Lecture 10_11

Abstract Classes

- An abstract class is one that is not used to create objects
- It is designed only to act as base class
- abstract class and interface interchangeably.
- **A class with pure virtual function is known as abstract class**
- For example the following function is a pure virtual function:

virtual void fun() = 0;

A pure virtual function is marked with a virtual keyword and has = 0 after its signature. You can call this function an abstract function as it has no body.

Why we need an abstract class?

- Creating a class Animal with two functions sound() and sleeping().

- Now, we know that animal sounds are different cat says “meow”, dog says “woof”.
- So what implementation do I give in Animal class for the function sound(), the only and correct way of doing this would be making this function pure abstract so that I need not give implementation in Animal class but all the classes that inherits Animal class must give implementation to this function.
- This we ensure that all the Animals have sound but they have their unique sound.

```
#include<iostream>
using namespace std;
class Animal{
public:
  //Pure Virtual Function
  virtual void sound() = 0;

  //Normal member Function
  void sleeping() {
    cout<<"Sleeping";
  }
};
class Dog: public Animal{
public:
  void sound() {
    cout<<"Woof"<<endl;
  }
};
int main(){
  Dog obj;
  obj.sound();
  obj.sleeping();
  return 0;
}
```

Rules of Abstract Class

1) Any class that has a pure virtual function is an abstract class.

2) We cannot create the instance of abstract class.

3) We can create pointer and reference of base abstract class points to the instance of child class. For example, this is valid:

```
Animal *obj = new Dog();
obj->sound();
```

4) Abstract class can have constructors.

5) If the derived class does not implement the pure virtual function of parent class then the derived class becomes abstract.

```
 #include<iostream>
using namespace std;
class Animal{
public:
   //Pure Virtual Function
   virtual void sound() = 0;};
   class Dog:public Animal
   {public:
   void sound()
   {cout<<"meow";
   }};
 int main()
{ Animal *obj = new Dog();
obj->sound();
    return 0; }
```

Ambiguity Resolution in Inheritance

- Ambiguity occurs when a function with the same name appears in more than one base class or appears in both base and derived class.

```
class M
{
   public:
      void display() {   cout << "\n class M" ;   }
};
class N
{
   public:
      void display() {   cout << "\n class N" ;   }
};
class P : public M, public N
```

Ambiguity Resolution in Inheritance

- The problem can be solved by defining a named instance within the derived class using the class resolution operator with the function name.

```
Class P : public M, public N
{
    public:
        void display()
        {                                   // overrides display()
                M :: display()              // of M and N
        }
};
```

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

class Base1
{
public:
    char c;
};

class Base2
{
public:
    int c;
};

class Derived: public Base1, public Base2
{
public:
    void show()  { cout <<Base2:: c; }
};

int main(void)
{     Derived d;
    d.show();     return 0; }
```

Initialization List

```
#include <iostream>
using namespace std;
class exam
{    int a, b, c;
public:
exam( int x, int y, int z):a(x),b(y),c(z){}//Initialization list a=x;b=y;c=z;
void disp(){cout<<a<<b<<c;}};
int main()
{    exam E1(1,2,3);    E1.disp();    return 0;}
```

Constructor

- object of a class created -----→ default constructor of that class is invoked automatically to initialize the members of the class.
- As long as no base class constructor takes any argument the derived class need not contain a constructor function
- If any base class contains a constructor with more than one arguments,it mandatory for derived class to have constructor and pass arguments to base class constructors
- When both base class and derived class contain constructor then base class constructor executed first and then the derived class

- Derived class takes the responsibility of supplying initial value to the base class,when a derived class object id declared we supply all the initial values
- **Why the base class's constructor is called on creating an object of derived class?**
- when we create an object of derived class, all of the members of derived class must be initialized but the inherited members in derived class can only be initialized by the base class's constructor as the definition of these members exists in base class only. This is why the constructor of **base class is called first to initialize all the inherited members.**

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

// base class
class Parent
{
    public:
        // base class constructor
    Parent()
    {       cout << "Inside base class" << endl;
    } };
  // sub class
class Child : public Parent
{
    public:
        //sub class constructor
    Child()
    {       cout << "Inside sub class" << endl;   } };
  // main function
int main() {
    // creating object of sub class
    Child obj;

    return 0;
}
```

Output

```
Inside base class
Inside derived class
```

```
// C++ program to show the order of constructor calls
// in Multiple Inheritance
  #include <iostream>
using namespace std;
  // first base class
class Parent1
{           public:
          // first base class's Constructor
    Parent1()
    {         cout << "Inside first base class" << endl;
    } };
  // second base class
class Parent2
{     public:
          // second base class's Constructor
    Parent2()
    {         cout << "Inside second base class" << endl;     } };
// child class inherits Parent1 and Parent2
class Child : public Parent1, public Parent2
{
    public:

    // child class's Constructor
    Child()
    {
        cout << "Inside child class" << endl;
    }
};

// main function
int main() {

    // creating object of class Child
    Child obj1;
    return 0;
      }
```

output

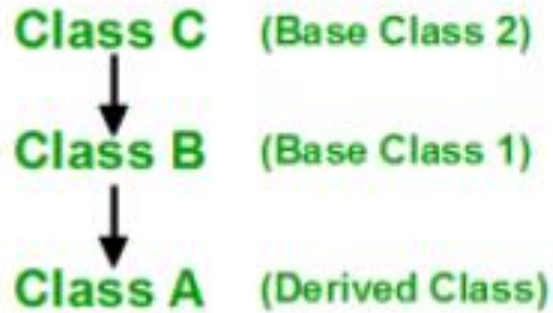
Inside first base class

Inside second base class

Inside child class

Order of constructor and Destructor call for a given order of Inheritance

Order of Inheritance



Order of Constructor Call

1. C() (Class C's Constructor)
2. B() (Class B's Constructor)
3. A() (Class A's Constructor)

Order of Destructor Call

1. ~A() (Class A's Destructor)
2. ~B() (Class B's Destructor)
3. ~C() (Class C's Destructor)

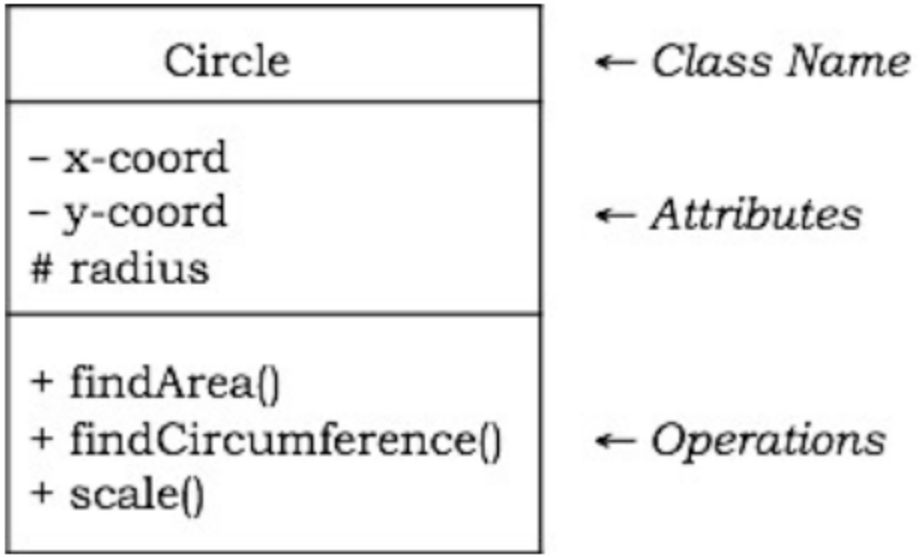
```
class alpha
{    int x;
  public:
   alpha(){cout<<"default base";}
  alpha(int i){x=i;cout<<"parameterized base"<<endl;}
};
class gamma:public alpha
{
    int m, n;
   public:
  gamma(){cout<<"default derived"<<endl;}
  gamma(int a, int b, int c):alpha(a)
   {       m=b;       n=c;       cout<<"parameterized derived"<<endl;    }
};
int main()
   { gamma g1;    gamma g(6,3,2);       return 0;}
```

Output
Default base
Default derived
parameterized base
parameterized derived

```
class A
{   int x;
  public:
  A(){}
  A(int i)
{x=i; cout<<x<<endl;}
 };
class B
{ int y;
 public:
 B(){}
 B(int j)
{y=j; cout<<y<<endl;}
};
```

```
class C: public B, public A
{int z;
public:
C(){}
C(int a, int b, int c):A(a),B(b)
 {    z=c;cout<<z<<endl;}
};
int main()
{    C c1(4,5,6);    C c2;    return
0;}
```

Representation of class diagram

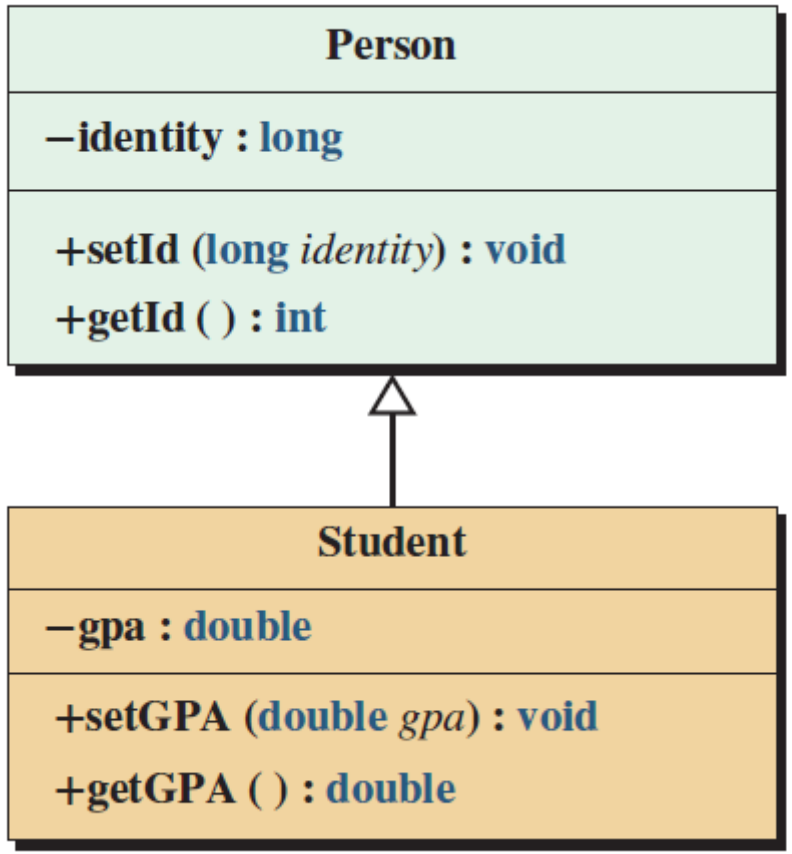


A class is represented by a rectangle having three sections −

- the top section containing the name of the class
- the middle section containing class attributes
- the bottom section representing operations of the class

The visibility of the attributes and operations can be represented in the following ways −

- **Public** − A public member is visible from anywhere in the system. In class diagram, it is prefixed by the symbol '+'.
- **Private** − A private member is visible only from within the class. It cannot be accessed from outside the class. A private member is prefixed by the symbol '−'.
- **Protected** − A protected member is visible from within the class and from the subclasses inherited from this class, but not from outside. It is prefixed by the symbol '#'.



Notes:

The type of data members and member functions is shown after the member name separated by a colon.

The minus signs define the visibility of data members as private; the plus signs define the visibility of the member fuctions as public.