Introduction to Inheritance

Class Augmentation

Md. Saidul Hoque Anik
onix.hoque.mist@gmail.com

Last Modified: 2 April, 2018

Two Types of Roles

Two Types of Roles

Computer Maker

Computer User

Two Types of Roles

Class Designer (Class Maker)

Those who design classes

Developer (Class User)

Those who use the classes (by making objects)

Two Types of Access Modifiers

(Think like a Class Designer)

`private` `public`

Two Types of Access Modifiers

`private`

Your toothbrush
(Only you can use it)

`public`

Two Types of Access Modifiers

`private`

Your toothbrush
(Only you can use it)

`public`

Your toothpaste
(Anyone can use it)

Inheritance

Creating a class from another class(s)

X **=** 1 + 2 + 3 + 4

Inheritance

Creating a class from another class(s)

X = 1 + 2 + 3 + 4

Y = 1 + 2 + 3 + 4 + 5

Inheritance

Creating a class from another class(s)

X = 1 + 2 + 3 + 4

Y = X + 5

Inheritance

Creating a class from another class(s)

```
class Point
{
public:
    int x;
    int y;
};
```

Inheritance

Creating a class from another class(s)

```
class Point
{
public:
    int x;
    int y;
};
```

```
class Point3D
{
public:
    int x;
    int y;
    int z;
};
```

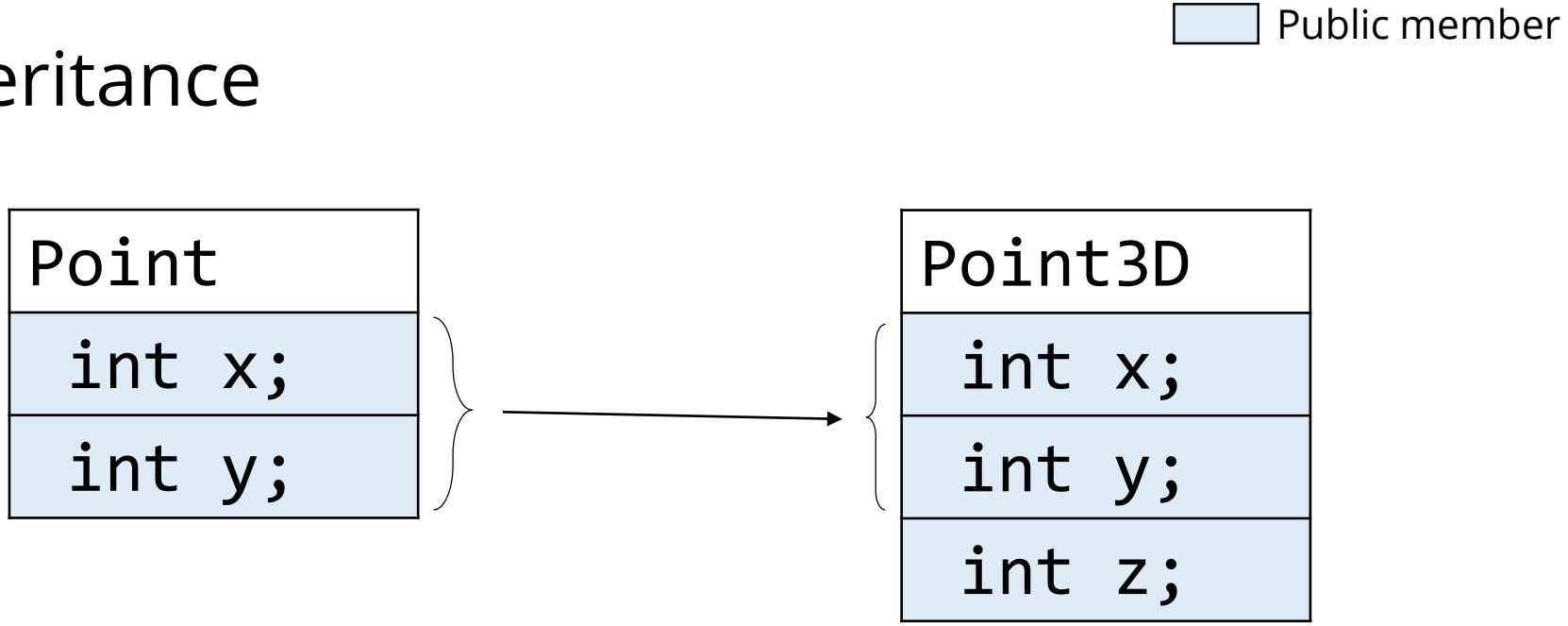
Inheritance

Creating a class from another class(s)

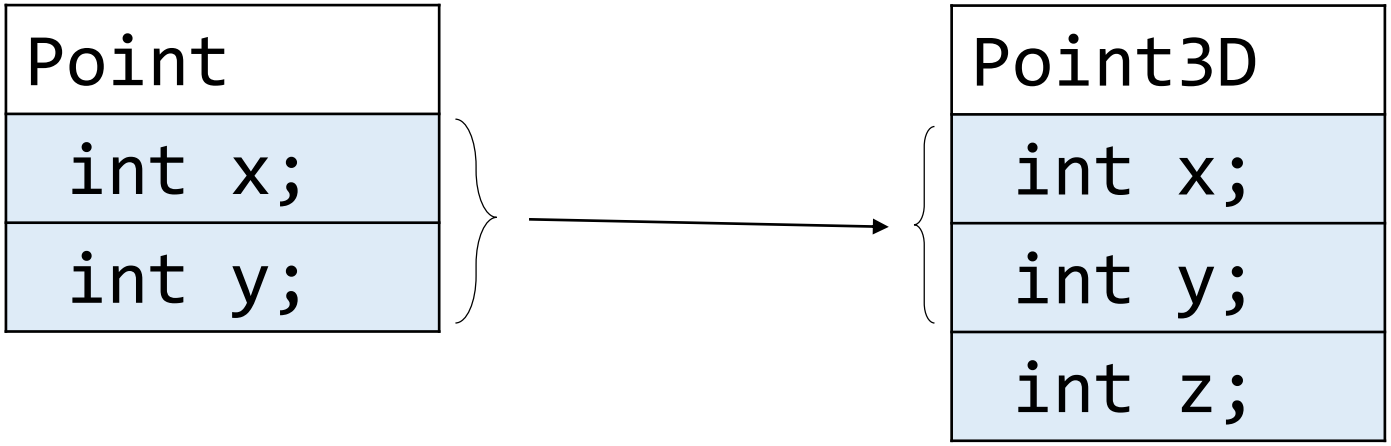
```
class Point
{
public:
    int x;
    int y;
};
```

```
class Point3D
{
public:
    int x;
    int y;
    int z;
};
```

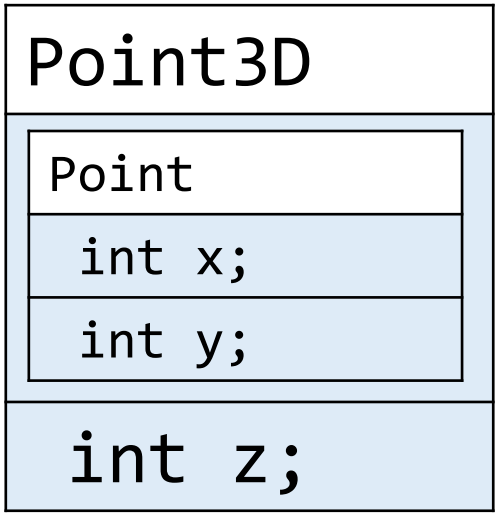
Inheritance



Inheritance



We need something like this:



Inheritance Syntax

```
class Point
{
public:
    int x;
    int y;
};
```

Point
int x;
int y;

Inheritance Syntax

```
class Point
{
public:
    int x;
    int y;
};

class Point3D :
{


};
```

Notice the mark

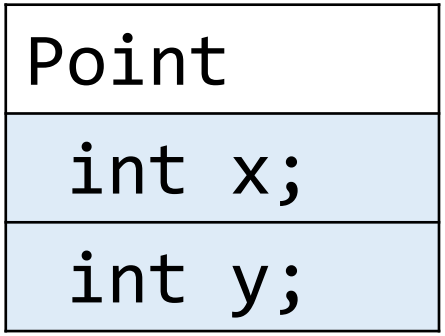
Point
int x;
int y;

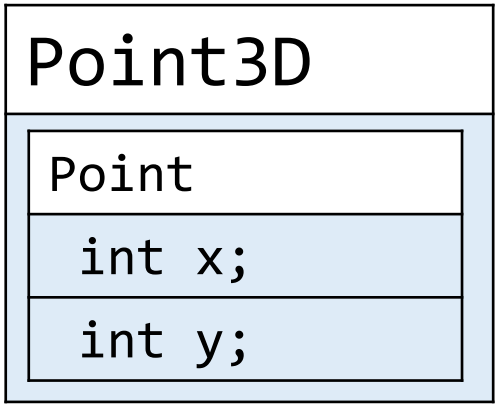
Point3D

Inheritance Syntax

```
class Point
{
public:
    int x;
    int y;
};

class Point3D : public Point
{


};
```

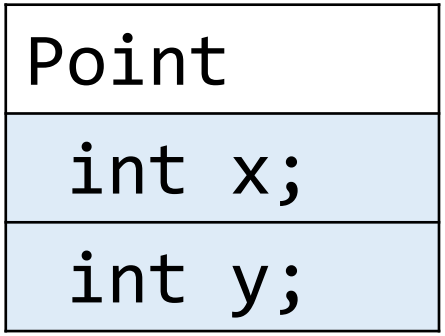
Not 'class Point'

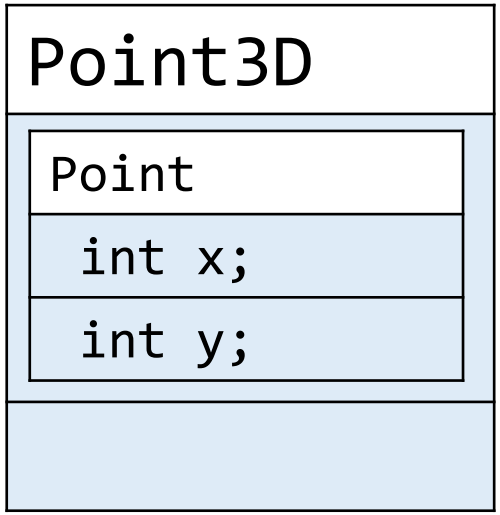




Inheritance Syntax

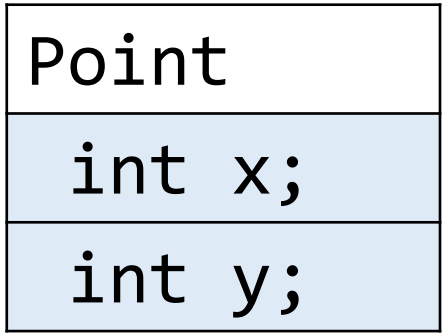
```
class Point
{
public:
    int x;
    int y;
};

class Point3D : public Point
{
public:

};
```



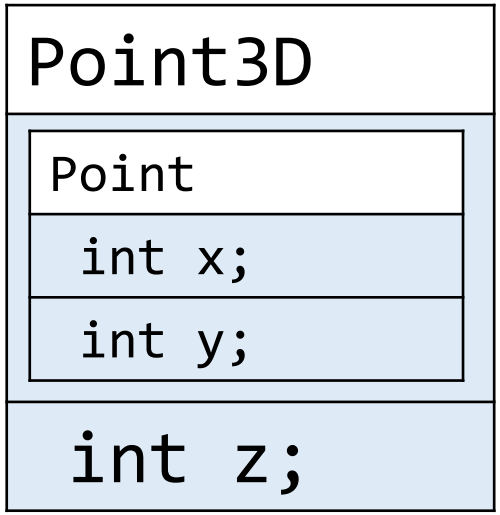


Inheritance Syntax

```
class Point
{
public:
    int x;
    int y;
};
```




```
class Point3D : public Point
{
public:
    int z;
};
```



Class designer's view

Inheritance Syntax

```
class Point
{
public:
    int x;
    int y;
};
```

Point
int x;
int y;

```
class Point3D : public Point
{
public:
    int z;
};
```

Point3D
int x;
int y;
int z;

Developer's view

Terminology

Derived class *Base class*

```
class Point3D : public Point
```

Point3D
int x;
int y;
int z;

Point
int x;
int y;

A More Useful Example

An undergraduate student in MIST receives marks in Theory and Sessional individually. Design a Student class that will hold the respective marks. Write appropriate getter and setter functions (except constructor).

Private member

Student Class

```
Student
 int theory;
 int sessional;
 void setMarks(...);
 int getTheory();
 int getSessional();
```

Private member

Extending Student Class

Student
int theory; int sessional;

void setMarks(...);
int getTheory();
int getSessional();

L4Student
... int thesis;

...
void setThesis(...);
int getThesis();

Extending Student Class

```
class L4Student
```

Student
int theory; int sessional;

void setMarks(_);
int getTheory();
int getSessional();

L4Student

Extending Student Class

```
class L4Student : public Student
```

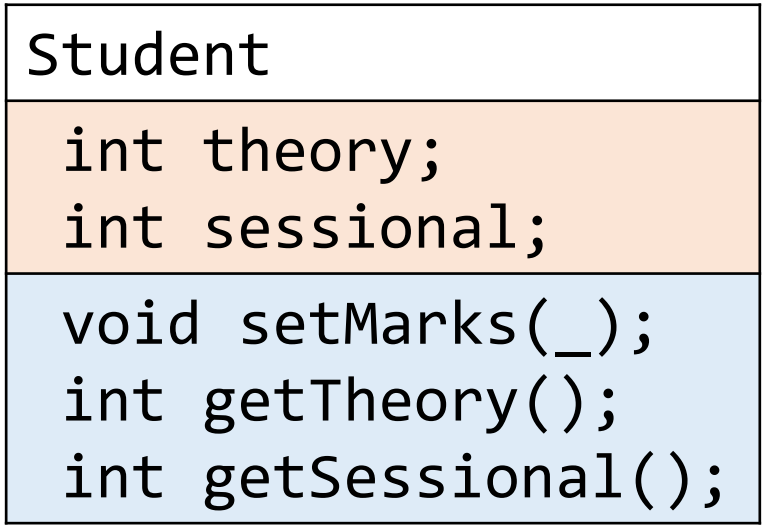
Student
int theory; int sessional;

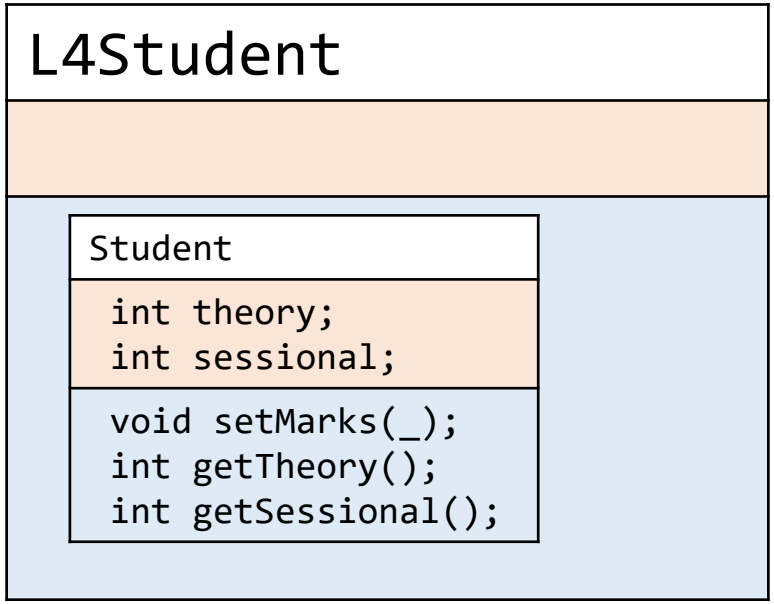
void setMarks(_);
int getTheory();
int getSessional();

L4Student

Extending Student Class

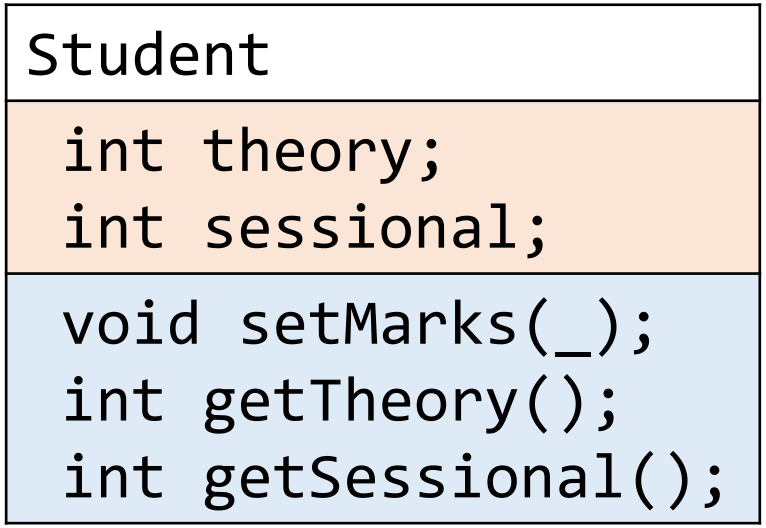
```
class L4Student : public Student
```

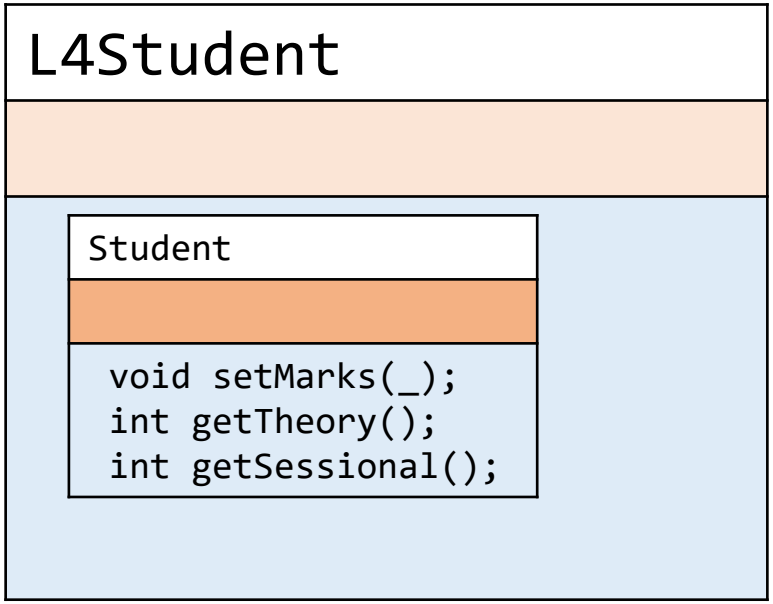




Extending Student Class

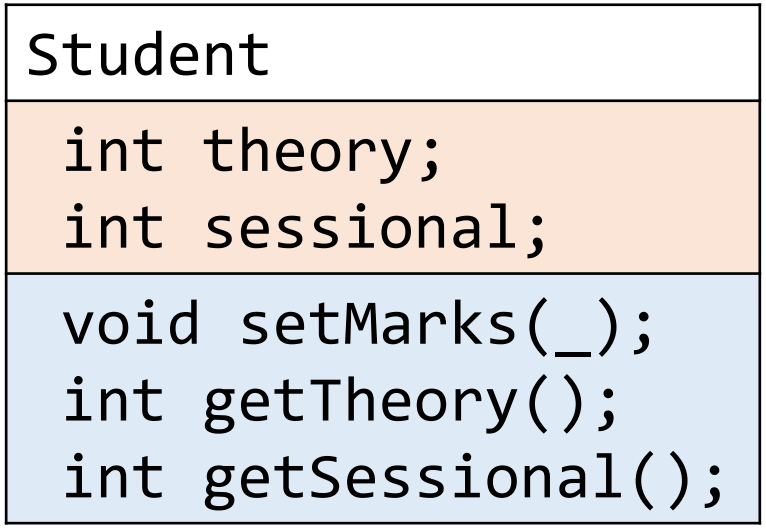
```
class L4Student : public Student
```

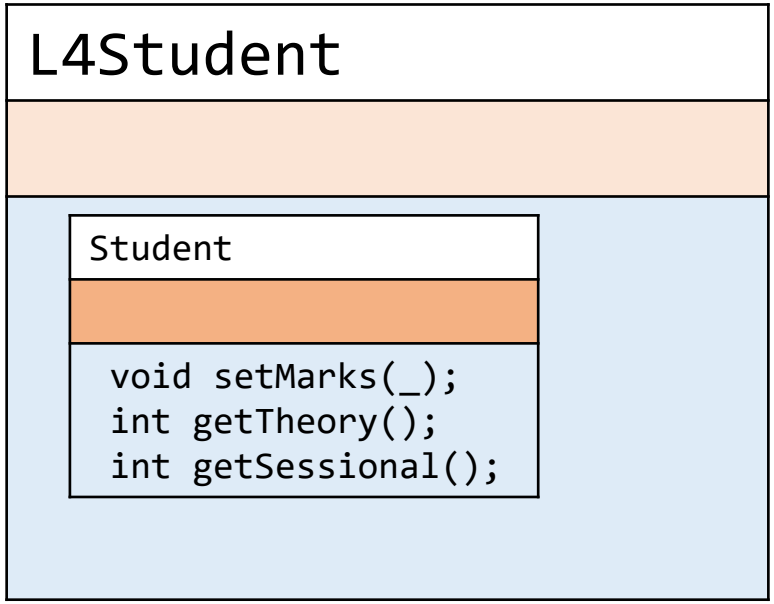




Extending Student Class

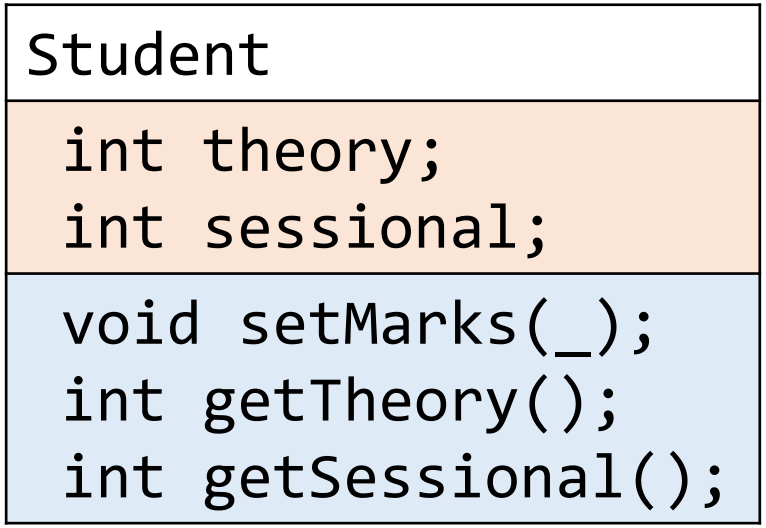
```
class L4Student : public Student
{
private:
```

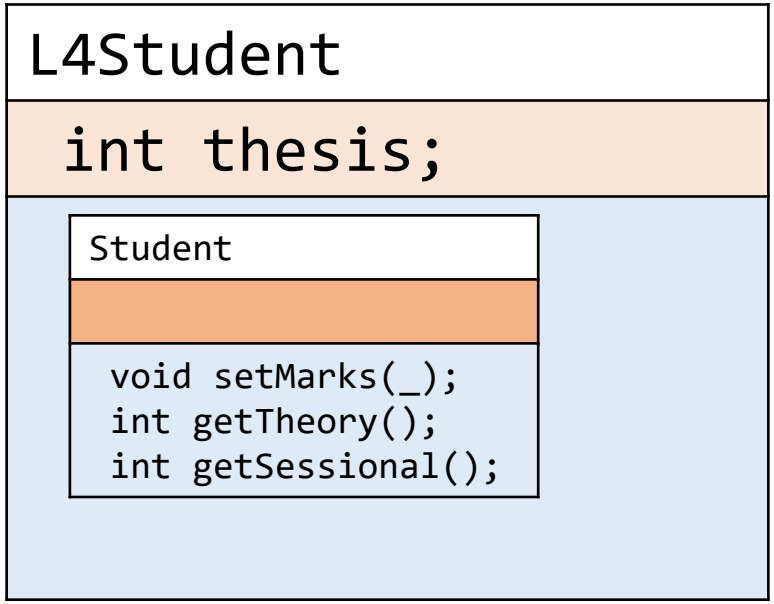




Extending Student Class

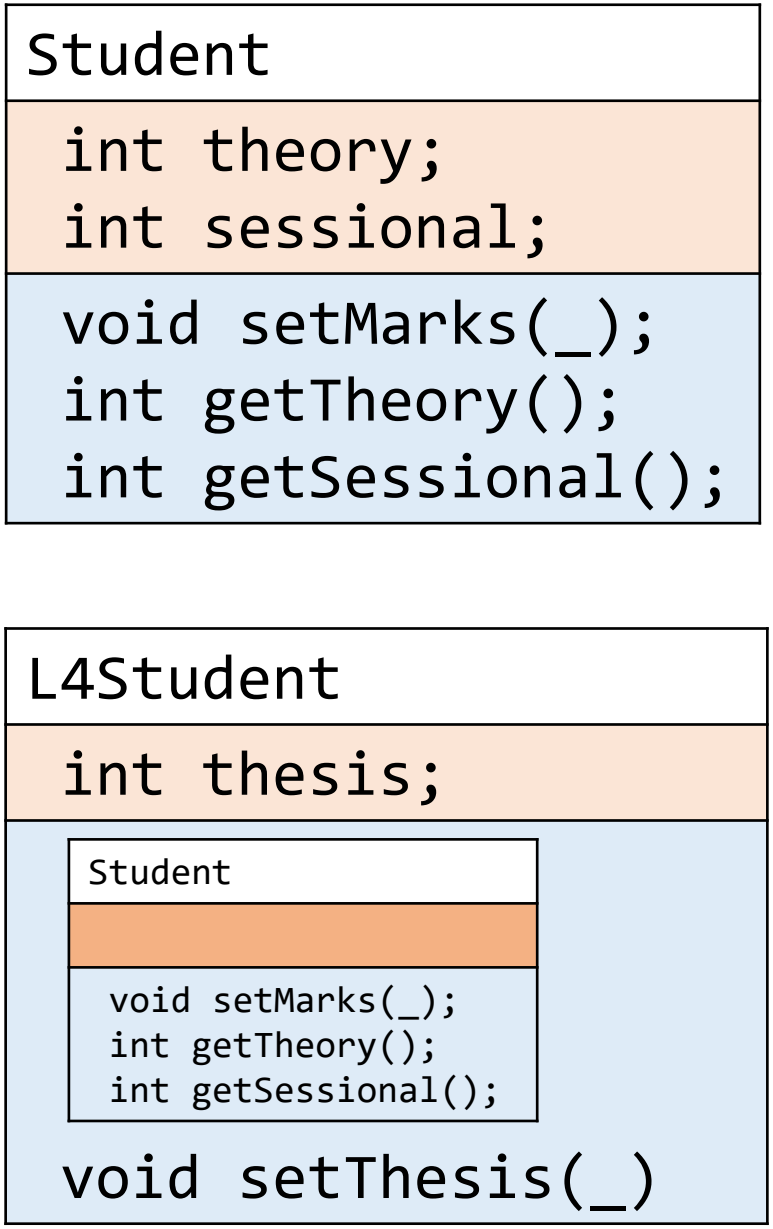
```
class L4Student : public Student
{
private:
    int thesis;
```





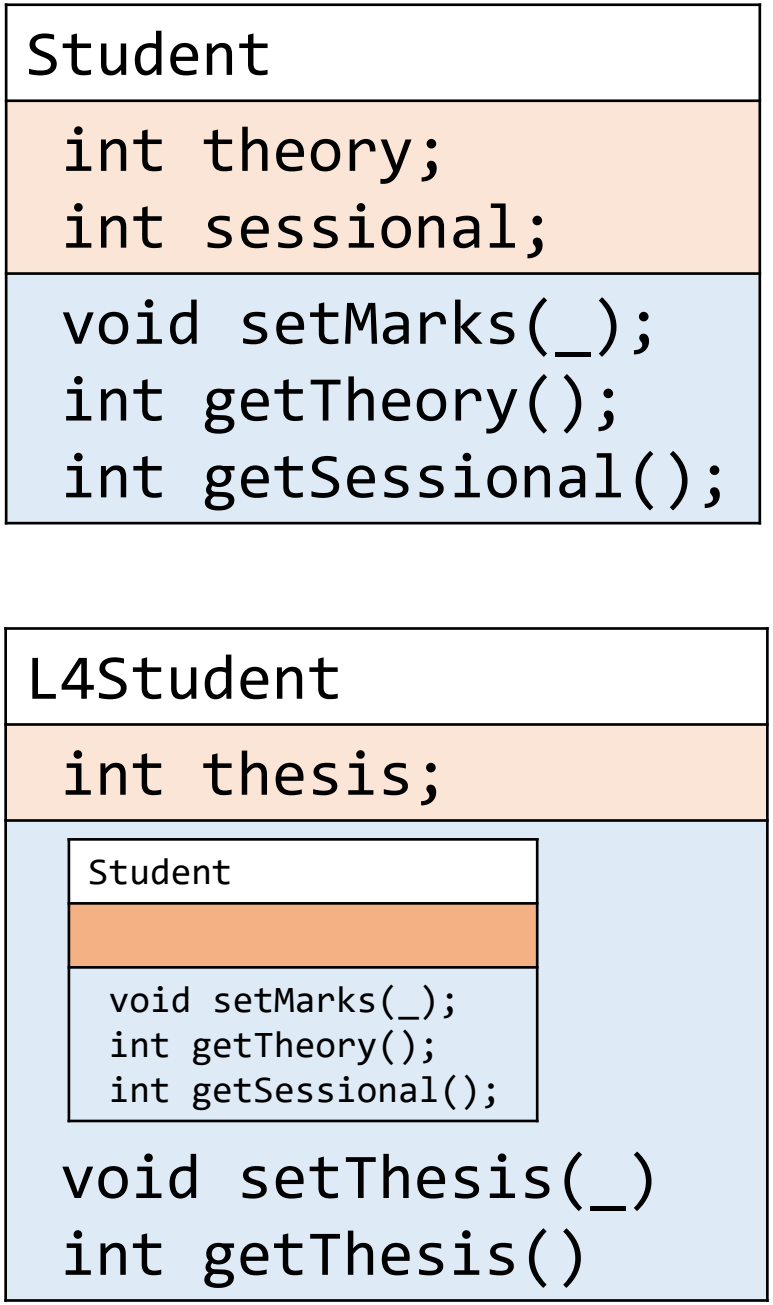
Extending Student Class

```
class L4Student : public Student
{
private:
    int thesis;
public:
    void setThesis(int t)
    {
        thesis = t;
    }
```

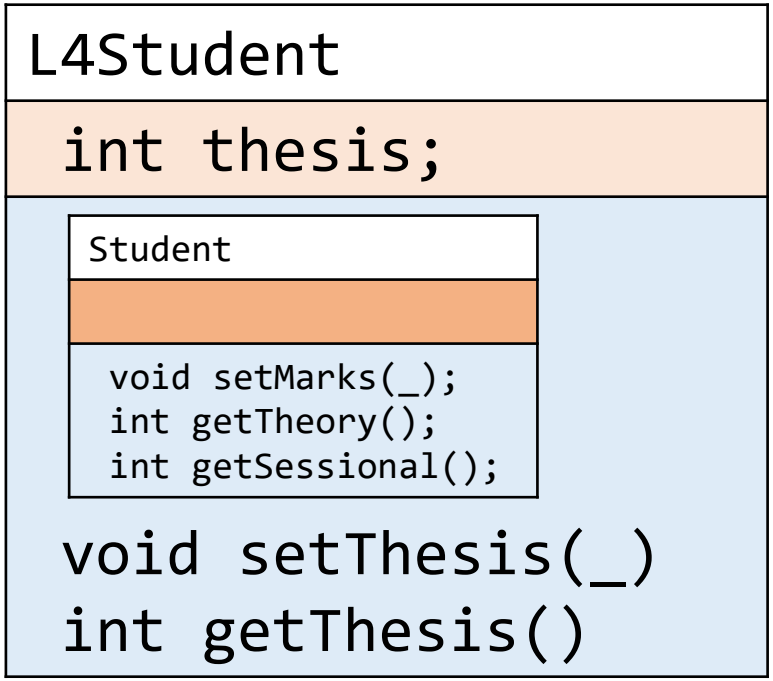


Extending Student Class

```
class L4Student : public Student
{
private:
    int thesis;
public:
    void setThesis(int t)
    {
        thesis = t;
    }
    int getThesis()
    {
        return thesis;
    }
};
```

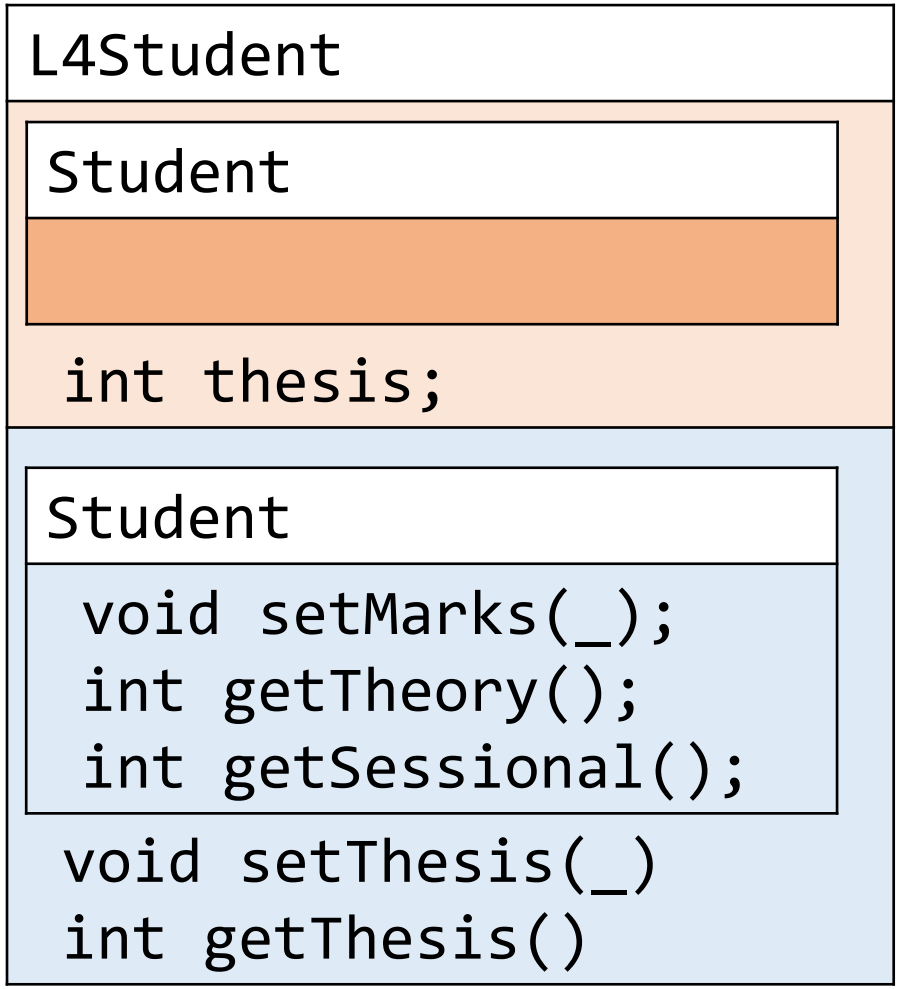


L4Student Class



Class designer's view

L4Student Class



Class designer's view
(Expanded)

L4Student Class

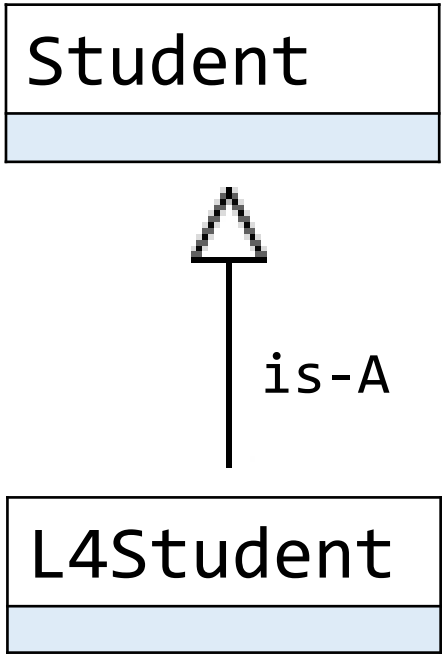
```
L4Student
---------------------------
void setMarks(_);
int getTheory();
int getSessional();
void setThesis(_)
int getThesis()
```

Developer's view

Notice that

L4Student	<u>is a</u>	Student
Point3D	<u>is a</u>	Point
Triangle	<u>is a</u>	Shape
SalaryAccount	<u>is an</u>	Account
Car	is a	Vehicle

`is-A` Notation (Applicable for public inheritance)

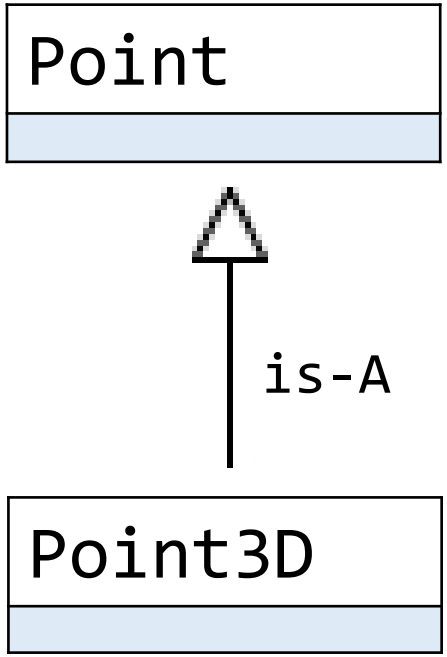


L4Student `is-A` **Student**

Derived-Class `is-A` **Base-Class**

The direction of the arrow may look counter-intuitive, but it’s accurate.

`is-A` Notation (Applicable for public inheritance)



Point3D `is-A` **Point**

Derived-Class `is-A` **Base-Class**

The direction of the arrow may look counter-intuitive, but it’s accurate.

Private Inheritance

```
class L4Student : private Student
```

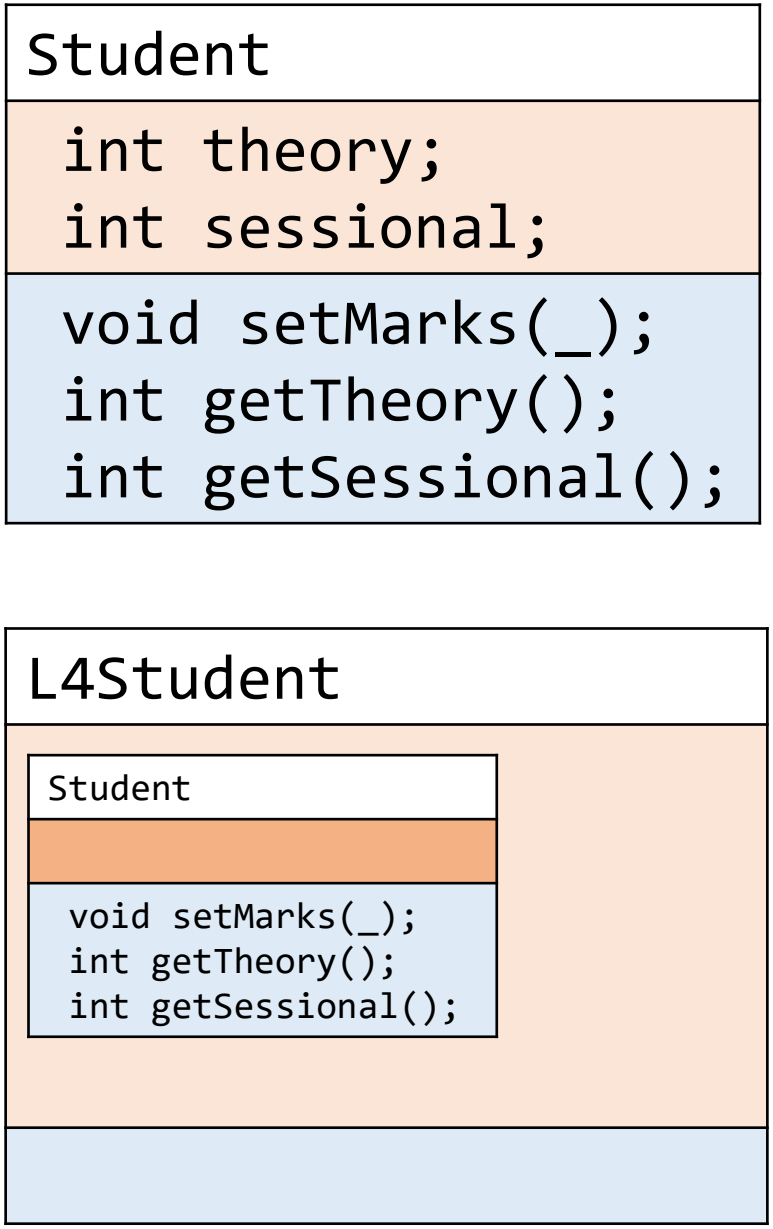
Student
int theory; int sessional;

void setMarks(_);
int getTheory();
int getSessional();

L4Student

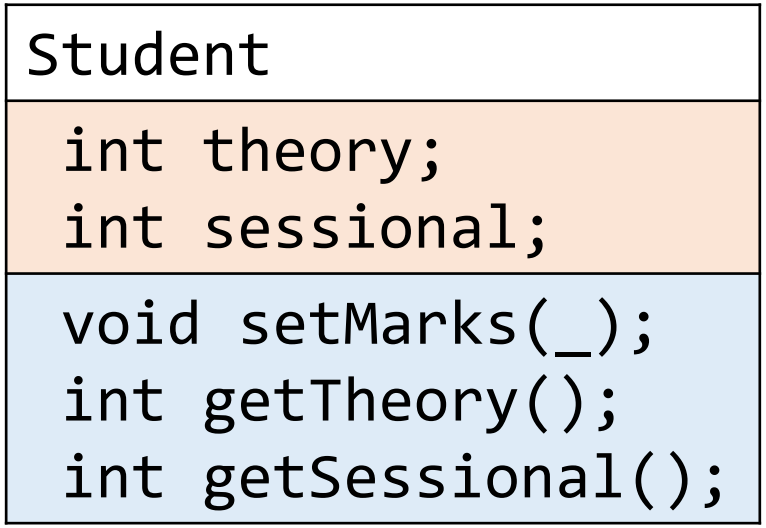
Private Inheritance

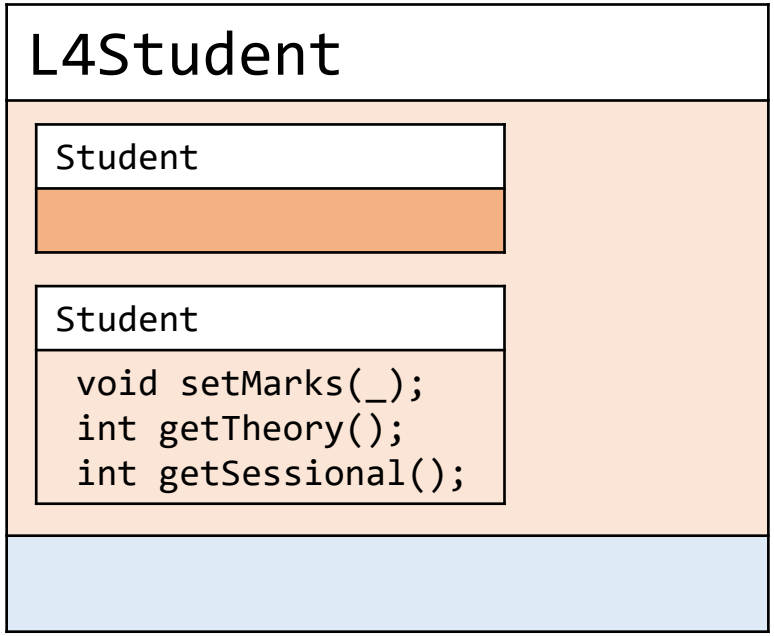
```
class L4Student : private Student
```



Private Inheritance

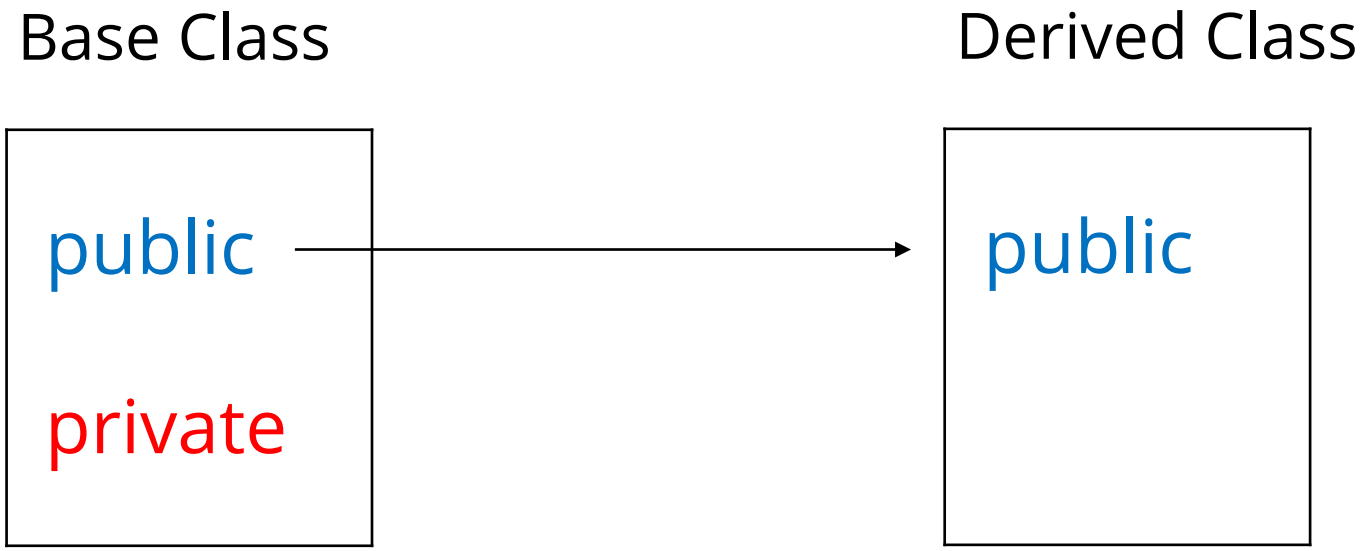
```
class L4Student : private Student
```





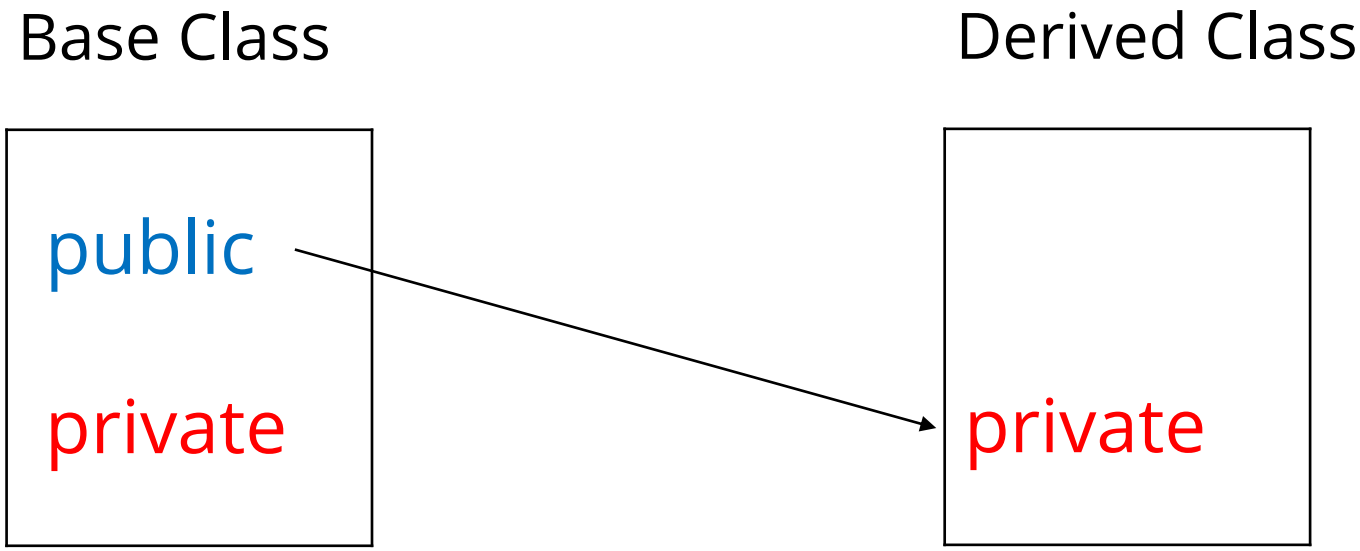
Access Specifier Summary

`public` Inheritance



Access Specifier Summary

`private` Inheritance



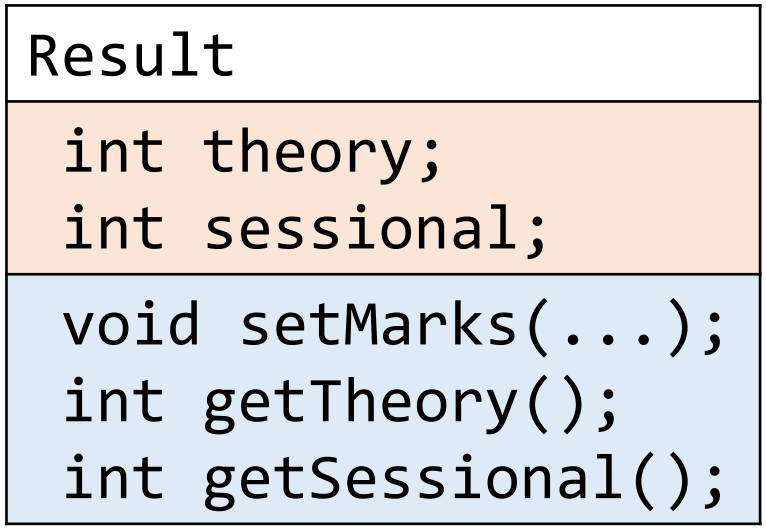
A better usage of Private Inheritance

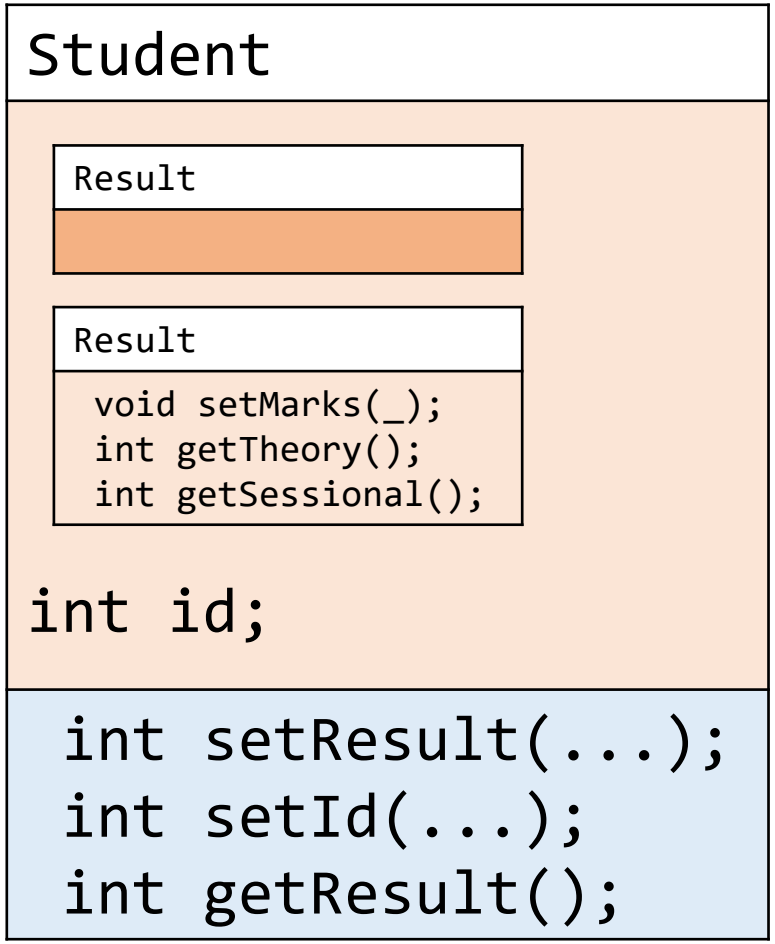
```
Result
  int theory;
  int sessional;
  void setMarks(...);
  int getTheory();
  int getSessional();
```

```
Student
  int id;
  int setResult(...);
  int setId(...);
  int getResult();
```

A better usage of Private Inheritance

```
class result : private Student
```





A better usage of Private Inheritance

```
class Student : private Result
{
    int id;
public:
    void setResult(int t, int s)
    {
        setTheory(t);
        setSessional(s);
    }
    void setId(int i) { id = i;}
    void publishResult()
    {
        printf("Student %d has got total %d marks\n",
               id, getTheory() + getSessional());
    }
};
```

These are called Delegate Functions, Because they 'delegate' the duties to another Functions.

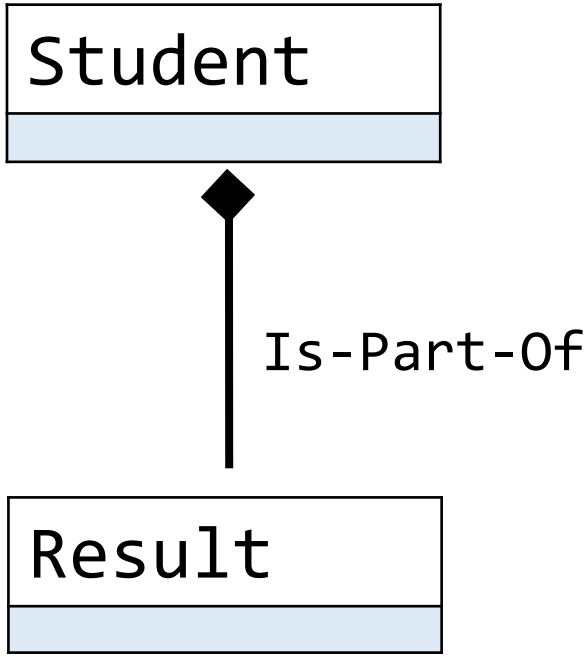
Note

Student is not a Result! So it’s not a `is-a` relation.

It’s a `has-a` relation.

Student `has-a` Result.

`has-A` Notation (private inheritance)



Student `has-A` **Result**

Car `has-A` **Engine**

The direction of the arrow may look counter-intuitive, but it’s accurate.

More Examples of `has-a`

Car has-an Engine

Body has-an Organ

School has-a Classroom

Course has-a ClassTest

Another Example

A person has a name, age and Address. An Address has home number, street number, and city name. Each class has a display() function that will display the contents.

- Design the class hierarchy
- Implement necessary functions