Inheritance

This lesson introduces the concept of Inheritance focusing on base and derived classes

We'll cover the following

What is Inheritance

- Provides a way to create a **new** class from an **existing** class.
- **New** class is a *specialized* version of the **existing** class.

Terminology

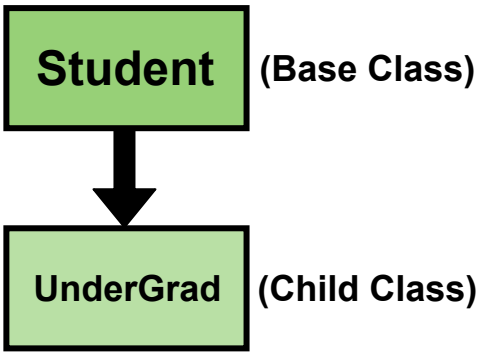
- **Base Class**(or Parent): *inherited* by **child** class.
- **Derived Class**(or child): *inherits* from base class.

Notation

Let's take a look at the notation for these **two** types.

```
class Student{ //base class
 //body
};

class UnderGrad : public Student{  //derived class
  //body
};
```



Inheritance establishes an **" is a"** relationship between *classes*.

An *object* of the *derived class* “is a” object of the *base class*. For example:

- An `UnderGrad` is `Student`.

> **Important Note:** A *derived* object has **all** characteristics of the *base* class.

What does a child have?

An *object* of **child** class has:

- All *members* defined in the **child** class.
- All *members* declared in the **parent** class.

An object of child class can use:

- All `public` members defined in the **child** class.
- All `public` members defined in the **parent** class.

Protected Members

- `protected` **member access specification**: similar to `private`, but accessible by objects of *derived* class.

Class Access Specifiers

- `public`: the *object* of the **derived** class can be treated as an object of the **base** class.
- `protected`: more restrictive than `public`, but allows **derived** classes to know details of *parents*.

- `private`: prevents objects of the **derived** class to be treated as objects of **base** class.

Example

Let's consider an example with *base* class `Shape` and *derived* class `Square`.

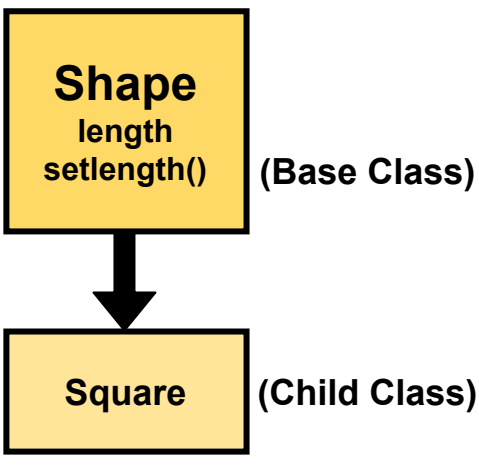
```
#include <iostream>
using namespace std;

// Base class
class Shape {
   public:
      Shape(){length = 0;} //default constructor
      void setlength(int l) {length = l;}
  protected:
      int length;
};

// Derived class
class Square: public Shape {
   public:
      Square() : Shape() {length = 0;} //declaring and initializing derived class constructor
      int get_Area(){ return (length * length); }
};

int main(void) {
   Square sq; //making object of child class Square
   sq.setlength(5); //setting length equal to 5
   // Print the area of the object.
   cout << "Total area of square is: " << sq.get_Area() << endl;

   return 0;
}
```



As you can see in the example above,

- The `shape` class is the *parent* class whereas the `Square` class is the *child* class *derived* from it.

- In our *child* class `Square`, we use *members* from the *parent* class such as
    - the `protected` `length` variable which gets *initialized* to **zero** in the *default* constructor.
    - `Length` also gets used in *child* class function `get_Area` to compute the *area* of the *square*.
- In `main` the `setlength` function which is a `public` *member* function of the *parent* class is accessible to the *child* class object `sq`
    - The **dot** operator is used to access `setlength` in the `main`.

In the next lesson, we will discuss the concept of *Polymorphism* in C++.