Topic 10
Inheritance

- Inheritance – overview
- Accessing inherited public members
- Accessing inherited protected members
- Introducing new members
- Constructions
- Destructions
- Inheritance hierarchy

Inheritance

- A **derived class** (**subclass**) inherits all the members of the **base class** (**superclass**). In addition, it can add new members and modify the inherited functions.

```
class Person {
   string name ;
   int age ;
   string address ;
public:
   Person(const string& theName,
      int theAge=1,
      const string& theAddress="") ;
   string getName() const ;
   void rename(const string& theName) ;
   int getAge() const ;
   void increaseAge() ;
   string getAddress() const ;
   void moveTo(const string& theAddress) ;
} ;
```

```
enum Grade { FRESH=1,
   SOPHOMORE, JUNIOR, SENIOR } ;

         하위 클래스

class Student : public Person {      상위 클래스
   string schoolName ;
   Grade grade ;
public:
   Student(const string& theName,
      Grade theGrade=FRESH) ;
   string getSchoolName() const ;
   void setSchoolName(
      const string& theSchoolName) ;
   Grade getGrade() const ;
   void upGrade() ;
} ;
```

Inheritance

하위 클래스

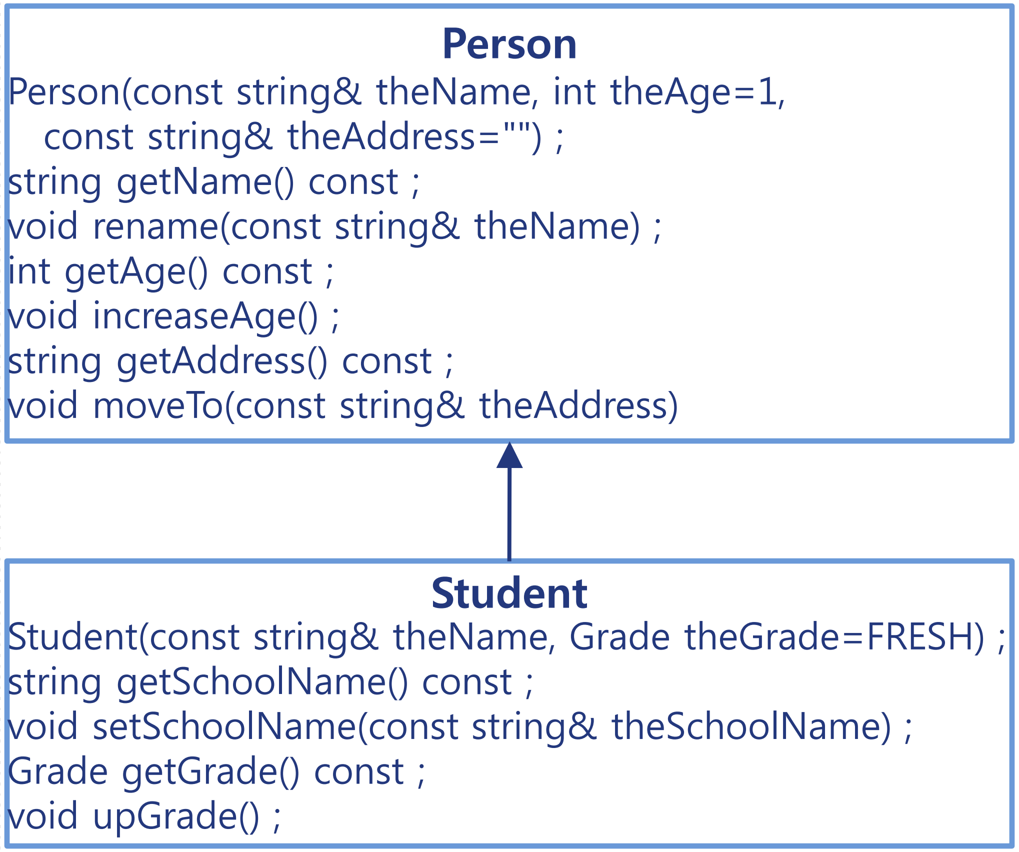
```
class Student : public Person {
```

상위 클래스

```
} ;
```

Inheritance

- Clients of a class can access the inherited members in addition to the members of the class itself.



Person

```
Person(const string& theName, int theAge=1,
   const string& theAddress="") ;
string getName() const ;
void rename(const string& theName) ;
int getAge() const ;
void increaseAge() ;
string getAddress() const ;
void moveTo(const string& theAddress)
```

Student

```
Student(const string& theName, Grade theGrade=FRESH) ;
string getSchoolName() const ;
void setSchoolName(const string& theSchoolName) ;
Grade getGrade() const ;
void upGrade() ;
```

```
int main() {
    Person p1("Brown"), p2("James") ;

    p1.rename("Jackson") ;
    p2.moveTo("Seoul") ;

    Student s1("Tom"), s2("Jane") ;
    s1.setSchoolName("한국대학교") ;
    s2.upGrade() ;

    s1.rename("Harrison") ;
    s2.increaseAge() ;
}
```

상속 멤버의 사용

- A subclass can access the **inherited public members** in addition to the members of the class itself.

```
enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
class Student : public Person {
    string schoolName ;
    Grade grade ;
public:
    …
    // operator overloading이 사실 바람직하다.
    void print() const {
        cout << "이름: " << getName() << endl ;        // name 직접 접근 불가
        cout << "나이: " << getAge() << endl ;         // age 직접 접근 불가
        cout << "주소: " << getAddress() << endl ;     // address 직접 접근 불가
        cout << "학교: " << schoolName << endl ;
        cout << "학년: " << grade << endl ;
    }
} ;
```

상속 멤버의 사용: protected member

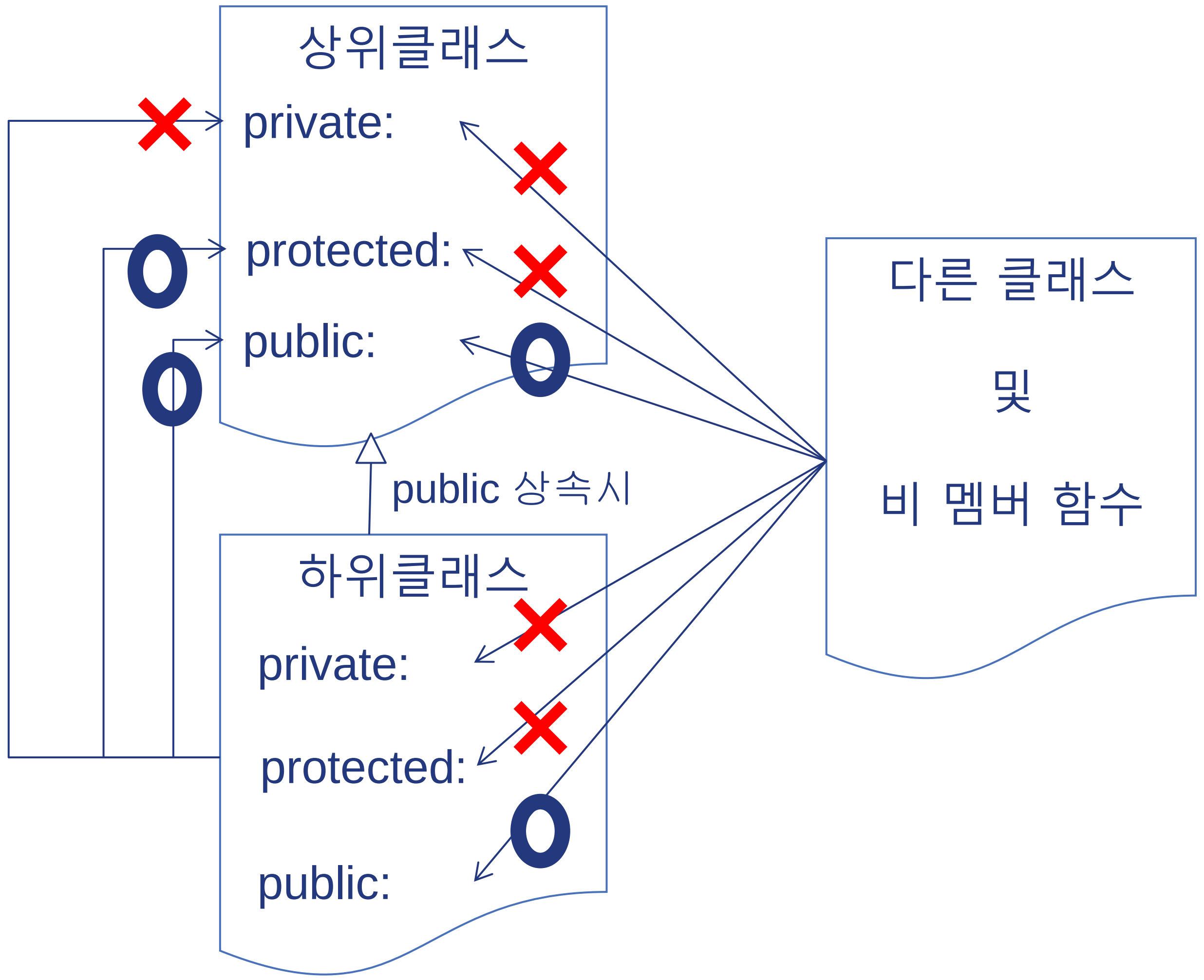
- **Protected members** can be directedly accessed from subclass.

```
class Person {
protected:
   string name ;
   int age ;
   string address ;

public:
   …
} ;
```

```
enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
class Student : public Person {
    string schoolName ;
    Grade grade ;
public:
    …
    // operator overloading이 사실 바람직하다.
    void print() const {
        cout << "이름: " << name << endl ;
        cout << "나이: " << age << endl ;
        cout << "주소: " << address << endl ;
        cout << "학교: " << schoolName << endl ;
        cout << "학년: " << grade << endl ;
    }
} ;
```

- **The use of protected members should be limited** because they can cause poor maintainability.

가시성(visibility) 요약



새 멤버의 추가

- A subclass can add new members for its own purpose.

```
enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
class Student : public Person {
    string schoolName ;
    Grade grade ;
public:
    Student(const string& theName, Grade theGrade=FRESH) : Person(theName) {
        this->grade = theGrade ;
    }
    string getSchoolName() const { return schoolName ; }
    void setSchoolName(const string& theSchoolName) { schoolName = theSchoolName ; }
    Grade getGrade() const { return grade ; }
    void upGrade() { if ( grade != SENIOR ) grade = Grade(grade+1) ; }
    void print() const {
        cout << "이름: " << getName() << endl ;
        cout << "나이: " << getAge() << endl ;
        cout << "주소: " << getAddress() << endl ;
        cout << "학교: " << schoolName << endl ;
        cout << "학년: " << grade << endl ;
    }
```

이름 충돌

- If there are members with the same name, they can be resolved by **scope operator ::**.

```
class Person {
protected:
   string name ;
   int age ;
   string address ;

public:
   …
} ;
```

```
enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
class Student : public Person {
    string name ; // 학교이름
    Grade grade ;
public:
    …
    string getSchoolName() const { return name ; }
    void setSchoolName(const string& theSchoolName) {
        name = theSchoolName ;
    }
    void print() const {
        cout << "이름: " << Person::name << endl ;
        cout << "나이: " << age << endl ;
        cout << "주소: " << address << endl ;
        cout << "학교: " << name << endl ;
        cout << "학년: " << grade << endl ;
    }
} ;
```

이름 충돌

```
class Person {
protected:
    string name ;
    int age ;
    string address ;

public:
    …

    void print() const {
       cout << "이름: " << name << endl ;
       cout << "나이: " << age << endl ;
       cout << "주소: " << address << endl ;
    }
} ;
```

```
class Student : public Person {
    string name ; // 학교이름
    Grade grade ;
public:
    …
    void print() const {
        Person::print() ;
        cout << "학교: " << name << endl ;
        cout << "학년: " << grade << endl ;
    }
} ;
```

```
int main() {
    Person p1("Brown") ;
    p1.print() ;

    Student s1("Tom") ;
    s1.print() ;
    s1.Person::print() ;
}
```

Inheritance and Overloading

- Overloading is not applied for inherited member functions.

```
class Person {
private:
  string name ;
  int age ;
  string address ;

public:
  …

  void print(int w) const {
     cout << "이름: " << setw(w) << name << endl ;
     cout << "나이: " << setw(w) << age << endl ;
    cout << "주소: " << setw(w) << address << endl ;
  }
} ;
```

```
class Student : public Person {
  string schoolName ;
  Grade grade ;
public:
  …
  void print() const {
   Person::print(20) ;
   cout << "학교: " << schoolName << endl ;
   cout << "학년: " << grade << endl ;
  }
} ;
```

```
int main() {
    Person p1("Brown") ;
    p1.print(10) ;

    Student s1("Tom") ;
    s1.print() ;
    s1.print(30) ;       // ERROR
    s1.Person::print(30) ;
}
```

Constructors

- A constructor of a subclass can initialize for members of a superclass through the constructor of the superclass.

```
class Person {
   string name ;
   int age ;
   string address ;
public:
   Person(const string& theName,
      int theAge=1, const string& theAddress="")
      : name(theName), address(theAddress) {
      this->age = theAge ;
      cout << "Person of " <<  name <<
          " are constructed !" << endl ;
   }
   …
}
```

```
class Student : public Person {
   string schoolName ;
   Grade grade ;
public:
   Student(const string& theName,
      const string& theSchoolName,
      Grade theGrade=FRESH,
      const string& theAddress="")
      :  Person(theName, theGrade+20,
theAddress),
         schoolName(theSchoolName) {
      this->grade = theGrade ;
      cout << "Student of " <<  getName()
         << " are constructed !" << endl ;
   }
```

Constructors

```
int main() {
   Student s1("Tom", "한국대학교"), s2("Jane", "미국대학교", JUNIOR, "LA") ;
   s1.print() ;
   s2.print() ;
}
```

```
Person of Tom are constructed !
Student of Tom are constructed !
Person of Jane are constructed !
Student of Jane are constructed !
이름: Tom
나이: 21
주소:
학교: 한국대학교
학년: 1
이름: Jane
나이: 23
주소: LA
학교: 미국대학교
학년: 3
```

하위 클래스 생성자의 동작 방식

```
class S : public P {
  C1 c1 ;
  C2 c2 ;
  int x ;
public:
  S() : c2(), c1(), P() { x = 0 ; }
} ;
```

하위 클래스의 생성자 S() 수행

1) 상위 클래스 P의 P() 생성자 수행
2) 객체 멤버 c1의 C1() 생성자 수행
3) 객체 멤버 c2의 C2() 생성자 수행
4) 생성자 S()의 본문 수행

기본 생성자 호출 생략

기본 생성자 호출 명시	기본 생성자 호출의 생략

class S : public P {
  C1 c1 ;
  C2 c2 ;
  int x ;
public:
  S() : **c2(), c1(), P()** { x = 0 ; }
} ;
	class S : public P {
	  C1 c1 ;
	  C2 c2 ;
	  int x ;
	public:
	  S() { x = 0 ; }
	} ;

Destructors

- The destructor of a superclass is automatically invoked in that of a subclass

```
class Person {
   string name ;
   int age ;
   string address ;
public:
   Person(const string& theName,
      int theAge=1, const string& theAddress="")
      : name(theName), address(theAddress) {
      this->age = theAge ;
      cout << "Person of " <<  name <<
          " are constructed !" << endl ;
   }
   ~Person() { cout << "Person of " << name
      << " are destructed !" << endl ;
   }
}
```

```
class Student : public Person {
   string schoolName ;
   Grade grade ;
public:
   Student(const string& theName,
      const string& theSchoolName,
      Grade theGrade=FRESH,
      const string& theAddress="")
      :  Person(theName, theGrade+20,
theAddress),
         schoolName(theSchoolName) {
      this->grade = theGrade ;
      cout << "Student of " <<  getName()
         << " are constructed !" << endl ;
   }
   ~Student() { cout << "Student of " <<
      getName() << " are destructed !" << endl ;
}
```

Destructors

```
int main() {
    Student s1("Tom", "한국대학교") ;
    {
       Student s2("Jane", "미국대학교", JUNIOR, "LA") ;
       s2.print() ;
    }
    s1.print() ;
}
```

```
Person of Tom are constructed !
Student of Tom are constructed !
Person of Jane are constructed !
Student of Jane are constructed !
이름: Jane
나이: 23
주소: LA
학교: 미국대학교
학년: 3
Student of Jane are destructed !
Person of Jane are destructed !
이름: Tom
나이: 21
주소:
학교: 한국대학교
학년: 1
Student of Tom are destructed !
Person of Tom are destructed !
```

하위 클래스 소멸자의 동작 방식

```
class S : public P {
  C1 c1 ;
  C2 c2 ;
  int x ;
public:
  S() : P(), c1(), c2(), P() { x = 0 ; }
  ~S() { … }
} ;
```

1) 하위 클래스 S의 ~S() 소멸자 수행
2) 객체 멤버 c2의 ~C2() 소멸자 수행
3) 객체 멤버 c1의 ~C1() 소멸자 수행
4) 상위 클래스 P의 ~P() 소멸자 수행

Copy construction and Assignment

- Copy construction and Assignment of a superclass with a subclass is possible, but not reverse.

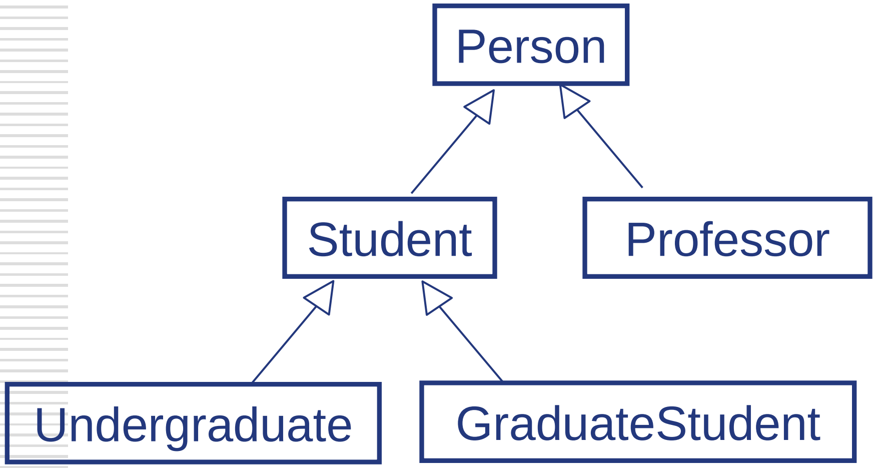
```
int main() {
   Student s1("Tom", "한국대학교") ;
   Person p1(s1) ;
   p1.print() ;

   Person p2("Jane", 30, "서울") ;
   p2 = s1 ;
   p1.print() ;

   Student s2(p1) ;  // ERROR
}
```

```
이름: Tom
나이: 21
주소:
이름: Tom
나이: 21
주소:
```

- However, they should be prohibited !

Inheritance Hierarchy

- A set of related classes comprizes an hierarchy.



```
class Food ;
class Person {
public:
    void sleep() ;
    void eat(Food food) ;
} ;
class Course ;
class Student : public Person {
public:
    void transferTo(School school) ;
    void takeCourse(Course course) ;
    void takeExam(Course course) ;
} ;
class GraduateStudent : public Student {
public:
    void writeThesis() ;
    void participateIn(Project project) ;
    void assignAdvisor(Professor professor) ;
} ;
class Professor : public Person {
public:
    void teach(Course course) ;
    void lead(Project project) ;
} ;
```

Practice #1

```
enum EmployeeLevel { 사원, 대리, 과장, 차장, 부장} ;
class Employee {
    string name ;
    EmployeeLevel level ;
public:
    …
} ;
class Manager : public Employee {
    vector<Employee*> group ;
public:
    …
} ;
int main() {
   Employee e1("홍", 사원), e2("김", 대리), e3("차", 사원) ;
    cout << e1 << e2 << e3 ;

    Manager m1("Tom", 차장) ;
    m1.addEmployee(&e1) ;
    m1.addEmployee(&e2) ;
    m1.addEmployee(&e3) ;
    cout << endl << "Information for Manager" << endl ;
    cout << m1 ;
}
```

```
0       홍
1       김
0       차

Information for Manager
3       Tom
List of employees managed by me
0       홍
1       김
0       차
```

Practice #2

- Construct an inheritance hierarchy from the list of classes.
  - Mouse, Keyboard, LCD Monitor, CRT Monitor, Device, Printer, Scanner, Input Device, Output Device, Monitor
- Based on the hierarchy, add member functions relevant to each class.