Lecture #11

Inheritance (2) : Polymorphism(2)

SE271 Object-Oriented Programming (2020)

Yeseong Kim

Original slides from Prof. Shin at DGIST

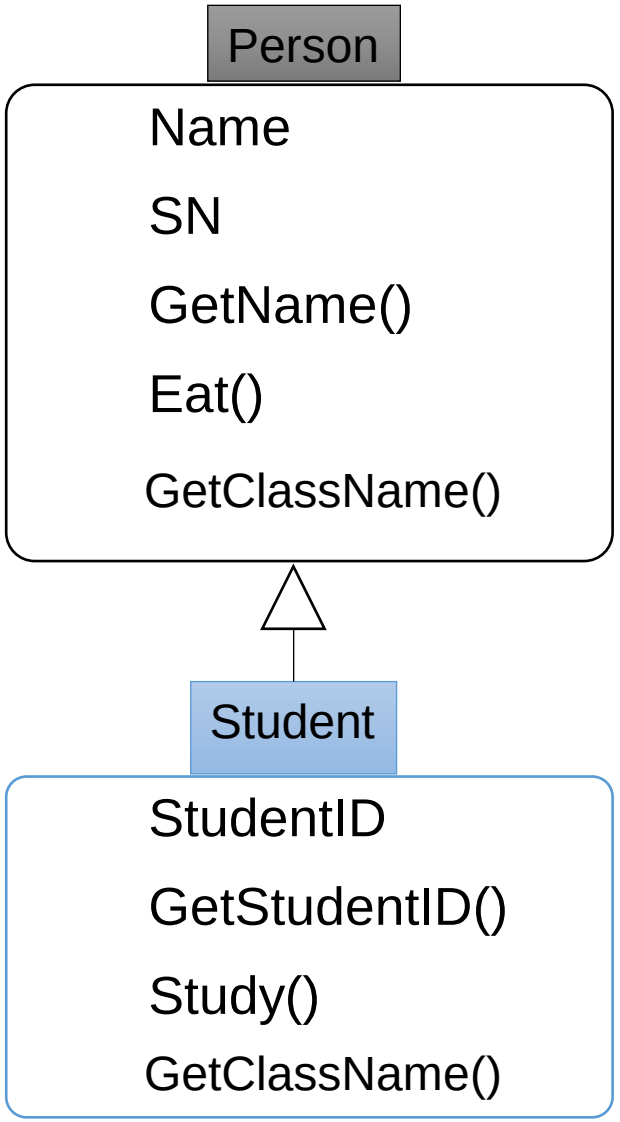
Short Notice

- HW3 will be released today
  - PLEASE comply the way-to-submit written in the instruction
  - PLEASE don't include main
  - Zip filename: HW3_학번.zip
    - Files in the zip file:
      - hw3.cpp hw3.h (O)
      - hw3_학번.cpp 학번_hw3.cpp (X)

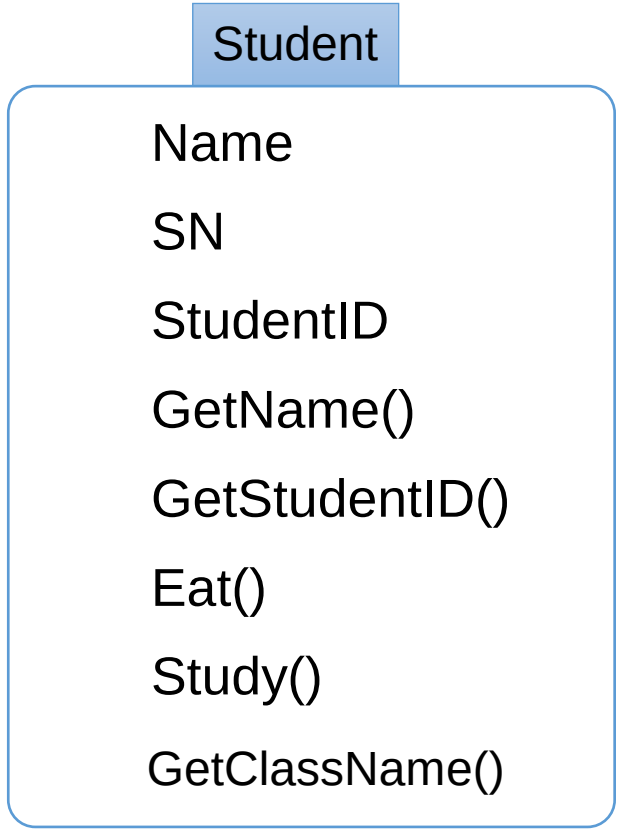
Today's Topic

- Inheritance
  - –Function Overriding: A way to implement polymorphism with inheritance
  - –Virtual Function
    - Virtual Function Binding
    - Pure Virtual Function/Class
    - Interface class
    - Specifier *final, override*

Example: Duplicated Method




```
string GetClassName()
{
    return ClassName;
}
```




```
Student s1;
cout << s1.GetClassName();
```

Function Overriding

- Syntax
  - The same signature with the same return type

```
class Base {
    return_type function_name (parameters);
};
class Derived : inheritance_type Base {
    return_type function_name (parameters);
};
```

```
string GetClassName(int a) {
    function body
}

string GetClassName(int a) {
    function body
}
```

Example: Function Overriding

```
class CPU {
private:
    int m_serialNumber = 1001;
public:
    int GetSerialNumber( ) {
        return m_serialNumber;
    }
};
class Computer : public CPU{
private:
    int m_serialNumber = 2001;
public:
    int GetSerialNumber( ) {
        return m_serialNumber;
    }
};
```

```
int main() {
    CPU myCPU;
    cout << myCPU.GetSerialNumber() << endl;

    Computer myPC;
    cout << myPC.GetSerialNumber() << endl;

// How can we know cpu S/N in my computer?
cout << myPC.CPU::GetSerialNumber() << endl;

}
```

How to Call Override Function with Up Casting

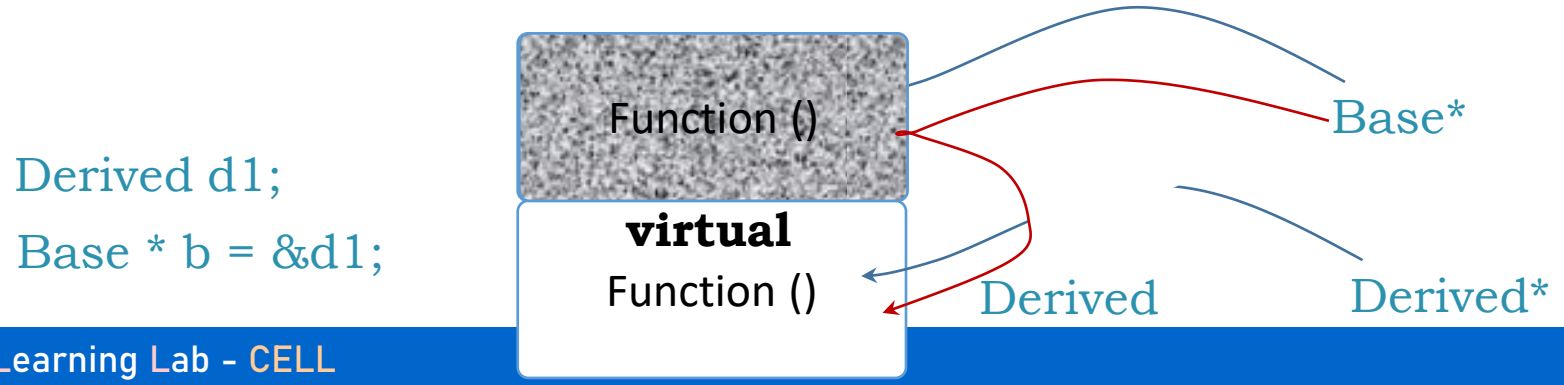
```
class Person {
public:
    std::string GetClassName() {
        return "Person";    }
};
class Student : public Person {
public:
    std::string GetClassName() {
        return "Student";   }
};
class Professor : public Person {
public:
     std::string GetClassName() {
        return "Professor";    }
};
```

```
int main() {
    Student s1;
    Professor f1;

    cout << s1.GetClassName() << endl;

    Person* dgist_member[2] = { &s1, &f1 };
    cout << dgist_member[0]->GetClassName() << endl;
    cout << dgist_member[1]->GetClassName() << endl;
}
```

Virtual Function

- Syntax
  - Write ‘virtual’ in front of the declaration of a function

```
class Base {
    virtual return_type function_name (parameters);
};
class Derived : inheritance_type Base {
    virtual return_type function_name (parameters);
};
```



Example: Virtual Function

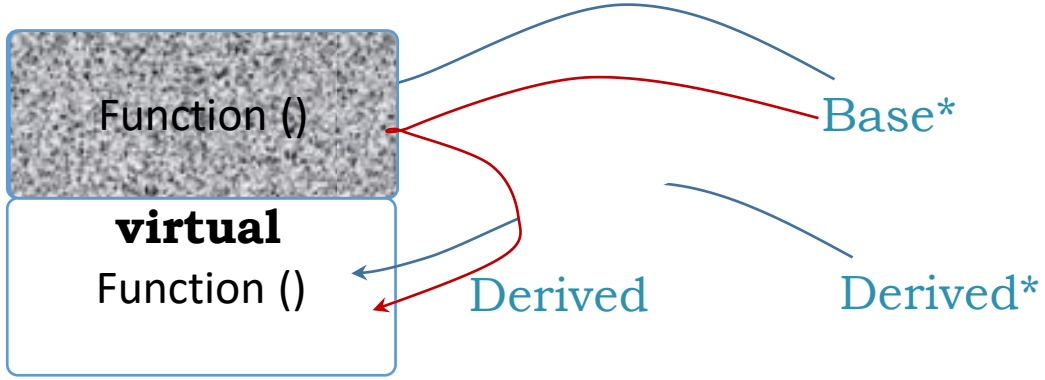
```
class Person {
public:
    virtual void work( ) {
        cout << “Working”;
    }
};
class Student : public Person{
public:
    virtual void work( ) {
        cout << “Studying”;
    } };
class Professor: public Person{
public:
    virtual void work( ) {
        cout << “Teaching”;
    } };
```

```
void Working(Person* p) {
    p->work();
    cout << endl;
}
int main() {
    Student s1, s2;
    Professor f1;
    Person* dgist_member[3] = { &s1,&s2,&f1 };
    for(int i=0;i<3;i++)
        Working(dgist_member[i]);
}
```

```
Studying
Studying
Teaching
```

Virtual Function – Dynamic Binding

```
class Person {
public:
    virtual void work( );
};
class Student : public Person{
public:
    virtual void work( );
};
class Professor: public Person{
public:
    void work( ); // virtual can be omitted.
};
void Working(Person* p) {
    p->work();
}
```



Why Virtual Function (destructor)

- It helps developers in the future (eg. Destructor)

```
class Person {
    int * SN;
public:
    Person() : SN{new int(0)} {
    }

    ~Person() {
        delete SN;
    }
};
```

```
class Student : public Person{
    int * S_ID;
public:
    Student() : S_ID{new int(0)} { }
    ~ Student() {delete S_ID; }
};
class Professor : public Person{
    int * E_ID;
public:
     Professor() : E_ID{new int(0)} { }
    ~ Professor() {delete E_ID; }
};
```

```
int main() {
    Student* s1 = new Student();
    Professor* f1 = new Professor();

    delete s1;
    Person* p1 = f1;
    delete p1;

    return 0;
}
```

Pure Virtual Function (Abstract function)

- Syntax
  - It should be implemented by derived class

```
class Base {
    virtual return_type function_name (parameters) = 0;
};
class Derived : inheritance_type Base {
    virtual return_type function_name (parameters) {
        // function body
    }
};
```

Example: Pure Virtual Function

```
class Person {
public:
    virtual void work( ) = 0;
};
class Student : public Person{
public:
    void study( ) { }
};
class Professor: public Person{
public:
    virtual void work( ) { }
};
```

```
int main() {
    Person p1; // error

    Student s1; // error

    Professor f1;

    return 0;
}
```

Abstract Base Class (Pure Virtual Class)

- Class with at least one pure virtual function

```
class Product {
public:
    virtual int GetSerialNumber() = 0;
    bool CheckFakeProduct() {
        if (this->GetSerialNumber() > 999)
            return true;
        else
            return false;
    }
};
```

```
class TV : public Product {
public:
    int GetSerialNumber() { return 1000; }
};


int main() {
    Product* p = new TV;
    cout << p->CheckFakeProduct();
}
```

Interface Class

- Class including pure virtual functions only

```
class Product {
public:
    virtual int GetSerialNumber() = 0;
    virtual bool CheckFakeProduct() = 0;
    virtual std::string GetName() = 0;
    virtual std::string Copyright() = 0;
    …
};
```

final (1)

- Specifier *final* does not allow being overrided from derived classes
  - For functions

```
class Person {
public:
    virtual void work( ) final;
};

class Student : public Person{
public:
    virtual void work( );
};
```

```
class GrandFather {
public:
    virtual void work();
};
class Father : GrandFather {
public:
    virtual void work() final;
};
class Son : public Father {
public:
    virtual void work();    };
```

final (2)

- Specifier *final* does not allow being overrided from derived classes
  - For classes

```
class Person {
public:
    virtual void work( );
};

class Student final : public Person {
public:
    virtual void work( );
};
```

```
class Tutor: public Student{
public:
    virtual void work( );
};
```

override (c++11)

- Specifier *override* checks if the function override or not

```
class Person {
public:
    virtual void work( int hour);
};

class Student : public Person {
public:
    virtual void work(short int hour );
    virtual void work(int hour ) const;
};
```

```
class Person {
    public:
    virtual void work(int hour);
};

class Student : public Person {
public:
    virtual void work(short int hour) override;
    virtual void work(int hour) override;
};
```

References

- Learn c++
  - https://www.learncpp.com/
  - Chapter : 11.6, 12.1-4, 12.6

ANY QUESTIONS?