C++: Inheritance & Polymorphism

C++: Abstract Class

- **Abstract Class**
  - Class that contain at least **one** *pure virtual function* are known as Abstract classes
  - Virtual Function
    - Member function that can be ***redefined*** in a derived class, i.e. override the methods in the derived class to define its behavior
    - Virtual function assigned **=0** is called ***pure virtual function***
  - **CANNOT** be instantiated

C++: Abstract Class

Refer to AbstractClass.cpp

```
class GeneralShape{

        double area;
public:
        double virtual getArea()=0;
};
```

Pure virtual function

```
int main(){

    GeneralShape * b;

    GeneralShape* b = new GeneralShape;
    GeneralShape b;
}
```

Cannot create an instance of an abstract class

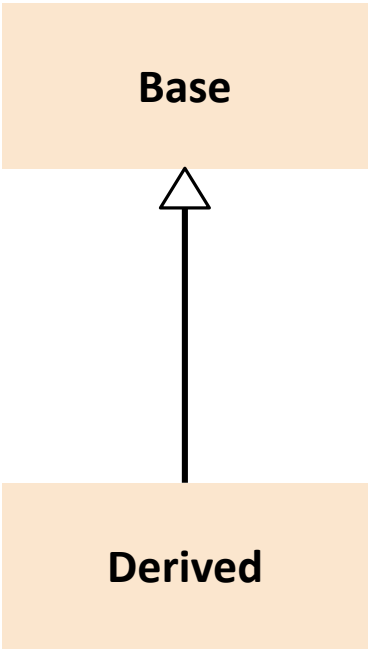
C++: Inheritance

- **Inheritance**
  - Allows creation of a *class* (derived) from *another* class (base)
  - How is it defined

    class Derived: *access_specifier* Base

    class Derived: *access_specifier* Base, *access_specifier* Base1, *access_specifier* Base2,…….

    - class Derived: public Base
      - Public Inheritance
        - *public members* of the base → *public members* of the derived
        - protected members of the base → protected members of the derived.



C++: Inheritance

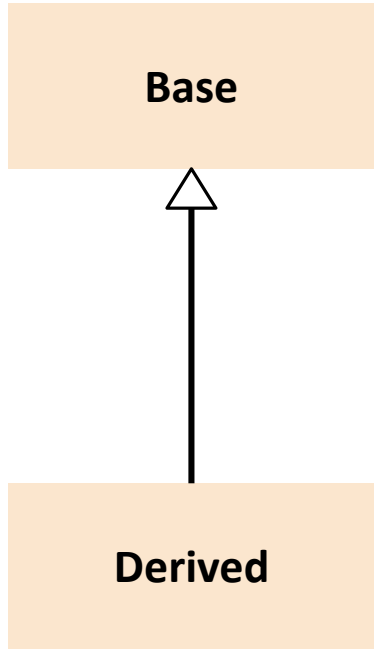
— Inheritance

› class Derived: protected Base

- Protected Inheritance
  - *public members* of the base → *protected members* of the derived
  - *protected members* of the base → protected members of the derived

› class Derived: private Base (default)

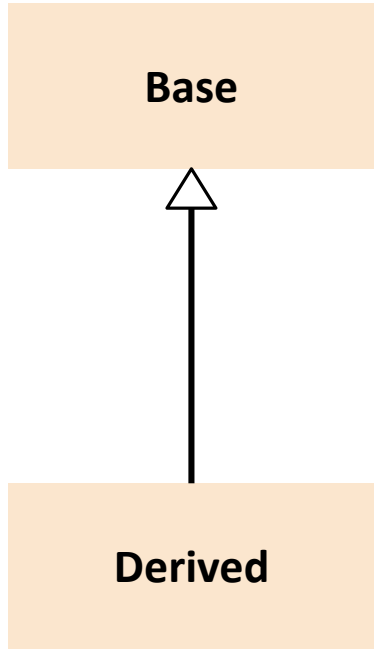
- Private Inheritance
  - *public members* of the base → *private members* of the derived
  - *protected members* of the base → *private members* of the derived



C++: Inheritance

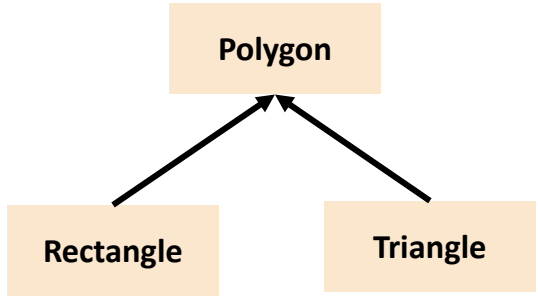
— Inheritance

- » Derived class *inherits* all Base class methods **EXCEPT**:
  - › Constructors, Destructors and copy constructors of the Base
  - › Overloaded operators of the Base



C++: Inheritance

— Example




```
class Polygon {
        protected:
                int width, height;
        public:
              void setValues(int a, int b)
              {
                    width = a;
                    height = b;
              }
};
```

```
class Rectangle : public Polygon {
        public:
        int area() {
                return width * height;
        }
};
```

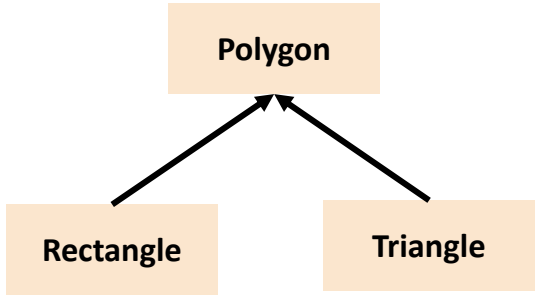
```
class Triangle: public Polygon {
        public:
        int area() {
                return width * height / 2;
        }
};
```

```
int main () {
  Rectangle rect;
  Triangle trgl;
  rect.setValues(4,5);
  trgl.setValues (4,5);
  cout << rect.area() << '\n';
  cout << trgl.area() << '\n';
  return 0;
}
```

C++: Inheritance

— Details

```
class Rectangle : public Polygon {
          public:
          int area() {
                    return width * height;
          }
};
```




```
class Rectangle {
protected:
          int width, height;
public:
          void setValues(int a, int b) {
                    width = a;
                    height = b;
                    }
          int area() {
                    return width * height;
          }
};
```

A copy inherited from Polygon

C++: Inheritance

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

class Player{
protected:
            string name;
            string organization;
            string team;

public:
            Player(string name,string organization,string team){
                        this->name = name;
                        this->organization = organization;
                        this->team = team;
            }
};
```

```
class NFLPlayer:public Player{

protected:
            double passCompletion;
public:
            NFLPlayer(string name,string organization,string team,double pc)
            :Player(name,organization,team), passCompletion(pc){
            }

};
```

```
NFLPlayer(string name,string organization,string team,doublepassCompletion):Player(name,organization,team),passCompletion(pc)
```

Constructor of NFLPlayer

Calling the base or superclass constructor - Player

Initialization of passCompletion Variable in NFLPlayer

C++: Inheritance

```
class classA{
  public:
   int x;
   int u;
   classA(){
    x = 10; u = 20;
   }
};
class classB{
 public:
  int y;
  classB(){
   y = 30;
  }
};
class classC: public classA, public classB{
 public:
  int z;
  classC(){
   z = 40;
 }
};
```

```
int main(){
    classC c;
}
```

What happens in the memory

classC
z
y
x
u

Constructors are called to initialize the variables

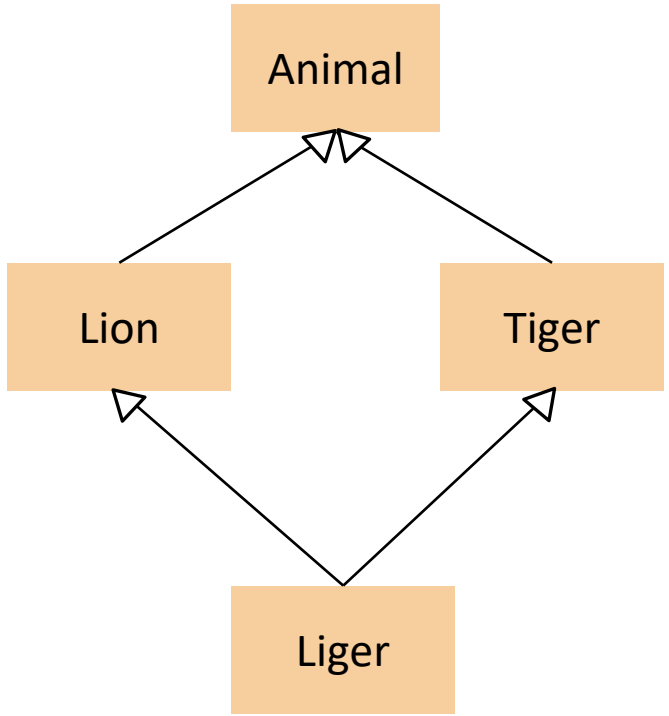
Destructors are called for clean up

C++: Inheritance

Refer to DiamondProblem.cpp

— Diamond Problem

» Caused due to multiple inheritance.

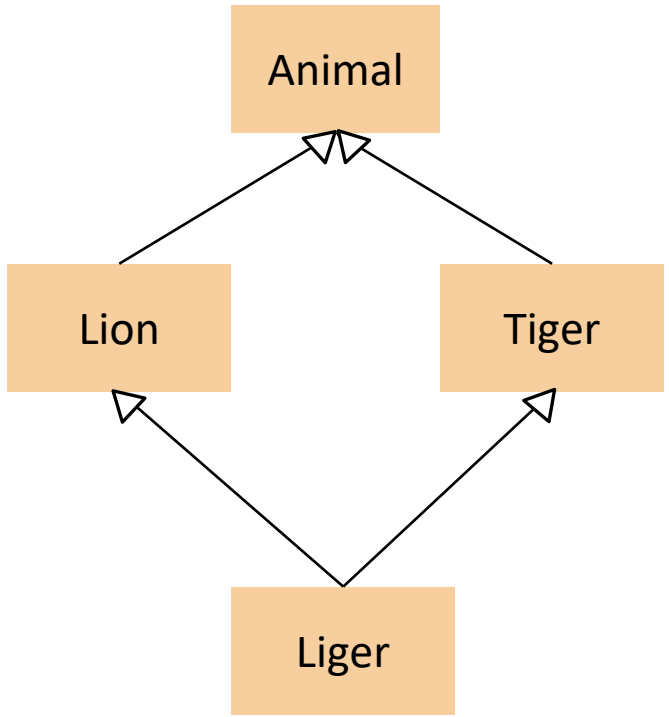


Liger has two definitions of **setWeight** and **getWeight** from both parents **Tiger** and **Lion** which they inherited from **Animal**

```
class Animal{
public:
      double weight = 0;
      Animal(){}
      void setWeight(double weight){
            this->weight = weight;
      }
      double getWeight(){
            return weight;
      }
};

class Tiger: public Animal{
.........
};

class Lion: public Animal{
.......
};

class Liger: public Tiger, public Lion{
.......
};
```

C++: Inheritance

Refer to DiamondProblem.cpp



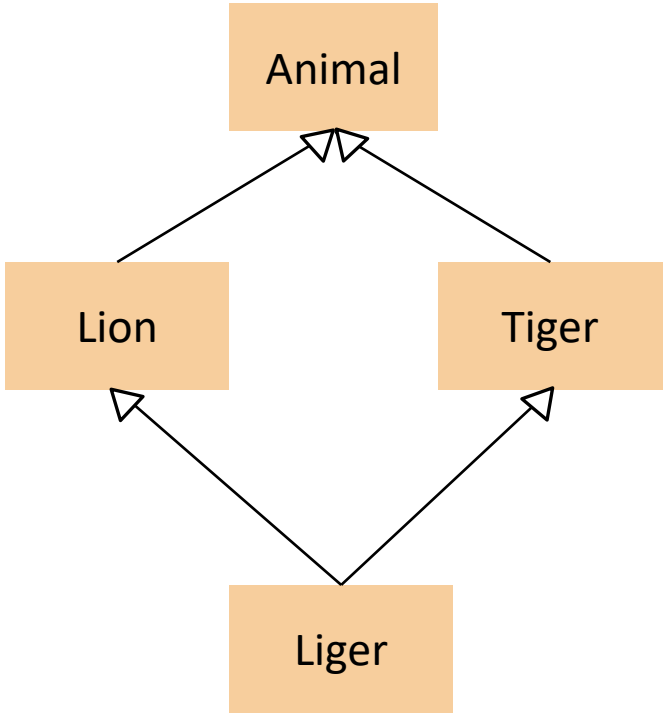
- **Diamond Problem**
  - What is happening
    - To create Liger object
      - Create Lion
        - Create Animal
      - Create Tiger
        - Create Animal
      - Finally, Liger has two instance of Animal

C++: Inheritance

Refer to DiamondProblem.cpp

— Diamond Problem

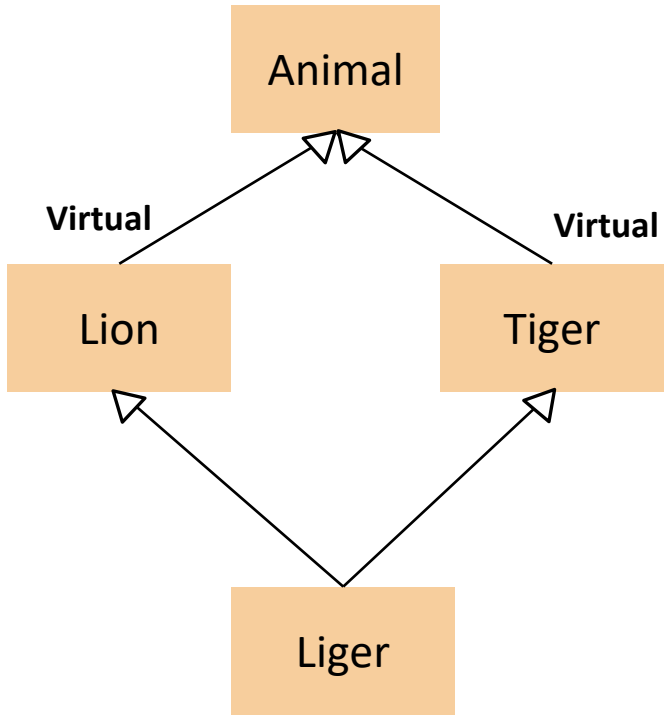
» Solution – Use virtual keyword




```
class Animal{
public:
      double weight = 0;
      Animal(){}
      void setWeight(double weight){
            this->weight = weight;
      }
      double getWeight(){
            return weight;
      }
};

class Tiger: virtual public Animal{
.........
};

class Lion: virtual public Animal{
.......
};

class Liger: public Tiger, public Lion{
.......
};
```

C++: Inheritance

Refer to DiamondProblem.cpp



- **Diamond Problem**
  - What is happening
    - To create Liger object
      - Create Lion
      - Create Tiger
      - Create Animal object
        - Shared between Tiger, Lion and Liger

C++: Polymorphism

- **Polymorphism**
  - Refers to the ability to associate many meanings to functions by means of the **late-binding** mechanism
  - Polymorphism
    - You are telling the compiler *"I do not know how this function is implemented. Wait until it is used in a program, and then get the function implementation from the instance"*

C++: Polymorphism

- **Virtual**
  - Provides *polymorphic* capability to the classes
  - Member function declared as *virtual* in the base class can later be redefined in the derived classes.
    - Non-virtual members can also be redefined in derived classes
      - **BUT** non-virtual members of derived classes cannot be accessed through a reference of the base class

C++: Polymorphism

```
#include <iostream>
using namespace std;
const double PI = 3.141;
class Shape {
    public:
        virtual void area() =0;
};

class Rectangle:public Shape{
  public:
    int w, h;
    Rectangle (int w, int h): w(w), h(h){}
    void area(){
        cout<<"Area of Rectangle is -  "<<w*h<<endl;
    }
};
class Circle:public Shape{
   public:
     int radius;
     Circle (int r): radius(r){}
     void area(){
        cout<<"Area of Circle is -  "<<radius*radius*PI<<endl;
     }
};
```

```
int main(){

Shape* s;
s = new Rectangle(10, 10);
s->area();
delete s;

s = new Circle(10);
s->area();
delete s;

return 0;
}
```

C++: Polymorphism

```
class Airplane{
protected:
            string name;
            int seats;
            string arrangment;

public:
      Airplane(string name,int seats, string arrangment){
            this->name = name;
            this->seats = seats;
            this->arrangment = arrangment;
      }
virtual void defaultSeating(){
      cout<<"By default, the each row has '3 2 3' seating arraignment
      "<<endl;
};
```

```
class Emirates:public Airplane{
      public:
            Emirates(string name,int seats, string arrangment)
                  :Airplane(name,seats,arrangment){
            }
      void defaultSeating(){
            cout<<"Emirates overrides the default behavior and have
            "<<arrangment<<" seating arraignment for each row"<<endl;
      }
};
```

Overriding the behavior

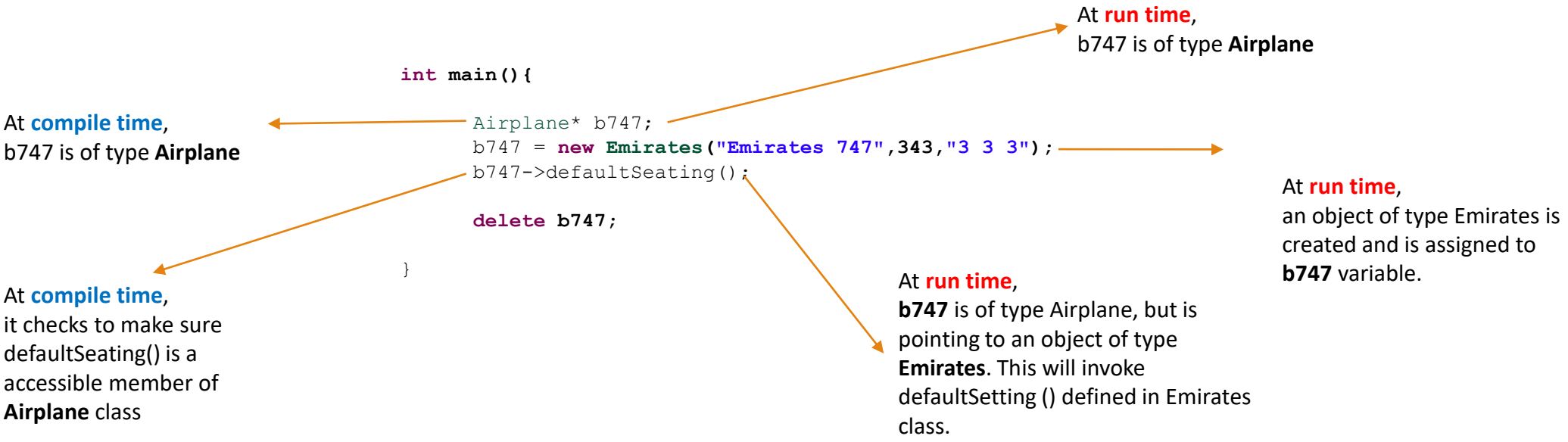
```
int main(){

      Airplane* b747;
      b747 = new Emirates("Emirates 747",343,"3 3 3");
      b747->defaultSeating();

      delete b747;

}
```

b747 is of type **Airplane**

b747 is pointing an object of type **Emirates**

Will call the defaultSeating() method in Emirate class

C++: Polymorphism



C++: Virtual Destructors

— Destructor

» Always make the Destructors ***Virtual.*** Why?

```
int main(){
        Animal* base = new Liger;
        base->setWeight(600);
        cout<<"Liger Weight - "<<base->getWeight()<<endl;
        delete base;
}
```

As the ***base*** is of type **Animal**, Animal destructor is only called.

Liger object is never destroyed.

```
class Animal{
public:
        double weight = 0;
        Animal(){}
        virtual void setWeight(double weight){
                this->weight = weight;
        }
        virtual double getWeight(){
                return weight;
        }
        ~Animal(){
                cout<<"Destroying Animal"<<endl;
        }
};

class Liger:public Tiger, public Lion{
public:
        string description;
        Liger(){}
        void setDescription(string description){
                this->description= description;
        }
        string getDescription(){
                return description;
        }
        ~Liger(){
                cout<<"Destroying Liger"<<endl;
        }
};
```

C++: Virtual Destructors

- **Destructor**
  - Make the destructor virtual

```
int main(){

    Animal* base = new Liger;

    base->setWeight(600);

    cout<<"Liger Weight - "<<base->getWeight()<<endl;

    delete base;
}
```

Destructor's of both Animal, Tiger, Lion and Liger are called.

```
class Animal{
public:
    double weight = 0;
    Animal(){}
    virtual void setWeight(double weight){
        this->weight = weight;
    }
    virtual double getWeight(){
        return weight;
    }
    virtual  ~Animal(){
        cout<<"Destroying Animal"<<endl;
    }
};

class Liger:public Tiger, public Lion{
public:
    string description;
    Liger(){}
    void setDescription(string description){
        this->description= description;
    }
    string getDescription(){
        return description;
    }
    ~Liger(){
        cout<<"Destroying Liger"<<endl;
    }
};
```

Thank You

Question, Comments & Feedback