: Output

(assume: **enum** Department { arts, physics, philosophy, ... })

```
--- Person details ---
Name: Mickey
Addr: Disney World
Dept: 0
--- Teacher details ---
Name: Albert Einstein
Addr: USA
Dept: 1
Rank: Full Professor
--- Student details ---
Name: Plato
Addr: Greece
Dept: 2
Enrolled in:
COMP151
```