A Revision Example:

- **Consider the following program segment:**

```
Counter::Counter() // constructor
{
count = 0;
} int Counter::getCount() // get current count
{
return count;
} void Counter::increaseBy(int x)// add x to the count
{
count += x;
}
```

Again, the first of these functions has the same name as the class itself. This is a special member function called a *constructor*

- **As discussed earlier, a constructor's job is to initialize the values of the class's member variables.**

- **The function getCount is commonly referred to as a "*getter*" function. *Such functions provide access to the private members of the class*.**

- **The following is an example how we might use the simple class given above.**

- **We declare a new object of type Counter, called ctr. This implicitly invokes the class's constructor, and thus initializes the counter's value to 0. *To invoke one of the member functions, we use the notation ctr.function name()*.**

```
Counter ctr; // an instance of Counter
cout << ctr.getCount() << endl;//prints the initial value (0)
ctr.increaseBy(3); // increase by 3
cout << ctr.getCount() << endl; // prints 3
ctr.increaseBy(5); // increase by 5
cout << ctr.getCount() << endl; // prints 8
```

Note that, we can have two types of Member Functions

getter function: uses but does not modify a member variable

ex: getSide

setter function: modifies a member variable

ex: setSide

Inheritance

- **Inheritance is the process of *creating* new classes, called *<u>derived classes</u>* (child), from existing or *base* classes (parent). *The base class is unchanged by this process*.**

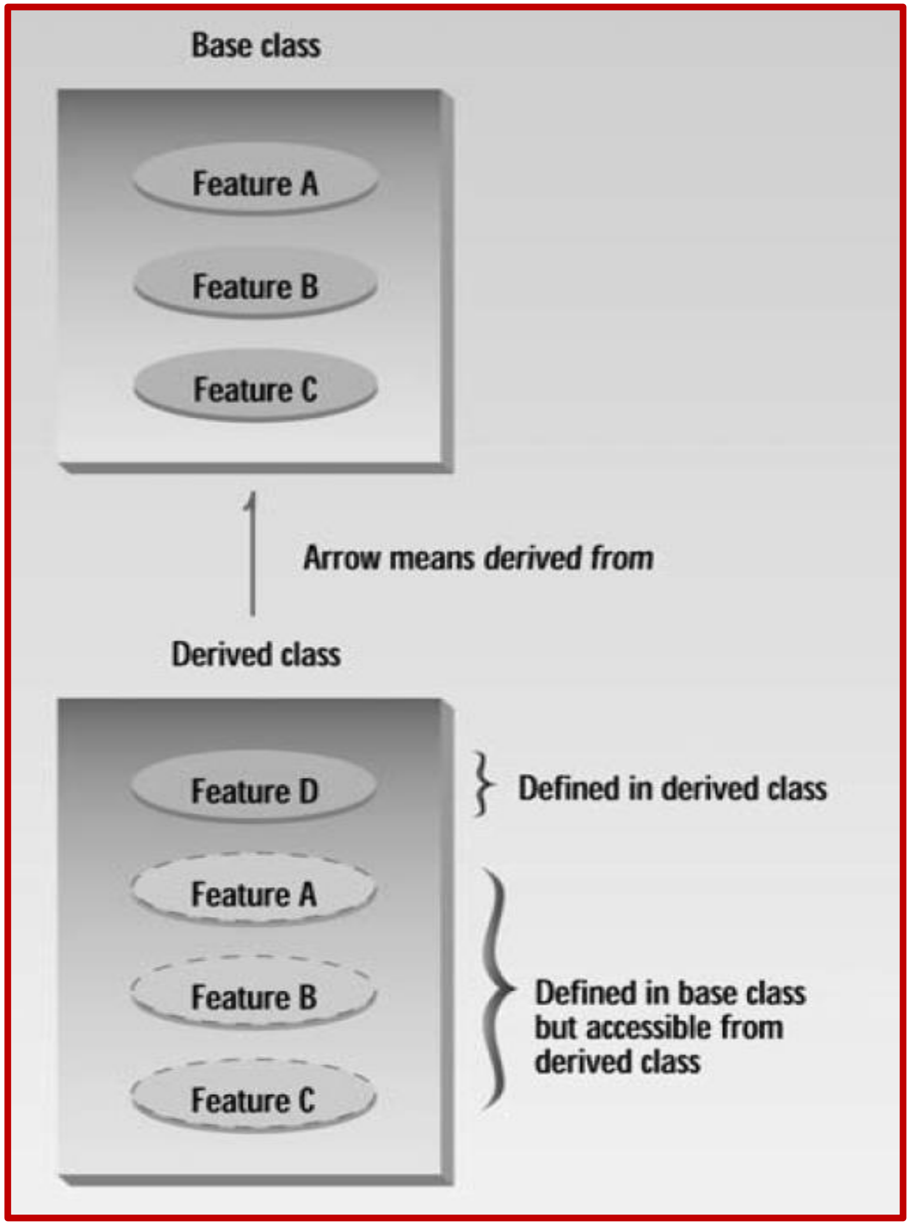
- **In other words, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. *Inheritance is a form of software reuse in which you create a class that absorbs an existing class's capabilities, then customizes or enhances them.***

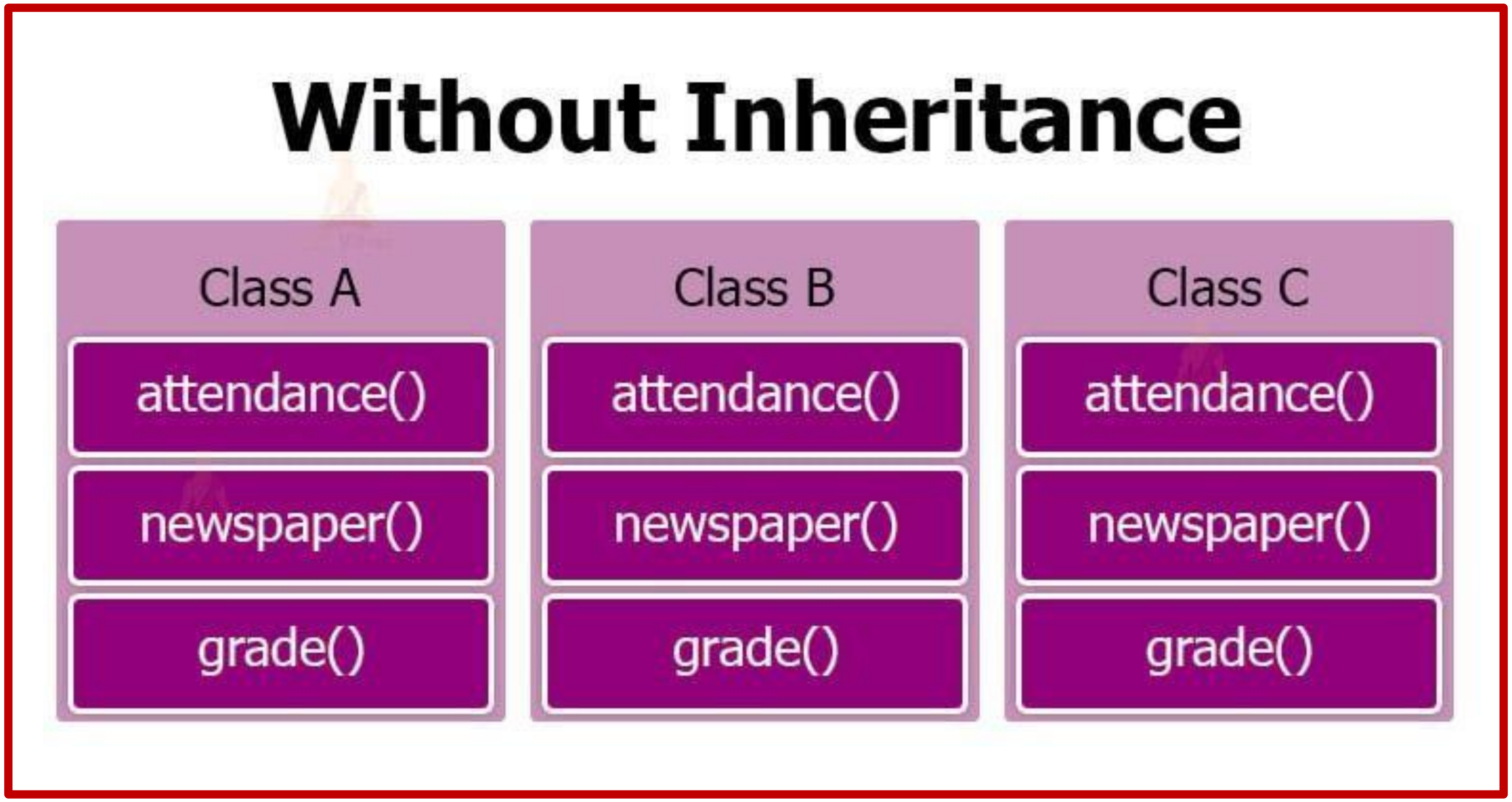
- *This means, you can reuse, extend or modify the attributes and behaviors which are defined in other class.*

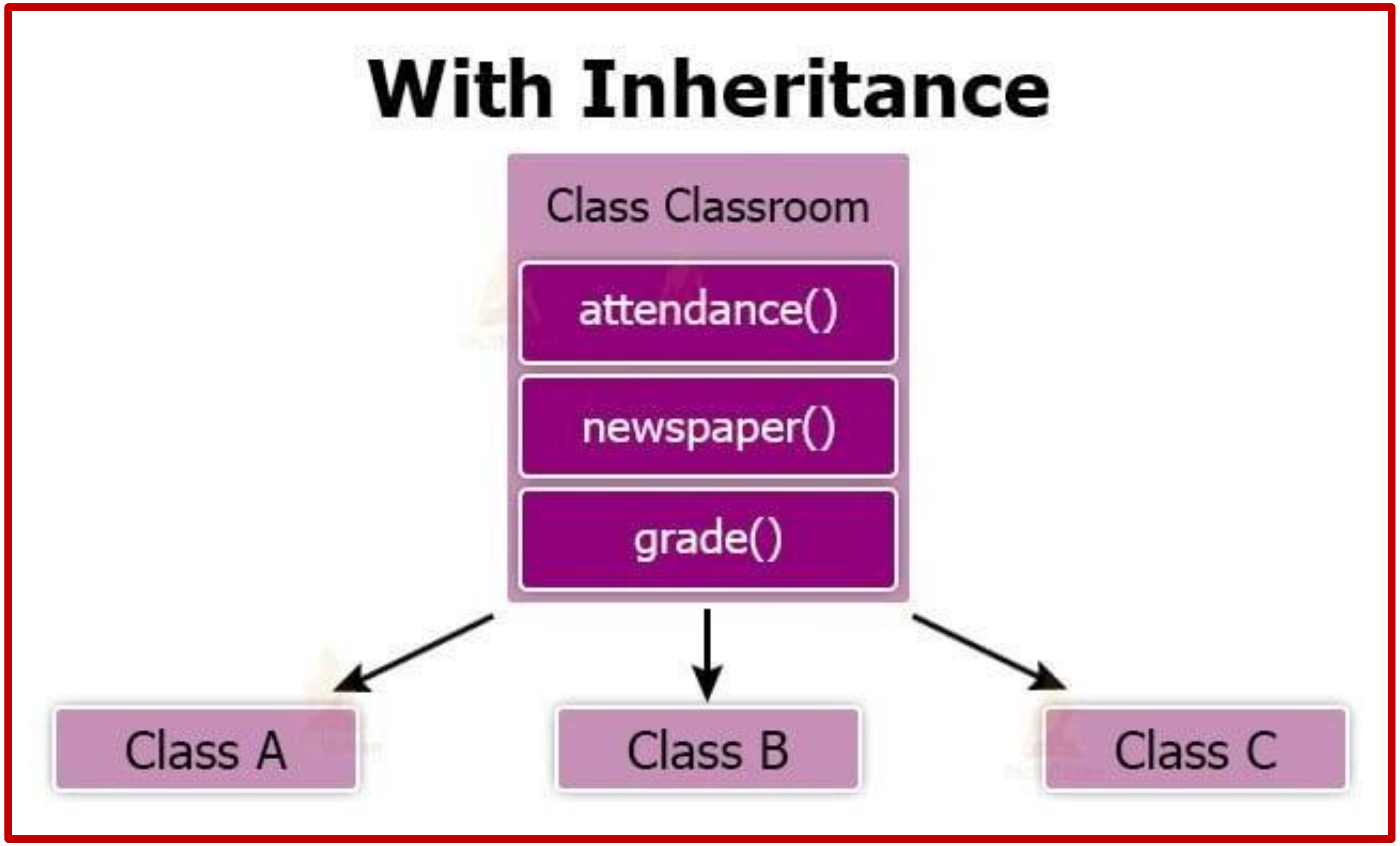
- **Reusing existing code saves time and money and increases a program’s reliability.**

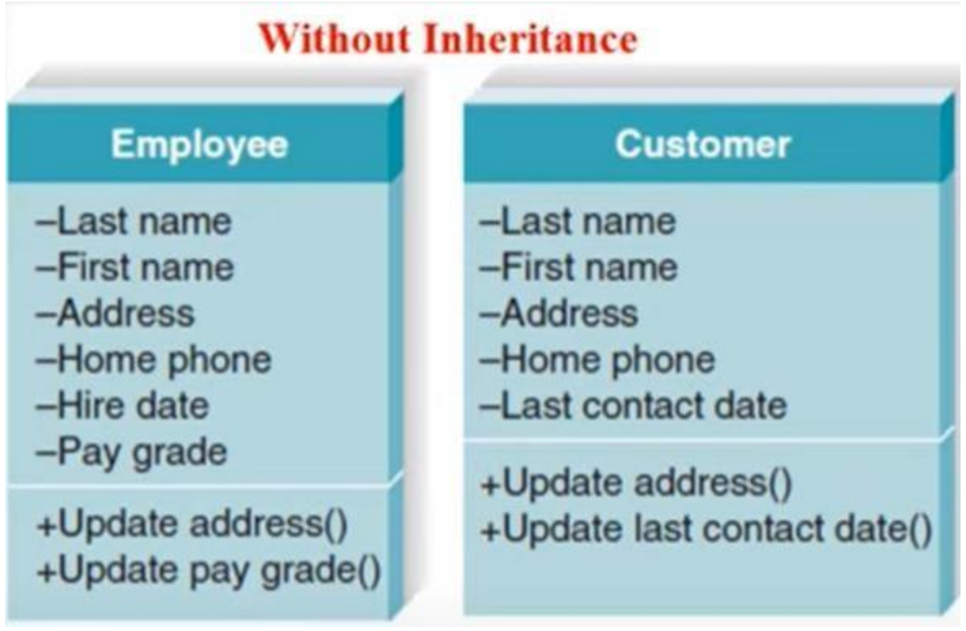
- **An important result of reusability is the ease of distributing class libraries.** ***A programmer can use a class created by another person or company, and, without modifying it, derive other classes from it that are suited to particular situations.***

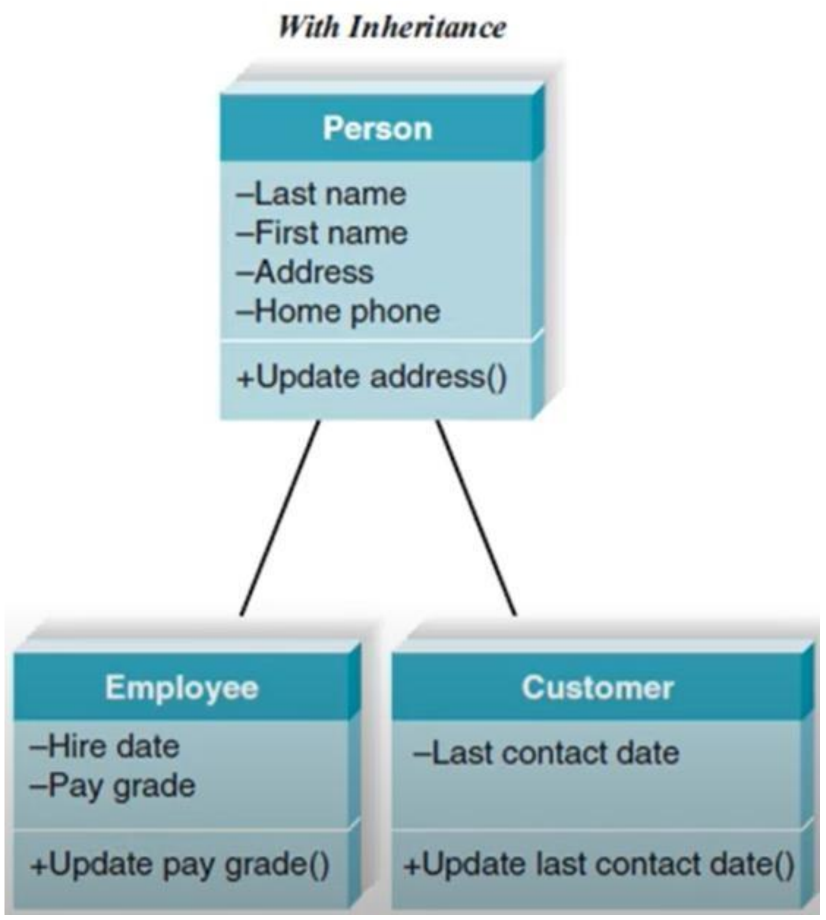
- **The inheritance relationship can be illustrated as follows:**

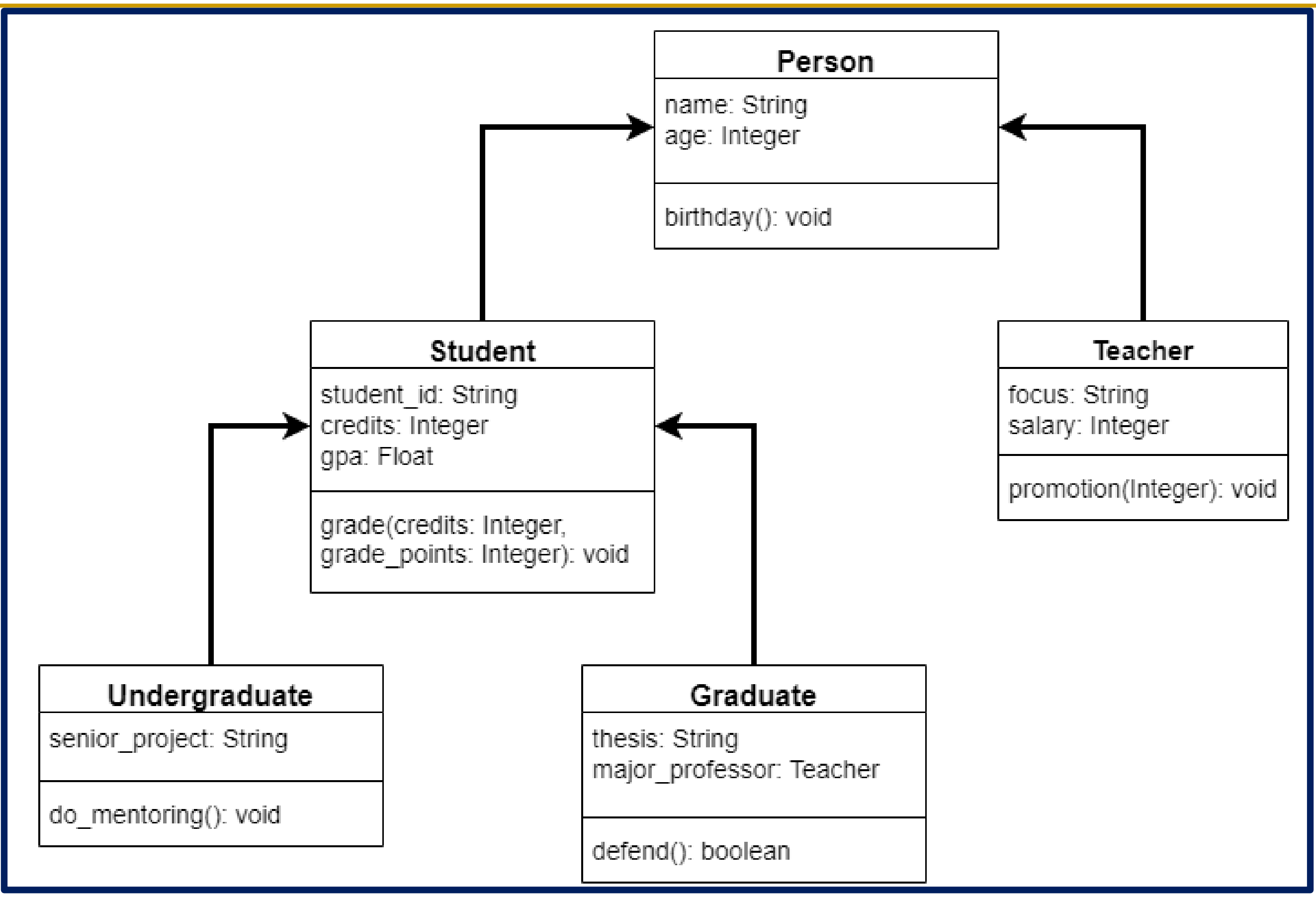

Base class
Feature A
Feature B
Feature C
Arrow means derived from
Derived class
Feature D
Defined in derived class
Feature A
Feature B
Feature C
Defined in base class but accessible from derived class


Without Inheritance
Class A
attendance()
newspaper()
grade()
Class B
attendance()
newspaper()
grade()
Class C
attendance()
newspaper()
grade()


With Inheritance
Class Classroom
attendance()
newspaper()
grade()
Class A
Class B
Class C


Without Inheritance
Employee
–Last name
–First name
–Address
–Home phone
–Hire date
–Pay grade
+Update address()
+Update pay grade()
Customer
–Last name
–First name
–Address
–Home phone
–Last contact date
+Update address()
+Update last contact date()


With Inheritance
Person
–Last name
–First name
–Address
–Home phone
+Update address()
Employee
–Hire date
–Pay grade
+Update pay grade()
Customer
–Last contact date
+Update last contact date()

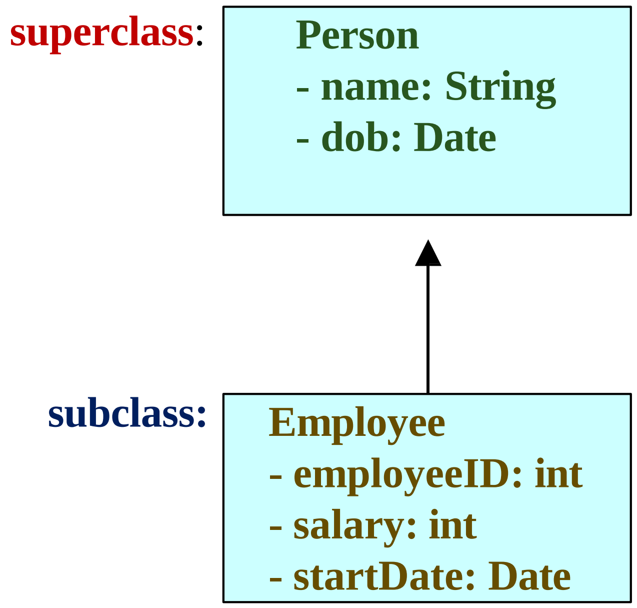


In a UML diagram, we can show the inheritance relationship using an open arrow between the classes. we can create another set of classes to represent Graduate and Undergraduate students, and have them inherit from the Student class. ***There is no limit to how many layers of inheritance we can create in our programs.***

- **Again, a generic class is also known as a base class, parent class, or superclass.**
- **Any class that extends a base class need not give new implementations for the general functions, for it inherits them.**
- **It should only define those functions that are specialized for this particular class.**
- **Such a class is called a *derived class*, *child class*, or *subclass*.**

- A class can be defined as a "subclass" of another class.
  - The subclass inherits all data attributes of its superclass
  - The subclass inherits all methods of its superclass
  - The subclass inherits all associations of its superclass

- The subclass can:
  - Add new functionality
  - Use inherited functionality



- To illustrate the above notations, suppose that we are writing a program to deal with people at a university.

```
class Person { // Person (base class)
private:
string name; // name
string idNum; // university ID number
public:
// . . .
void print(); // print information
string getName(); // retrieve name
};
```

- Suppose we next wish to define a student object. We can derive our class <u>Student</u> from class <u>Person</u> as follows:

```
class Student : public Person { // Student (derived from Person)
private:
string major; // major subject
int gradYear; // graduation year
public:
// . . .
void print(); // print information
void changeMajor(const string& newMajor); // change major
};
```

For creating a sub-class that is inherited from the base class we have to follow the below syntax. Derived Classes: A Derived class is defined as the class derived from the base class.

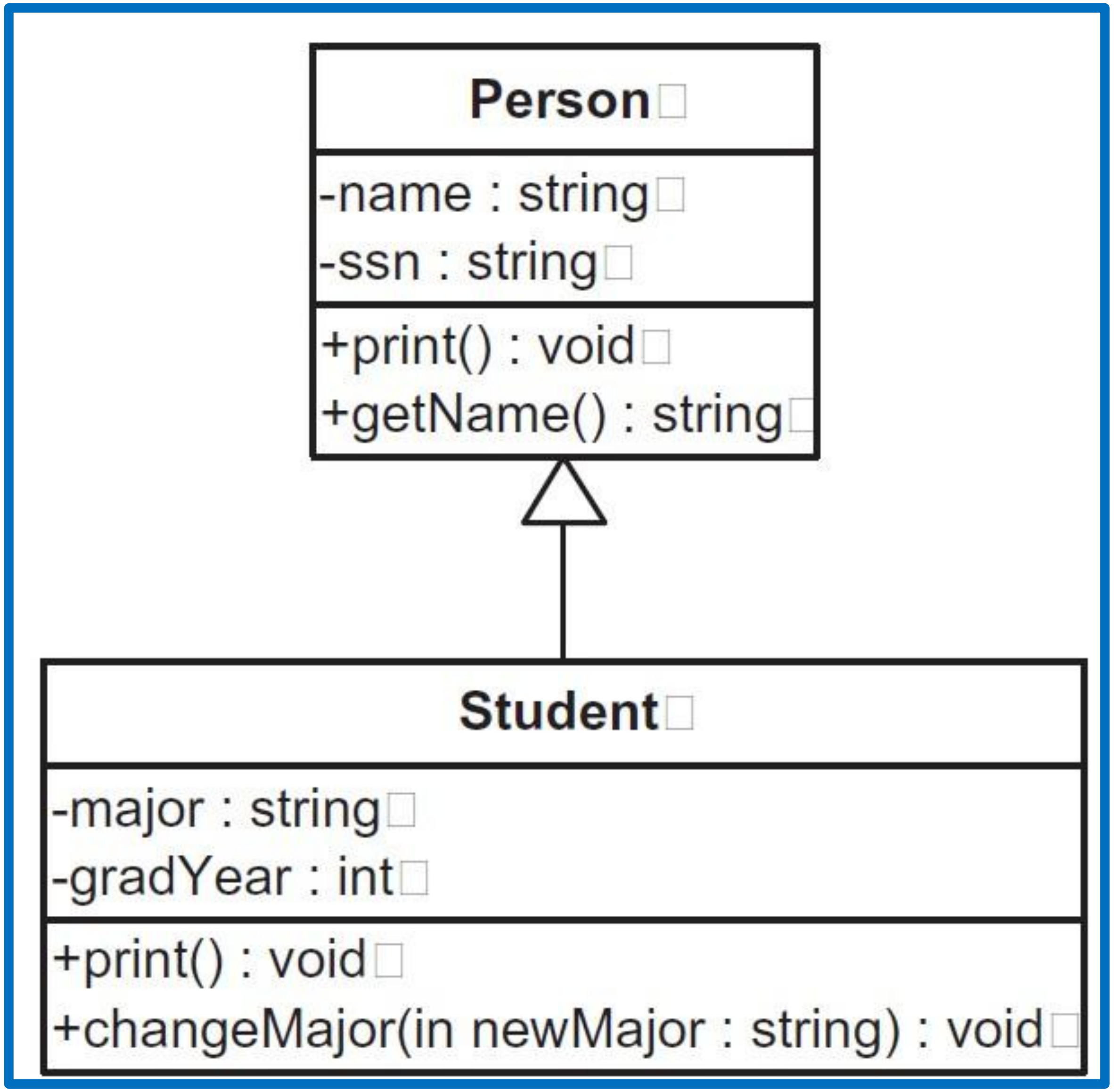
```
class <derived_class_name> : <access - specifier> <base_class_name>
{
//body
}
```

- **The “public Person” phrase indicates that the Student is derived from the Person class. *The keyword “public” specifies public inheritance*.**

- **When we derive classes in this way, there is an implied “is a” relationship between them. In this case, a Student “is a” Person.**

- **In particular, a Student object inherits all the member data and member functions of class Person in addition to providing its own members.**

- **The relationship between these two classes is shown graphically in the following class inheritance diagram.**


Person
-name : string
-ssn : string
+print() : void
+getName() : string
Student
-major : string
-gradYear : int
+print() : void
+changeMajor(in newMajor : string) : void

- An object of type ***Person*** can access the public members of Person.

- ***An object of type Student can access the public members of both classes*.**

- **If a Student object invokes the shared print function, it will use its own version by default.**

- **We use the *class scope operator* (::) to specify which class's function is used, as in Person:: print and Student::print. Note that an object of type Person cannot access members of the base type, and thus it is not possible for a Person object to invoke the changeMajor function of class Student.**

```
Person person("Ahmed", "12-345");// declare a Person
Student student("Ali", "98-764", "Math", 2012);//      declare a Student
cout << student.getName() << endl; // invokes Person::getName()
person.print(); // invokes Person::print()
student.print(); // invokes Student::print()
person.changeMajor("Physics"); // ERROR!
student.changeMajor("English"); // okay
```

- When we say derived class inherits the base class, it means, the derived class inherits all the properties of the base class, without changing the properties of base class and may add new features to its own.

- These new features in the derived class will not affect the base class.

- The derived class is the specialized class for the base class.

- Again, sub class: The class that inherits properties from another class is called Subclass or Derived Class.

- Super Class: The class whose properties are inherited by a subclass is called Base Class or Superclass.

Ex. Consider the following example:

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

class Person {
    int id;
    char name[100];

public:
    void set_p()
    {
        cout << "Enter the Id:";
        cin >> id;
        cout << "Enter the Name:";
        cin >> name;
    }

    void display_p()
    {
      cout << endl << "Id: " << id << "\nName: " << name << endl;
    }
};
```

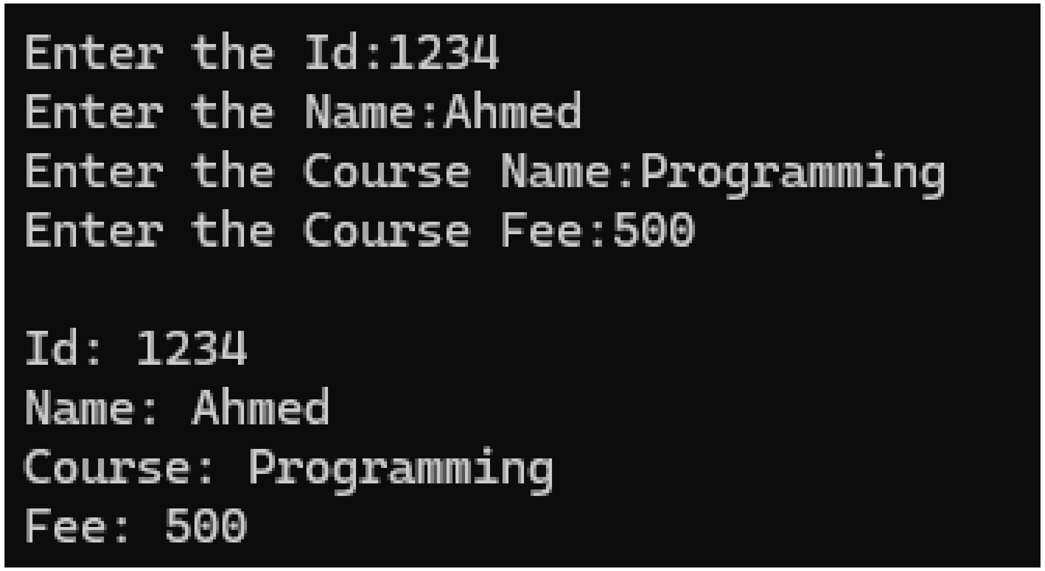
Continued

```
class Student : private Person {
    char course[50];
    int fee;

public:
    void set_s()
    {
        set_p();
        cout << "Enter the Course Name:";
        cin >> course;
        cout << "Enter the Course Fee:";
        cin >> fee;
    }

    void display_s()
    {
        display_p();
        cout << "Course: " << course << "\nFee: " << fee << endl;
    }
};
```

Continued

```
int main()
{
    Student s;
    s.set_s();
    s.display_s();
    return 0;
}
```

The output of this program will be as follows:



```
Enter the Id:1234
Enter the Name:Ahmed
Enter the Course Name:Programming
Enter the Course Fee:500

Id: 1234
Name: Ahmed
Course: Programming
Fee: 500
```