object-oriented programming (OOP)

KIAN _ ACADEMY

Abstract Class :

An abstract class : is a class that is designed to be used as a base class.

An abstract class : contains at least one pure virtual function.

You declare a pure virtual function by using a pure (= 0) in the declaration of a virtual member function in the class declaration.

Example :

```
class A {
public:
  virtual void f() = 0;
};
********************************
```

Example :

```
#include<iostream>
using namespace std;
class Base
{
protected:
    int x;
public:
    virtual void fun(int x)=0;
};
```

```
class Derived: public Base
{
public:
    void fun(int x)
    {
        cout << "x is : "<<x<<endl;
    }
};
```

```
int main()
{
    Derived d;
    d.fun(5);
    return 0;
}
```

- **Note 1:**

From the example above.....

A class is abstract if it has at least one pure virtual function.

Another Example From the last session

```
#include<iostream>
using namespace std;
class Base
{
public:
    virtual void show() = 0;
};
```

```
class Derived: public Base
{
public:
    void show()
    {
        cout << "In Derived \n";
    }
};
```

```
int main()
{
    Base *bp = new Derived();
    bp->show();
    return 0;
}
```

- **Note 2:**

From the example above.....

We can have pointers and references of abstract class type.

Another Example

```
#include<iostream>
using namespace std;
class Base
{
public:
    virtual void show() = 0;
};
```

```
class Derived : public Base
{ };

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

- **Note 3:**

From the example above.....

If we do not override the pure virtual function in derived class, then derived class also becomes abstract class.

Another Example

```
#include<iostream>
using namespace std;
class Base
{
protected:
   int x;
public:
   virtual void fun() = 0;
   Base(int i)
   {
      x = i;
      cout<<"Constructor of base 
called"<<endl;
   }
};
```

```
class Derived: public Base
{
   int y;
public:
   Derived(int i, int j):Base(i)
   {
      y = j;
   }
   void fun()
   {
      cout <<"x = "<<x<<" y = "<<y<<endl;
   }
};
```

```
int main()
{
    Derived d(4, 5);
    d.fun();

    Base *ptr=new Derived(6,7);
    ptr->fun();
    return 0;
}
```

- **Note 4:**

From the example above.....

An abstract class can have constructors.

last example in abstract class:

```
#include<iostream>
using namespace std;
class Base
{
protected:
    int x;
public:
    virtual void fun() = 0;
```

```
 Base(int i)
   {
       x = i;
       cout<<"Constructor of base
called"<<endl;
   }
};
```

```
int main()
{
    Base b(5);
    return 0;
}
```

- **Note 5:**

From the example above.....

Abstract classes cannot be instantiated

Final Classifier

used to prevent inheritance of class or struct .

If a class or struct is marked as final then it becomes non inheritable and it cannot be used as base class/struct .

example

```
#include <iostream>
using namespace std;
class Base
{
public:
    virtual void fun() final
    {
        cout << "fun() in
Base"<<endl;
    }
};
```

```
class Derived : public Base
{
    void fun()
    {
        cout << "fun() in Derived"<<endl;
    }
};
```

```
int main()
{
    Derived d;
    Base b = d;
    b.fun();
    return 0;
}
```

Another example

```
#include <iostream>
class Base final
{
};

class Derived : public Base
{
};
```

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

Friend Function :

- is defined as a function that can access private, protected and public members of a class.

- The friend function is declared using the friend keyword inside the body of the class.

example

```
#include<iostream>
using namespace std;
class Box
{
private:
   int length;
public:
   Box()
   {
      length=5;
   }
   friend int printLength (Box b);
};
```

```
int printLength (Box b)
{
   b.length +=10;
   return b.length;
}
int main ()
{
   Box b;
   cout <<"Length of box : "<<printLength(b)<<endl;
   return 0;
}
```

- **Note:**

the function is not inherited.

Another Example

```
#include<iostream>
using namespace std;
class B; //forward declaration.
class A
{
   int x;
public:
   void setdata(int i)
   {
      x=i;
   }
   friend void max (A, B); //friend
function.
};
```

```
class B
{
   int y;
public:
   void setdata(int i)
   {
      y=i;
   }
   friend void max (A, B);
};
void max(A a, B b)
{
```

```
   if (a.x >= b.y)
        cout<< a.x <<endl;
     else
        cout<< b.y <<endl;
}
int main ()
{
   A a;
   B b;
   a. setdata (10);
   b. setdata (20);
   max(a, b);
   return 0;
}
```

Friend Class:

A friend class can access private and protected members of other class in which it is declared as friend.

Another Example

```
#include <iostream>
using namespace std;
class Person
}
private:
    string name;
    float age;
    string address;
    string nationality;
public:
    Person()
}
 name="Ahmed";
        address="zagazig";
        age=20;
        nationality="Egypt";
{
void setName(string name)
}
        this->name=name;
{
    void setAge(float age)
}
        this->age=age;
{
void setAddress(string address)
}
        this->address=address;
{
    void setNationality(string nationality)
}
        this->nationality=nationality;
{
    string getName()
}
        return name;
{
```

```
string getAddress()
}
    return address;
{
  string getNationality()
}
    return nationality;
{
  float getAge()
}
    return age;
{
```

```
friend class printclass;
;{
class printclass
}
public:
  void print(Person b)
}
  cout<<"The Name is : "<<b.name<<endl<<"The Address is
"<<b.address<<endl<<"The Age is :
"<<b.age<<endl<<"The nationality is :
"<<b.nationality<<endl;
{
;{
```

```
int main()
}
  Person p;
  printclass pc;
  pc.print(p);
{
```

Note:

the class is not inherited