

# CS-1004 Object Oriented programming

## Week 13

---

**Instructor:**

ATIYA

LECTURER

(COMPUTER SCIENCE DEPARTMENT)

NATIONAL UNIVERSITY- FAST (KHI CAMPUS)

EMAIL: ATIYA.JOKHIO@NU.EDU.PK

ROOM: (LECTURER ROOM#19, CS DEPT BASEMENT-2 )

# Abstract Class

---

- There are cases in which it's useful to define classes that are never instantiated. Such classes are called **abstract base classes** or simply **abstract classes**
- Abstract base classes are incomplete — derived classes must define the “missing pieces”

# Abstract Class

---

- An abstract class provides a base class from which other classes can inherit
- Classes that can be used to instantiate objects are called **concrete classes**
- An abstract can inherit another abstract class

# Concrete Class

---

- An abstract class is meant to be used as the base class from which other classes are derived.
- The derived class is expected to provide implementations for the methods that are not implemented in the base class.
- A derived class that implements all the missing functionality is called a **concrete class**
- **A concrete class is an ordinary class which has no purely virtual functions and hence can be instantiated.**

# Example

---

- Suppose that we have an application that works with **2D** shapes
- We have a class called **2D** with a function **draw()**. But... what shape would the function draw? Should **draw()** be implemented in base class??
- We can extend the class with derived classes such as **Square**, **Triangle** & **Circle**. And provide implementation for **draw()**

# Abstract Class

---

- Abstract base classes are too generic to define real objects.
- Concrete classes provide the specifics that make it reasonable to instantiate objects

# Pure Virtual Functions

---

- A class is made abstract by declaring one or more of its virtual functions to be “**pure**”
- A pure virtual function is specified by placing “= 0” in its declaration
- Example: `virtual void draw() = 0;`

# Virtual vs Pure Virtual

---

- A virtual function has an implementation in the base class; a pure virtual function does not have an implementation in the base class
- Virtual functions *can be* overridden by the derived classes; pure virtual functions *must be* overridden by the derived classes



# Pure Virtual Function

---

- **Pure virtual functions are used to make the class abstract**, so that it can't be instantiated, but a child class can override the pure virtual methods to form a concrete class.
- A **pure virtual function** (or abstract function) in C++ is a virtual function for which we don't have implementation, we only declare it.
- A pure virtual function is declared by assigning 0 in declaration. This is a good way to define an interface in C++.

# Example

---

```
1  #include <iostream>
2  using namespace std;
3  class AnimalMovement {
4  public:
5      virtual void walk() = 0;
6      virtual void run() = 0;
7  };
8
9  class Movement : public AnimalMovement {
10 public:
11     void walk(){cout <<"walk function calling "<<endl;}
12     void run(){cout <<"run function calling "<<endl;}
13 };
14 int main()
15 {
16     Movement m;
17     m.walk();
18     m.run();
19
20     return 0;
21 }
22
```

# Why we use pure virtual function???

---

# Why we use pure virtual function

---

- A pure virtual function is useful when we have a function that we want to put in the base class, but only the derived classes know what it should return.
- A pure virtual function makes it so the base class can not be instantiated, and the derived classes are forced to define these functions before they can be instantiated.
- This helps ensure the derived classes do not forget to redefine functions that the base class was expecting them to.

# Virtual vs Pure Virtual

---

Definition is given in base class.	No definition is given in base class.
Base class having virtual function can be instantiated i.e. its object can be made.	Base class having pure virtual function becomes abstract i.e. it cannot be instantiated.
If derived class do not redefine virtual function of base class, then it does not affect compilation.	If derived class do not redefine virtual function of base class, then no compilation error but derived class also becomes abstract just like the base class.
All derived class may or may not redefine virtual function of base class.	All derived class must redefine pure virtual function of base class otherwise derived class also becomes abstract just like base class.

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

---

```
class Base{
    int x;
public:
    virtual void fun() = 0;
    int getX() {
        return x; };
// This class inherits from Base and implements fun()
class Derived: public Base{
    int y;
public:
    void fun() { cout << "fun() called"; };
int main(void){
    Derived d;
    d.fun();
    return 0;
}
```

---

```
fun() called
```

```
-----
```

```
Process exited after 0.7964 seconds with return value 0
```

```
Press any key to continue . . . |
```

# We can not instantiate an abstract class

---

```
#include<iostream>

using namespace std;

class Test{
    int x;
public:
    virtual void show() = 0;
    int getX() {
        return x; };
int main(void){
    Test t;
    return 0;}
```



---

**In function 'int main()':**

[Error] cannot declare variable 't' to be of abstract type 'Test'

[Note] because the following virtual functions are pure within 'Test':

[Note] virtual void Test::show()

# Abstract Class

---

- Although we cannot instantiate objects of an abstract base class, we *can use the* abstract base class to declare pointers and references that can refer to objects of its derived class
- Abstract classes support the principle of Polymorphism

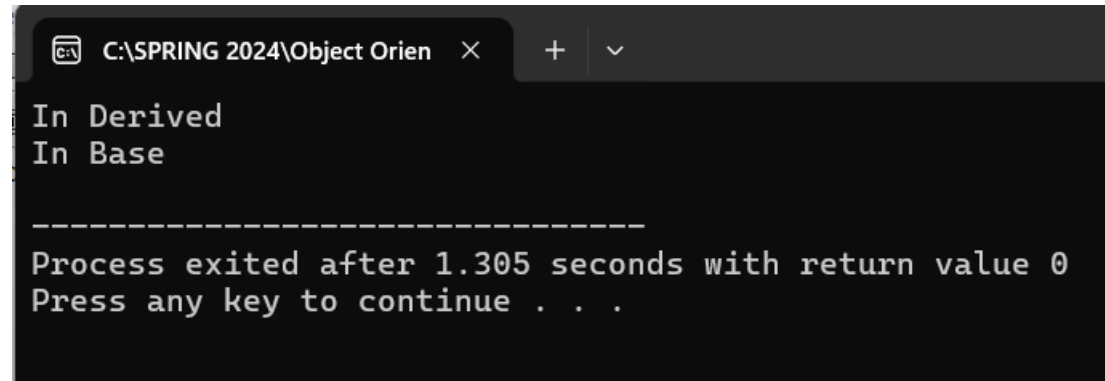
# *We can have pointers and references of abstract class type.*

---

```
class Base{
    int x;
public:
    void getX() { cout << "In Base \n"; }
    virtual void show() = 0;};
class Derived: public Base{
public:
    void show() { cout << "In Derived \n"; };
int main(void){
    Base *bp = new Derived();
    bp->show();
    bp->getX();
    return 0;}
```

*We can have pointers and references of abstract class type.*

---



```
C:\SPRING 2024\Object Orien  X + v
In Derived
In Base

-----
Process exited after 1.305 seconds with return value 0
Press any key to continue . . .
```

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

---

```
class Base {  
public:  
    virtual void show() = 0;};  
class Derived : public Base { };  
int main(void){  
    Derived d;  
    return 0;}
```

---

In function 'int main()':

[Error] cannot declare variable 'd' to be of abstract type 'Derived'

[Note] because the following virtual functions are pure within 'Derived':

[Note] virtual void Base::show()

---

---

```
class Base{
public:
virtual void show() = 0;};
class Derived : public Base { };
class Derived1 : public Derived {
public:
void show() {
cout<<"show method implementation \n";} };
int main(void){
Derived1 d;
d.show();
return 0;}
```

---

```
show method implementation
```

```
-----
```

```
Process exited after 0.1622 seconds with return value 0
```

```
Press any key to continue . . .
```



# *An abstract class can have constructors.*

---

```
class Base{
protected:
int x;
public:
virtual void fun() = 0;
Base(int i) {
    x = i;
    cout<<"Constructor of base called\n"; }};
class Derived: public Base{
    int y;
public:
    Derived(int i, int j):Base(i) { y = j; }
    void fun() { cout << "x = " << x << ", y = " << y<<"\n";
};
```

# *An abstract class can have constructors.*

---

```
int main(void){  
    Derived d(4, 5);  
    d.fun();  
    //object creation using pointer of base class  
    Base *ptr=new Derived(6,7);  
    ptr->fun();  
    return 0;}
```

# *An abstract class can have constructors.*

---

```
Constructor of base called  
x = 4, y = 5  
Constructor of base called  
x = 6, y = 7  
  
-----  
Process exited after 0.2104 seconds with return value 0  
Press any key to continue . . .
```

# Can a destructor be pure virtual in C++?

---






- Yes, it is possible to have pure virtual destructor.
- if a class contains a pure virtual destructor, it must provide a function body for the pure virtual destructor.
- You may be wondering why a pure virtual function requires a function body. ?
- The reason is because destructors (unlike other functions) are not actually ‘overridden’, rather they are always called in the reverse order of the class derivation.
- This means that a derived class’ destructor will be invoked first, then base class destructor will be called.
- If the definition of the pure virtual destructor is not provided, then what function body will be called during object destruction?
- Therefore the compiler and linker enforce the existence of a function body for pure virtual destructors.

# Pure-Virtual destructor

```
#include <iostream>
using namespace std;
class Base {
public:
    virtual ~Base() = 0; // Pure virtual destructor
};

class Derived : public Base {
public:
    ~Derived() {
        cout << "Derived destructor called." << std::endl;
    }
};

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

es  Compile Log  Debug  Find Results  Console  Close

Message

es (x86)\Embarcadero\Dev-Cpp\T... C:\Users\atiya\AppData\Local\Temp\ccYJqthq.o:Virtual Destructor.cpp:(.text\$\_ZN7DerivedD1Ev[\_ZN7DerivedD1Ev]+0x44): undefined reference to `Base::~~Base()'

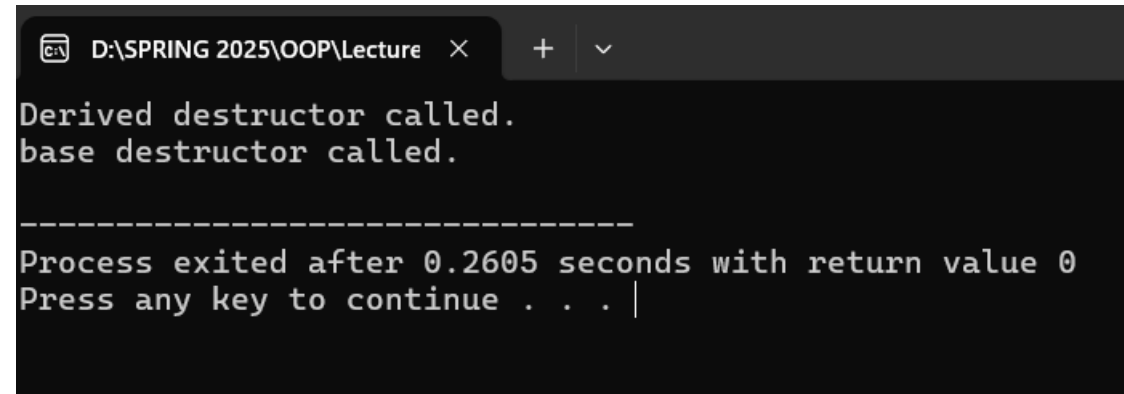
# Pure-Virtual destructor

```
#include <iostream>
using namespace std;
class Base {
public:
    virtual ~Base() = 0; // Pure virtual destructor
};

// Pure virtual destructor without definition
Base::~~Base(){
    cout << "base destructor called." << std::endl;
}

class Derived : public Base {
public:
    ~Derived() {
        cout << "Derived destructor called." << std::endl;
    }
};

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

A screenshot of a terminal window showing the output of a C++ program. The window title is "D:\SPRING 2025\OOP\Lecture". The output shows the derived destructor being called first, followed by the base destructor. A separator line of dashes is used to separate the execution time and return value information from the destructor output. The text "Process exited after 0.2605 seconds with return value 0" and "Press any key to continue . . ." are visible at the bottom.

```
D:\SPRING 2025\OOP\Lecture x + v
Derived destructor called.
base destructor called.

-----
Process exited after 0.2605 seconds with return value 0
Press any key to continue . . . |
```

# TASK

---

You are developing a language learning app that teaches various forms of English writing, such as essays, poems, and reports.

Each writing type shares some common behavior (like writing content and checking grammar), but each has its unique writing style and format.

Design an abstract class called `EnglishWriting` with at least one pure virtual function.

Then, create derived classes `Essay`, `Poem`, and `Report` that implement the specific behavior.

- Write a C++ code and show implementation of base class using pure virtual functions and show how would polymorphism be useful in this scenario?
- Demonstrate how the `EnglishWriting*` pointer could be used to handle different writing types interchangeably.