Abstract Class in C++

By – Mohammad Imran

Abstract Class

In C++, an abstract class is defined by having at least one pure virtual function, a function without a concrete definition. These classes are essential in object-oriented programming.

Sometimes implementation of all functions cannot be provided in a base class because we don’t know the implementation. Such a class is called an **abstract class**. For example, let Shape be a base class. We cannot provide the implementation of function **draw()** in Shape, but we know every derived class must have an implementation of **draw()**.

Abstract Class

A **pure virtual function** (or abstract function) in C++ is a virtual function for which we can have an implementation, But we must override that function in the derived class, otherwise, the derived class will also become an abstract class. A pure virtual function is declared by assigning 0 in the declaration.

A pure virtual function is implemented by classes that are derived from an Abstract class.

Abstract Class

Characteristics

- ✓ Abstract Classes must have at least one pure virtual function.
- ✓ Abstract Classes cannot be instantiated, but pointers and references of Abstract Class types can be created. You cannot create an object of an abstract class. Here is an example of a pointer to an abstract class.
- ✓ Abstract Classes are mainly used for **Upcasting**, which means its derived classes can use its interface.
- ✓ Classes that inherit the Abstract Class must implement all pure virtual functions. If they do not, those classes will also be treated as abstract classes.

By – Mohammad Imran

Abstract Class

Example - 1

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

By – Mohammad Imran

Abstract Class

Example - 1

```
class Derived : public Base
{
   int y;
  public:
   void fun()
   {
      cout << "fun() called" << endl;
   }
   void print(int a)
   {
      cout << getX(a);
   }
};
```

OUTPUT

```
fun() called
50
```

```
int main(void)
{
      Derived d;
      d.fun();
      d.print(50);
      return 0;
}
```

By – Mohammad Imran

Abstract Class

Some Interesting Facts

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

If a class is abstract so you cannot create a simple object of the class but you can create pointer object.

Abstract Class

Some Interesting Facts

```
#include <iostream>
using namespace std;
class Test
{
   int x;
  public:
   virtual void show() = 0;
   void display()
   {
      cout << "Hello World";
   }
};
```

OUTPUT

Abstract class cannot be instantiated

```
int main(void)
{
   Test t;
   return 0;
}
```

By – Mohammad Imran

Abstract Class

Some Interesting Facts

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

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

class Derived : public Base
{
};
```

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

By – Mohammad Imran

3. An abstract class can have constructors.

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

Abstract Class

Some Interesting Facts

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

By – Mohammad Imran

Abstract Class

Some Interesting Facts

```
int main(void)
{
   Derived d(4, 5);
   d.fun();
   Base* ptr = new Derived(6, 7);
   ptr->fun();
   return 0;
}
```

By – Mohammad Imran

Virtual Destructor

As we already know about destructor and calling sequence of constructor and destructor from base to derived class.

In late binding we create pointer object of base class and store address of derived class. So when we use new operator for memory allocation for the object pointer we need to delete those memory after use which is not possible to delete using normal destructor.

A virtual destructor is used to free up the memory space allocated by the derived class object or instance while deleting instances of the derived class using a base class pointer object.

By – Mohammad Imran

Virtual Destructor

Example - 3

```
#include<iostream>
using namespace std;
class Parent
{
  public:
   Parent()
   {
      cout<< "Constructor Parent\n";
   }
   ~Parent()
   {
      cout<< "Destructor Parent\n";
   }
};
```

By – Mohammad Imran

Virtual Destructor

Example - 3

```
class Child : public Parent
{
  public:
   Child()
   {
      cout << "Constructor Child\n" ;
   }
   ~Child()
   {
      cout << "Destructor Child\n" ;
   }
};
```

OUTPUT

```
Constructor Parent
Constructor Child
Destructor Parent
```

```
int main(void)
{
   Parent *bptr = new Child;
   delete bptr;
   return 0;
}
```

By – Mohammad Imran

Virtual Destructor

Example - 4

```
#include<iostream>
using namespace std;
class Parent
{
  public:
   Parent()
   {
      cout<< "Constructor Parent\n";
   }
   virtual ~Parent()
   {
      cout<< "Destructor Parent\n";
   }
};
```

By – Mohammad Imran

Virtual Destructor

Example - 4

```
class Child : public Parent
{
  public:
   Child()
   {
      cout << "Constructor Child\n" ;
   }
   ~Child()
   {
      cout << "Destructor Child\n" ;
   }
};
```

OUTPUT

```
Constructor Parent
Constructor Child
Destructor Child
Destructor Parent
```

```
int main(void)
{
   Parent *bptr = new Child;
   delete bptr;
   return 0;
}
```

By – Mohammad Imran

Order of Execution of Virtual Function

In C++, the order of execution of virtual functions is determined by the rules of dynamic dispatch and the structure of the class hierarchy. Here's a concise explanation of how it works:

Key Concepts

- ✓ **Dynamic Binding**: Virtual functions enable dynamic binding, meaning the function that gets executed is determined at runtime based on the actual object type, not the type of the pointer/reference.

By – Mohammad Imran

Order of Execution of Virtual Function

- ✓ **Base and Derived Classes**: When a virtual function is called on a base class pointer/reference that points to a derived class object, the derived class's implementation of that function is executed.

Order of Execution

Constructor and Destructor Order:

- ✓ **Base class constructor** executes first, followed by the **derived class constructor**.

When destructors are called, the order is reversed: the **derived class**

Order of Execution of Virtual Function

destructor executes first, followed by the **base class destructor**.

Calling Virtual Functions:

- ✓ If a virtual function is called inside a base class constructor or destructor, the base class version is executed, even if the object being constructed or destructed is of a derived class.
- ✓ If a virtual function is called after the construction of the derived class is complete, the derived class version will be executed.

Order of Execution of Virtual Function

Example - 5

```
#include <iostream>
using namespace std;
class Base
{
  public:
   Base()
   {
      cout << "Base Constructor" << endl;
      show();
   }
   virtual void show()
   {
      cout << "Base Show" << endl;
   }
   virtual ~Base()
   {
      cout << "Base Destructor" << endl;
      show();
   }
};
```

By – Mohammad Imran

Order of Execution of Virtual Function

Example - 5

```
class Derived : public Base
{
  public:
   Derived()
   {
      cout << "Derived Constructor" << endl;
      show();
   }
   virtual void show()
   {
      cout << "Derived Show" << endl;
   }
   virtual ~Derived()
   {
      cout << "Derived Destructor" << endl;
      show();
   }
};
```

By – Mohammad Imran

Order of Execution of Virtual Function

Example - 5

```
int main()
{
   Base *obj=new Derived();
   delete obj;
   return 0;
}
```

OUTPUT

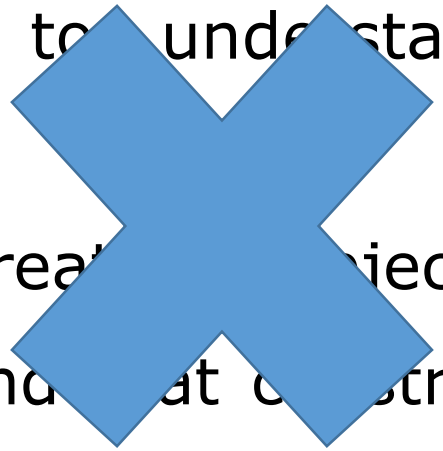
```
Base Constructor
Base Show
Derived Constructor
Derived Show
Derived Destructor
Derived Show
Base Destructor
Base Show
```

By – Mohammad Imran

Overriding Member Function

Constructor is a class member function with the same name as the class name. The main job of the constructor is to allocate memory for class objects. Constructor is automatically called when the object is created. It is very important to understand how constructors are called in inheritance.

We know when we create an object, then automatically the constructor of the class is called and that constructor takes the responsibility to create and initialize the class members. Once we create the object, then we can access the data members and member functions of the class using the object.



By – Mohammad Imran

Concept of Binding

Constructor is a class member function with the same name as the class name. The main job of the constructor is to allocate memory for class objects. Constructor is automatically called when the object is created. It is very important to understand how constructors are called in inheritance.

We know when we create an object, then automatically the constructor of the class is called and that constructor takes the responsibility to create and initialize the class members. Once we create the object, then we can access the data members and member functions of the class using the object.

By – Mohammad Imran

Coding Questions

By – Mohammad Imran

Question - 1

In a C++ program designed to perform basic arithmetic operations, you have two classes: **Addition** and **Multiplication**. The **Addition** class takes two integers by argument constructor and computes their sum, while the **Multiplication** class inherits from **Addition** and computes the product of the same two integers using argument constructor while we created only derived class object.

Click here to see code

By – Mohammad Imran

QUIZ

By – Mohammad Imran

Quiz - 1

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

class Base
{
  public:
   virtual void show() = 0;
   void display()
   {
      cout << "Hello" << endl;
   }
};
```

```
class Derived : public Base
{
  public:
   void display()
   {
      cout << "Hello World";
   }
};
int main(void)
{
   Derived d;
   d.display();
   return 0;
}
```

OUTPUT

Error on object d

By – Mohammad Imran