**National University of Computer & Emerging Sciences, Karachi**

**Computer Science Department**

**Spring 2023, Lab Manual - 12**

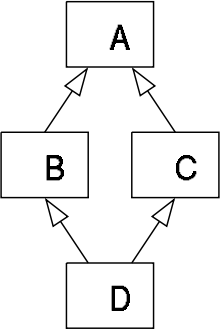
|  |  |
| --- | --- |
| **Course Code: CL-1004** | **Course : Object Oriented Programming Lab** |
| **Instructor(s) :** | **Nida Munawar** |

# Outline:

1. Diamond Problem in Hybrid Inheritance
2. Abstract Classes
3. Virtual Functions
4. Lab Tasks

## **Diamond Problem in Hybrid Inheritance:**

In case of hybrid inheritance, a Diamond problem may arise. The “dreaded diamond” refers to a class structure in which a particular class appears more than once in a class’s inheritance hierarchy.



**Example of Diamond Problem:**

#include<iostream>

using namespace std;

class A {

public:

int a;

};

class B : public A{

public:

int b;

};

class C : public A{

public:

int c;

};

class D : public B, public C{

public:

int d;

};

int main() {

D obj;

obj.a = 200; //will cause an error

}

## **How to Solve the Diamond Problem?**

**Answer: Virtual Base Classes**

To share a base class, simply insert the “virtual” keyword in the inheritance list of the

derived class. This creates what is called a **virtual base class**, which means there is only

one base object. The base object is shared between all objects in the inheritance tree and it

is only constructed once.

**Solving the Diamond Problem:**

#include<iostream>

using namespace std;

class A {

public:

int a;

};

class B : virtual public A{ //adding the virtual keyword

public:

int b;

};

class C : virtual public A{ //adding the virtual keyword

public:

int c;

};

class D : public B, public C{

public:

int d;

};

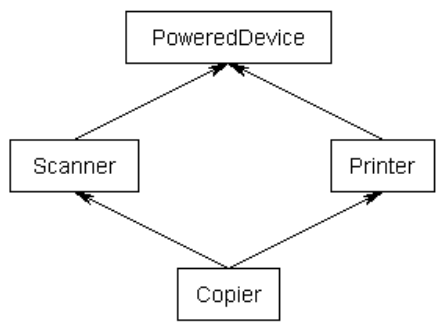
int main() {

D obj;

obj.a = 200; //will no longer cause an error

}

## **Diamond Problem with Real Classes and Objects:**



#include <iostream>

using namespace std;

class PoweredDevice {

public:

PoweredDevice(int power){

cout << "PoweredDevice: " << power << '\n';

}

};

class Scanner: public PoweredDevice{

public:

Scanner(int scanner, int power) : PoweredDevice(power){

cout << "Scanner: " << scanner << '\n';

}

};

class Printer: public PoweredDevice{

public:

Printer(int printer, int power) : PoweredDevice(power){

cout << "Printer: " << printer << '\n';

}

};

class Copier: public Scanner, public Printer {

public:

Copier(int scanner, int printer, int power) : Scanner(scanner, power), Printer(printer, power) { }

};

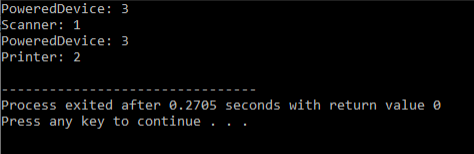
int main() {

Copier copier(1, 2, 3);

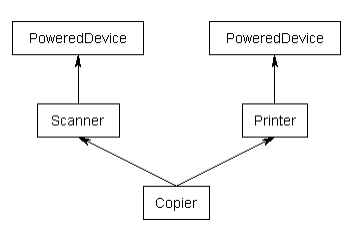
return 0;

}

If you were to create a Copier class object, by default you would end up with two copies of the PoweredDevice class -- one from Printer, and one from Scanner. This has the following structure:



**By using Virtual Base Classes:**



#include <iostream>

using namespace std;

class PoweredDevice {

public:

PoweredDevice(int power){

cout << "PoweredDevice: " << power << '\n';

}

};

class Scanner: virtual public PoweredDevice{ // note: PoweredDevice is now a virtual base class

public:

Scanner(int scanner, int power) : PoweredDevice(power){ // this line is required to create Scanner objects, but ignored in this case

cout << "Scanner: " << scanner << '\n';

}

};

class Printer: virtual public PoweredDevice{ // note: PoweredDevice is now a virtual base class

public:

Printer(int printer, int power) : PoweredDevice(power){ // this line is required to create Printer objects, but ignored in this case

cout << "Printer: " << printer << '\n';

}

};

class Copier: public Scanner, public Printer {

public:

Copier(int scanner, int printer, int power) : PoweredDevice(power), Scanner(scanner, power), Printer(printer, power) // PoweredDevice is constructed here at PoweredDevice(power)

{ }

};

int main() {

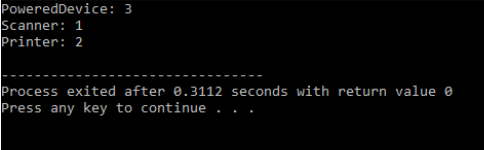
Copier copier(1, 2, 3);

return 0;

}

Now, when you create a Copier class object, you will get only one copy of PoweredDevice per Copier that will be shared by both Scanner and Printer. However, this leads to one more problem: if Scanner and Printer share a PoweredDevice base class, who is responsible for creating it?

The answer, as it turns out, is Copier. The Copier constructor is responsible for creating PoweredDevice. Consequently, this is one time when Copier is allowed to call a non-immediate-parent constructor directly.



## **Abstract Classes:**

An **abstract class** is a class that has at least one pure virtual function (i.e., a function that has no function body). The classes inheriting the abstract class must provide a definition for the pure virtual function; otherwise, the subclass would become an abstract class itself.

Abstract classes are essential for providing abstraction to the code to make it reusable and extendable. For Example, a *Vehicle* parent class with *Truck* and *Motorbike* inheriting from it is an abstraction that easily allows more vehicles to be added. However, even though all vehicles have wheels, not all vehicles have the same number of wheels – this is where a pure virtual function is needed.

## **Virtual Functions:**

A pure virtual function (or abstract function) in C++ is a [virtual function](https://www.geeksforgeeks.org/virtual-functions-and-runtime-polymorphism-in-c-set-1-introduction/)for which we don’t have implementation (we do not have function body), we only declare it. A pure virtual function is declared by assigning 0 in declaration.

See the following example.

// An abstract class

class Test

{

    // Data members of class

public:

    // Pure Virtual Function

    virtual void show() = 0;

   /\* Other members \*/

};

**Note:**

* The = 0  syntax doesn’t mean we are assigning 0 to the function. It’s just the way we define pure virtual function.
* A pure virtual function is implemented by classes which are derived from Abstract class.
* A class is abstract if it has at least one pure virtual function. In example given above, Test is abstract class as it has virtual function show().

**A complete example :**

A pure virtual function is implemented by classes which are derived from a Abstract class. Following is a simple example to demonstrate the same.

|  |
| --- |
| #include<iostream>  using namespace std;    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;  } |

Output:

fun() called

## **Object of abstract class cannot be created**

// pure virtual functions make a class abstract

#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; }

Output:

Compiler Error: cannot declare variable 't' to be of abstract

type 'Test' because the following virtual functions are pure

within 'Test': note: virtual void Test::show()

* **An abstract class can have constructors. Consider this code that compiles and runs fine.**

#include<iostream>

using namespace std;

// An abstract class with constructor

class Base

{

protected:

   int x;

public:

  virtual void fun() = 0;

  Base(int i) { x = i; }

};

class Derived: public Base

{

    int y;

public:

    Derived(int i, int j):Base(i) { y = j; }

    void fun() { cout << "x = " << x << ", y = " << y; }

};

int main(void)

{

    Derived d(4, 5);

    d.fun();

    return 0; }

Output:

x = 4, y = 5

## **Example: Abstract class and Virtual Function in calculating Area of Square and Circle:**

// C++ program to calculate the area of a square and a circle

#include <iostream>

using namespace std;

// Abstract class

class Shape {

protected:

float dimension;

public:

void getDimension() {

cin >> dimension;

}

// pure virtual Function

virtual float calculateArea() = 0;

};

// Derived class

class Square : public Shape {

public:

float calculateArea() {

return dimension \* dimension;

}

};

// Derived class

class Circle : public Shape {

public:

float calculateArea() {

return 3.14 \* dimension \* dimension;

}

};

int main() {

Square square;

Circle circle;

cout << "Enter the length of the square: ";

square.getDimension();

cout << "Area of square: " << square.calculateArea() << endl;

cout << "\nEnter radius of the circle: ";

circle.getDimension();

cout << "Area of circle: " << circle.calculateArea() << endl;

return 0;

}

**Output**

Enter length to calculate the area of a square: 4

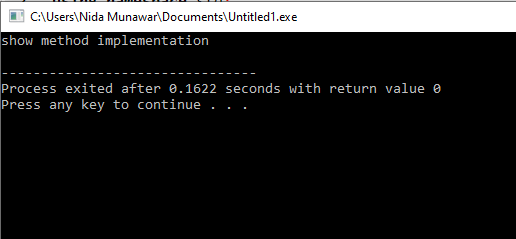
Area of square: 16

Enter radius to calculate the area of a circle: 5

Area of circle: 78.5

In this program, virtual float calculateArea() = 0; inside the Shape class is a pure virtual function.

That's why we must provide the implementation of calculateArea() in both of our derived classes, or else we will get an error.

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

**Lab Tasks:**

**Task 1:**

Suppose interest rates of HMB, HBL & MCB banks are 4, 2 and 3 percent respectively. You need to write a program using the concept of abstract classes and virtual functions that simply displays interest rate for each bank.

**Task 2:**

We have to calculate the area of a rectangle, a square and a circle. Create an abstract class 'Shape' with three abstract methods namely 'RectangleArea' taking two parameters, 'SquareArea' and 'CircleArea' taking one parameter each. The parameters of 'RectangleArea' are its length and breadth, that of 'SquareArea' is its side and that of 'CircleArea' is its radius. Now create another class 'Area' containing all the three methods 'RectangleArea', 'SquareArea' and 'CircleArea' for printing the area of rectangle, square and circle respectively. Create an object of class 'Area' and call all the three methods.

**Task 3:**

Write a program to calculate final bill after discount. “ImtiazStore” gives 7 percent discount on total\_bill while “BinHashimStore” gives 5 percent discount on total\_bill. You have to initialize value of total\_bill through a constructor and then calculate final bill after discount for both stores using the concept of abstract class and virtual functions.

**Task 4:**

We have to calculate the percentage of marks obtained in three subjects (each out of 100) by student A and in four subjects (each out of 100) by student B. Create an abstract class 'Marks' with an abstract method 'getPercentage'. It is inherited by two other classes 'A' and 'B' each having a method with the same name which returns the percentage of the students. The constructor of student A takes the marks in three subjects as its parameters and the marks in four subjects as its parameters for student B. Create an object for eac of the two classes and print the percentage of marks for both the students.

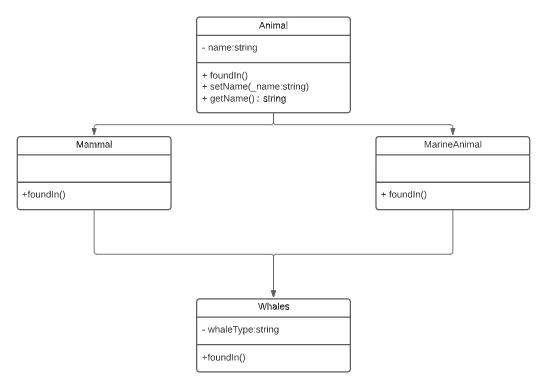
**Task 5:**

Design and implement a program that shows the relationship between person, student and professor. Your person class must contain two pure virtual functions named getData() of type void and isOutstanding() of type bool and as well getName() and putName() that will read and print the person name. Class student must consist of function name getData (), which reads the GPA of specific person and isOutstanding() function which returns true if the person GPA is greater than 3.5 else should return false. Class professor should take the respective persons publications in getData() and will return true in Outstanding() if publications are greater than 100 else will return false . Your main function should ask the user either you want to insert the data in professor or student until and unless user so no to add moredata.

**Task 6:**

### **Create an abstract class 'Bank' with an abstract method 'getBalance'. $100, $150 and $200 are deposited in banks A, B and C respectively. 'BankA', 'BankB' and 'BankC' are subclasses of class 'Bank', each having a method named 'getBalance'. Call this method by creating an object of each of the three classes.**

## Task - 07:

****

Create a class, as depicted in the diagram. Make sure to use virtual base classes for this task.

Make an object **whaleBaleen**. Then set the **whaleType to “Baleen”,** **and name to “whale”**

Write the following in the function descriptions for **foundIn()**

* **Animal::foundIn()** – Prints: An animal can be found in many places
* **Mammal::foundIn()** – Prints: A mammal can be found in water or on land
* **MarineAnimal::foundIn()** – Prints: A marine animal can only be found in bodies of water
* **Whales::foundIn()** – Prints: A <whaleType> <name> can only be found in the ocean.