SLIIT
Discover Your Future

Object Oriented Concepts

Lecture-12

Implementation of Inheritance using C++

SLIIT
FACULTY OF COMPUTING

Learning Outcomes

- At the end of the lecture, students should be able to
  - Implement Inheritance, virtual functions
  - Understand and apply polymorphism

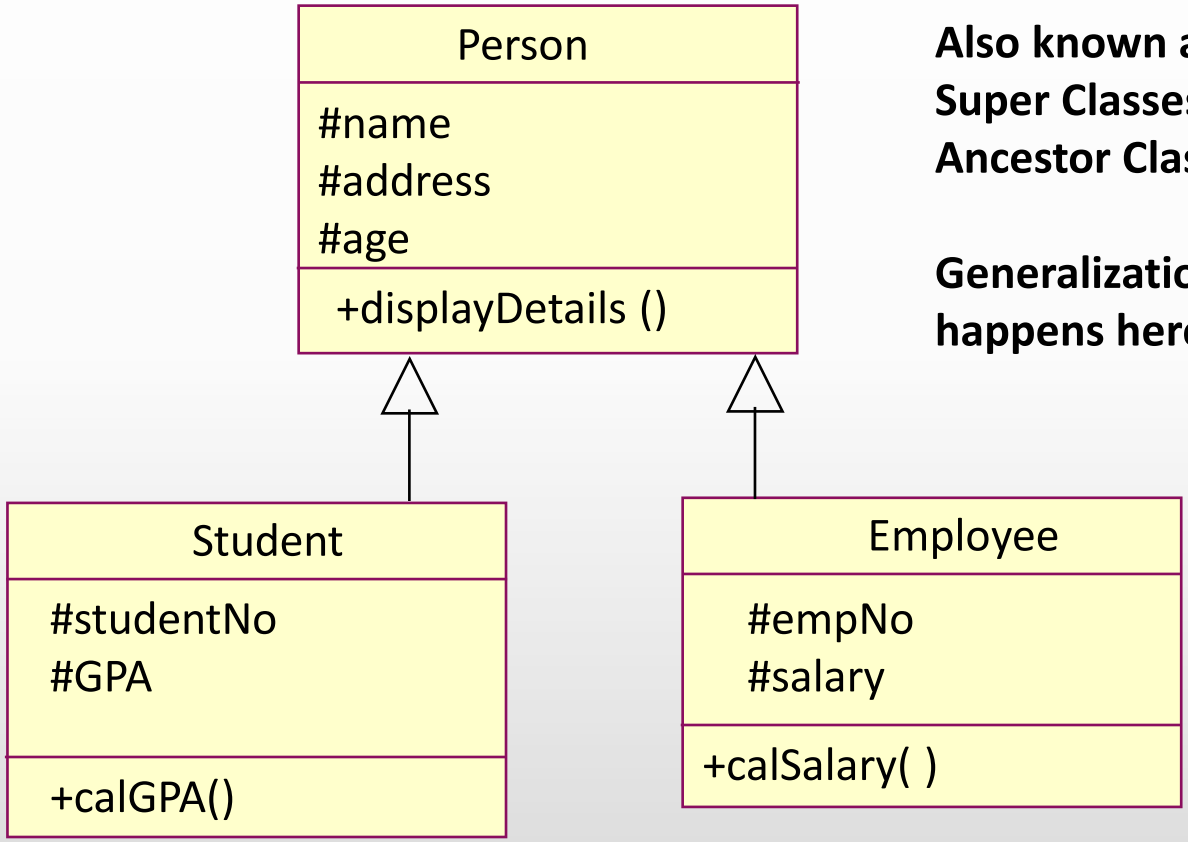
SLIIT
Discover Your Future

Inheritance

- Inheritance is the process by which one object can acquire the properties of another object.
- Instead of writing completely new data members and member functions, you can specify that the new class should inherit the members of existing class.
- The existing class is called the base class and the new class is called the derived class.

Base class and derived class

Base class



Also known as
Super Classes
Ancestor Classes

Generalization
happens here

Derived
class

Also known as, Sub Classes, Descendant Classes
Specialization happens here

SLIIT
Discover Your Future

C++ code

Base Class

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

class Person{
protected :
  char name[20];
  char address[20];
  int age;
public:
   Person(){};
   void display() {
      cout << "this is person class" << endl;
   }
   void displayDetails();
};
```

Derived Classes

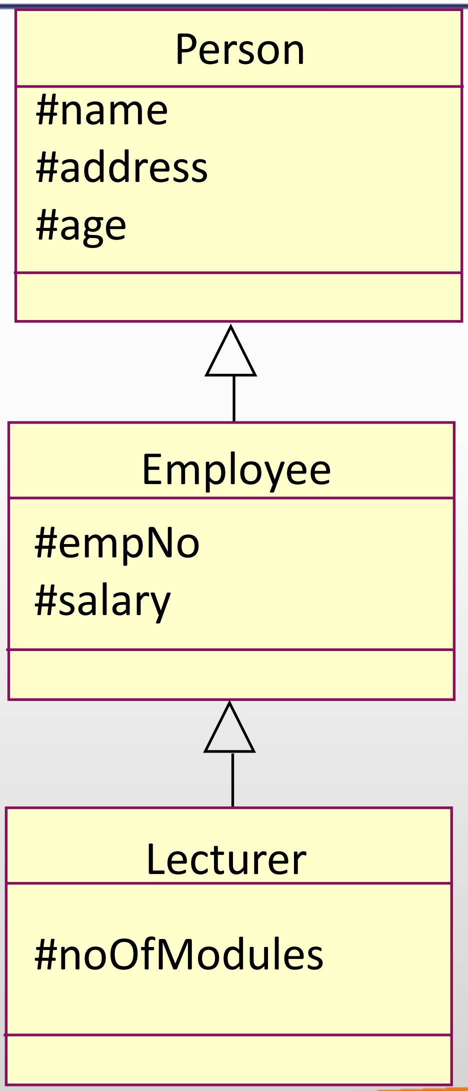
```
class Student : public Person{
protected :
   int studentNo;
   double GPA;
public:
   Student(){};
    void display() {
      cout << "this is student class. derived class
from person" << endl;}
   void calGPA();
};
class Employee : public Person{
protected :
   int empNo;
   double salary;
public:
   Employee(){};
    void display() {
              cout << "this is employee class.
                  Derived class from person"<< endl; }
           void calSalary();
};
```

Sample Code

Access Rules

	Public +	Protected #	Private +
Same class	Yes	Yes	Yes
Derived class	Access	Yes	No
Outside class	Yes	No	No

Multilevel Inheritance



SLIIT Discover Your Future

C++ code

Base Class

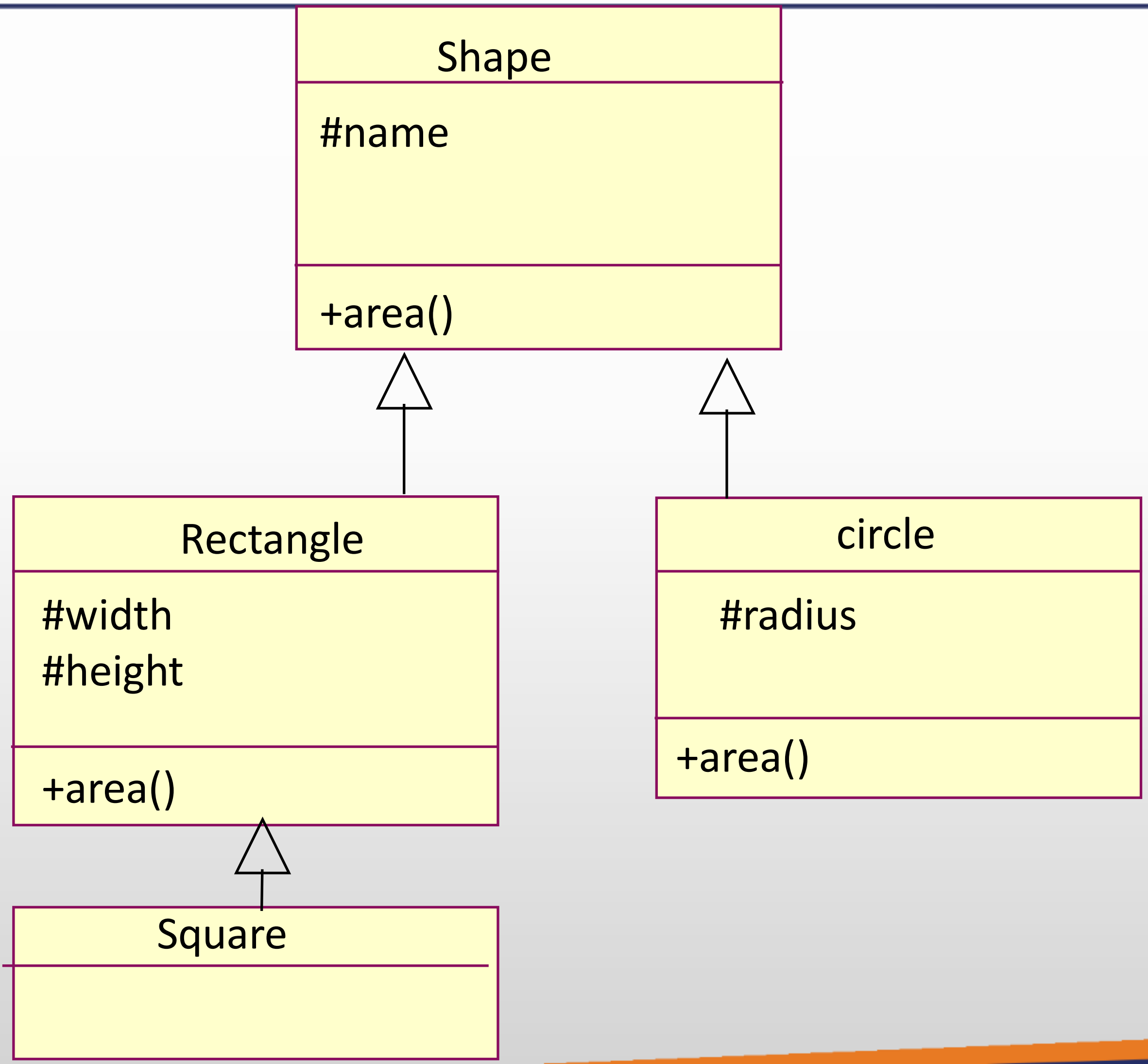
```
# include <iostream>
using namespace std;

class Person{
protected :
  char name[20];
  char address[20];
  int age;
public:
   Person(){};
   void display() {
      cout << "this is person class" << endl;
   }
   void displayDetails();
};
```

Derived Classes

```
class Employee : public Person{
   protected :
      int empNo;
      double salary;
   public:
      Employee(){};
      void display() {
                cout << "this is employee class.
                    Derived class from person"<< endl; }
             void calSalary();
};
class Lecturer: public  Employee{
     protected :
             int noOfModules;
     public :
             Lecturer( ){};
             void display(){
                cout << "this is employee class.
                    Derived class from person" << endl;
             }
};
```

Sample Code

Handling constructors



Default Constructors

```
class Shape
{
    protected:
        char name[20];
    public:
        Shape() {
            cout << "Shape
called" << endl
        }
};
```

```
class Rectangle : public Shape
{
    protected:
        int length, width
    public:
        Rectangle() {
            cout << "Rectangle
called" << endl
        }
};
```

The compiler will insert code in the derived class (Rectangle) class constructor to call the Shape class constructor. This is the first instruction that will be executed in the Rectangle class constructor.

Rectangle rec; // will produce the following output

Shape Called

Rectangle Called

Sample Code

Overloaded Constructors

Base class

```
class Shape
{
     protected:
        char name[20];
     public:
        Shape();
        Shape ( char tname[])
        {
              name = tname;
        }
};
```

Derived class

```
class Rectangle: public Shape{
     protected:
          int length;
          int width;
          public:
     Rectangle (char tname[],
int l, int w) : Shape ( tname)
          {
               length = l;
               width = w;
          }

};
```

Sample Code

Overriding methods

Base class

```
class Shape
{
     protected:
         char name[20];
     public:
         Shape();
         …
         int area() {
              return 0;
          }
};
```

Derived class

```
class Rectangle: public Shape{
          protected:
             int length;
             int width;
          public :
          Rectangle ();
          ….
         int area() {
             return length * width;
         }
};
```

Sample Code

Overriding

- In the previous example we saw that the Rectangle class redefines the area method()

int area()

- This definition is exactly the same as in the Shape class. This is called overriding.
- If a sub class has a different behavior of a method we can override it and redefine the code associated.

- Please Note that in overloading there is a difference in parameters and overloading usually happens within the same class.

Overriding

- Consider the following code

```
Shape sh(“myshape”);
Rectangle rec(“whiteboard”, 10,5);
cout << sh.area(); // will produce 0, Shape::area() called
cout << rec.area(); // will produce 50, Rectangle::area() called
```

Situations where overriding doesn’t work properly in C++ and how to fix it

```
Shape *sh;
sh = new Rectangle(“whiteboard”, 10, 5);
cout << sh->area();
```

- // produces 0, Shape::area() called

- In the above code a Rectangle type object is created in the second line, but when we run the code the Shape classes area() function is called

Virtual functions as a fix

- We can define the function that we are overriding as a virtual function. This enables dynamic binding where the overriden methods are called correctly at runtime.
- A virtual function is a member function that is declared with in the base class and redefined by a derived class.
- The function in the base class must precede the keyword **virtual**
- **Virtual functions** support dynamic polymorphism
- e.g. we have to add the word virtual to the area method of the Shape class and everything will work properly.

```
virtual int area() {
    return 0;
}
```

Sample Code

SLIIT
Discover Your Future

Polymorphism

- Greek meaning *“having multiple forms“*
- Ability to assign a different meaning or usage to something in different contexts
- Specifically, to allow an entity such as a variable, a function, or an object to have more than one form
- Overriding is a type of polymorphism, here we generally expect dynamic binding to work as well (use of virtual functions)

An example

- Consider the request (analogues to a method) *“please cut this in half“* taking many forms

For a cake:

- Use a knife
- Apply gentle pressure

For a cloth:

- Use a pair of scissors
- Move fingers in a cutting motion

"please cut this in half"

- Imagine this task is automated…
- Without polymorphism
  - Need to tell the computer how to proceed for each situation
- With polymorphism
  - Just tell *please cut this in half*
  - The computer will handle the rest!

Polymorphism

- Polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.
- C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Abstract Classes

- In some situations we want to prevent the creation of objects of a given class. e.g. We may want to restrict the creation of Shape type objects.
- In C++ we can create an abstract class by including at least one pure virtual method
- We can make a method a pure virtual method by assigning zero to the virtual method.
- e.g. in the Shape Class we can do the following

```
virtual int area() = 0;
```

- You cannot declare an instance (object) of an abstract base class; you can use it only as a base class when declaring other classes.

SLIIT
Discover Your Future

Abstract Classes

Base class

```
class Shape
{
    protected:
        char name[20];
    public:
        Shape();
        …
        virtual int area() = 0;
};
```

Derived class

```
class Rectangle: public Shape{
        protected:
            int length;
            int width;
        public :
        Rectangle ();
        ….
       int area() {
           return length * width;
       }
};
```

Now the following code will generate a compilation error

Shape myshape;

However we can do the following

Shape *shp;

Shp = new Rectangle();

SLIIT
Discover Your Future

Example – Polymorphism

```
class Animal {
protected:
        char name[20];
public:
        Animal() {}
        Animal(char tname[]) {
                strcpy(name, tname);
        }
        virtual void speak() {}
        void song() {
                cout << name << "'s Song " << endl;
                speak();
                cout << "la la la la" << endl;
                speak();
                cout << "la la la la" << endl;
                speak();
        }
```

Sample Code

Animal Example

```
class Cat : public Animal {
public:
        Cat() {}
        Cat(char tname[]) : Animal(tname) { }
        void speak() {
                cout << "Meow... Meow..." << endl;
        }
};

class Dog : public Animal {
public:
        Dog() {}
        Dog(char tname[]): Animal(tname) { }
        void speak() {
                cout << "Bow... Bow..." << endl;
        }
};
```

Animal Example

```
class Cow: public Animal {
public:
        Cow() {}
        Cow(char tname[]) : Animal(tname) { }
        void speak() {
                cout << "Moo... Moo..." << endl;
        }
};
int main()
{
        Animal *ani[4];
        ani[0] = new Cat("Micky the Cat");
        ani[1] = new Dog("Rover the Dog");
        ani[2] = new Cow("roo the Cow");
        ani[3] = new Animal("no name");
        for (int r=0;r<4; r++)
                ani[r]->song();

        char ch;
        cin >> ch;
        return 0;
}
```