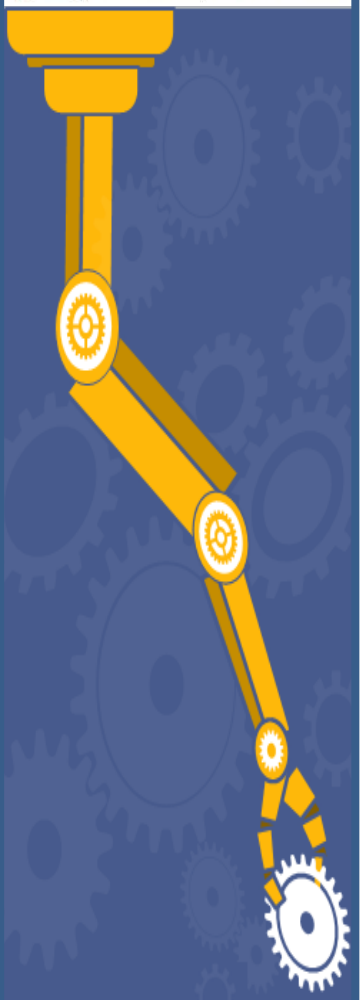
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)



BCSE102L- Structured and object-oriented programming

Dr. P.Keerthika
Associate Professor
School of Computer Science & Engineering
VIT, Vellore.

BCSE102L- Structured and object-oriented programming

Module:1	**C Programming Fundamentals**	**2 hours**
Variables - Reserved words – Data Types – Operators – Operator Precedence - Expressions - Type Conversions - I/O statements - Branching and Looping: if, if-else, nested if, if-else ladder, switch statement, goto statement - Loops: for, while and do…while – break and continue statements.		
Module:2	**Arrays and Functions**	**4 hours**
Arrays: One Dimensional array - Two-Dimensional Array – Strings and its operations. User Defined Functions: Declaration – Definition – call by value and call by reference - Types of Functions - Recursive functions - Storage Classes - Scope, Visibility and Lifetime of Variables.		
Module:3	**Pointers**	**4 hours**
Declaration and Access of Pointer Variables, Pointer arithmetic – Dynamic memory allocation – Pointers and arrays - Pointers and functions.		
Module:4	**Structure and Union**	**2 hours**
Declaration, Initialization, Access of Structure Variables - Arrays of Structure - Arrays within Structure - Structure within Structures - Structures and Functions – Pointers to Structure -		
Module:5	**Overview of Object-Oriented Programming**	**5 hours**
Features of OOP - Classes and Objects - "this" pointer - Constructors and Destructors - Static Data Members, Static Member Functions and Objects - Inline Functions – Call by reference - Functions with default Arguments - Functions with Objects as Arguments - Friend Functions and Friend Classes.		
Module:6	**Inheritance**	**5 hours**
Inheritance - Types of Inheritance: Single inheritance, Multiple Inheritance, Multi-level Inheritance, Hierarchical Inheritance - Multipath Inheritance - Inheritance and constructors.		
Module:7	**Polymorphism**	**4 hours**
Function Overloading - Operator Overloading – Dynamic Polymorphism - Virtual Functions - Pure virtual Functions - Abstract Classes.		
Module:8	**Generic Programming**	**4 hours**
Function templates and class templates, Standard Template Library.		
	Total Lecture hours:	**30 hours**

BCSE102L- Structured and object-oriented programming – Text Books and Reference Books

Text Book(s)	
1.	Herbert Schildt, C: The Complete Reference, 4th Edition, McGraw Hill Education, 2017
2.	Herbert Schildt, C++: The Complete Reference, 4th Edition, McGraw Hill Education, 2017.
Reference Books	
1.	Yashavant Kanetkar, Let Us C: 17th Edition, BPB Publicaitons, 2020.
2.	Stanley Lippman and Josee Lajoie, C++ Primer, 5th Edition, Addison-Wesley publishers, 2012.

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

BCSE102P- Structured and object-oriented programming Laboratory

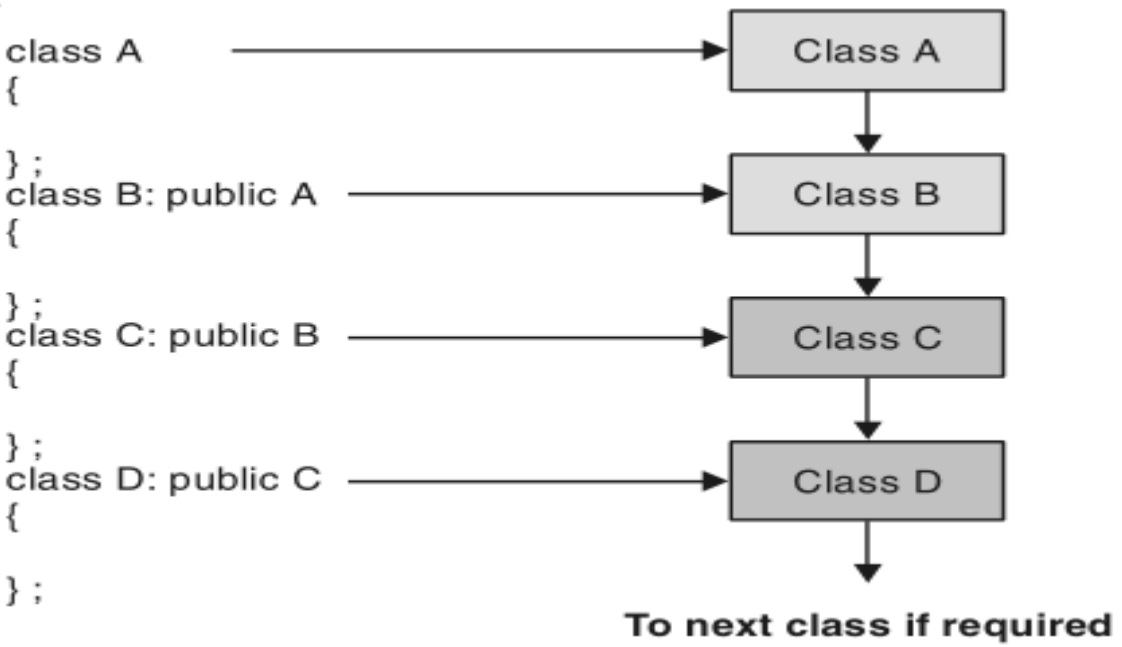
	Indicative Experiments
1.	Programs using basic control structures, branching and looping
2.	Experiment the use of 1-D, 2-D arrays and strings and Functions
3.	Demonstrate the application of pointers
4.	Experiment structures and unions
5.	Programs on basic Object-Oriented Programming constructs.
6.	Demonstrate various categories of inheritance
7.	Program to apply kinds of polymorphism.
8.	Develop generic templates and Standard Template Libraries.

Text Book(s)	
1.	Robert C. Seacord, Effective C: An Introduction to Professional C Programming, 1st Edition, No Starch Press, 2020.
Reference Book(s)	
1.	Vardan Grigoryan and Shunguang Wu, Expert C++: Become a proficient programmer by learning coding best practices with C++17 and C++20's latest features, 1st Edition, Packt Publishing Limited, 2020.

BCSE102L- Structured and Object-Oriented Programming

- **Module-6: Inheritance**
  - **Inheritance- Introduction & Types**
  - **Single Inheritance**
  - **Multiple Inheritance**
  - **Multilevel Inheritance**
  - **Hierarchical Inheritance**
  - **Multipath/Hybrid Inheritance**
  - **Inheritance and Constructors**

Multilevel Inheritance



- When a derived(child) class inherits the base class and acts as the base class(parent class) to the other class, it is called Multilevel Inheritance.
- There can be any number of levels i.e any number of derived classes in multilevel inheritance.

Example : I Two- level Inheritance

```
class first
{
public :
void show_first( )
{
cout<<"Hello from first"<<endl;
}
};

class second :public first
{
public :
void show_second( )
{
cout<<"Hello from second"<<endl;
}
};

class third :public second
{
public :
void show_third( )
{
show_first( );
show_second( );
cout<<"Hello fromthird"<<endl;
}
};

int main( )
{
third obj;
obj.show_third( );
return 0;
}
```

OUTPUT:

Hello from first
Hello from second
Hello from third

Example : 2 Find max of three class data

```
class first
{
protected :
int first_data;
public :
void input_first( )
{
cout<<“Enter firstdata :=”;
cin>>first_data;
}
};
```

```
class second :public first
{
protected :
int second_data;
public :
void input_second( )
{
input_first( );
cout<<“Enter seconddata :=”;
cin>>second_data;
}
};
```

Example : 2 Find max of three class data

```
class third :public second
{
protected :
int third_data;
public :
void input_third( )
{
input_second( );
cout<<"Enter third data:=";
cin>>third_data;
}
void show( )// MEMBER FUNCTION
{
cout<<"First class Data="<<first_data<<endl;
cout<<"Second class Data="<<second_data<<endl;
cout<<"Third class Data="<<third_data<<endl;
}
```

```
int max( )// MEMBER FUNCTION
{
int t1,t2;
t1=first_data>second_data ?first_data
:second_data;
t2=third_data>t1 ?third_data :t1;
return t2;
}
};
```

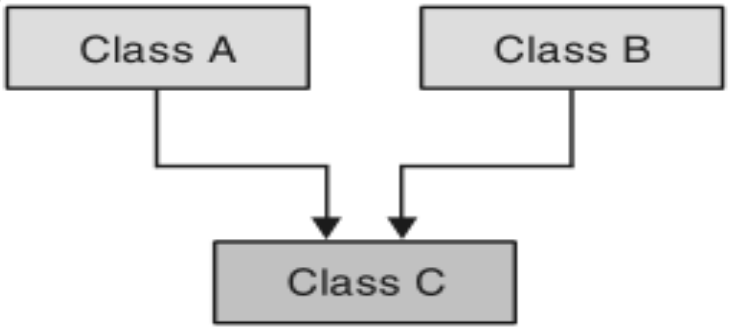
Example : 2 Find max of three class data

```
int main( )
{
        third obj;
        obj.input_third( );
        third.show( );
        cout<<"Max of three data is "<<third.max( )<<endl;
return 0;
}
```

OUTPUT:

Enter First Data:	**10**
Enter Second Data:	**20**
Enter Third Data:	**30**
First class Data :	**10**
Second Class Data:	**20**
Third Class Data:	**30**
Max of three data is:	**30**

Multiple Inheritance



- In multiple inheritance a child can have more than parent i.e., a child can inherit properties from more than one class.

Example 1 : Multiple Inheritance

```
class Internal
{
protected :
int i_marks;
public :
void input_im( )
{
        cout<<"Enter internal marks :=";
        cin>>i_marks;
        if(!(i_marks>=0 && i_marks<=60))
        {
                cout<<"Invalid Marks";
                exit(0);
        }
}
void show_im( )
{
cout<<"Internal marks :="<<i_marks<<endl;
}
};
```

Example 1 : Multiple Inheritance

```
class External
{
protected :
        int e_marks;
public :
void input_em( )
{
        cout<<"Enter external marks :=";
        cin>>e_marks;
        if(!(e_marks>=0 && e_marks<=40))
        {
                cout<<"Invalid Marks";
                exit(0);
        }
}
void show_em( )
{
        cout<<"External marks :="<<e_marks<<endl;
}
};
```

Example 1: Multiple Inheritance

```
class Total :public Internal, public External
{
int total_marks;
public :
void input( )
{
        input_im( );
        input_em( );
}
void show( )
{
        show_im( );
        show_em( );
        total_marks = i_marks+e_marks;
        cout<<"Total Marks :="<<total_marks<<endl;
}
};
```

Example 1: Multiple Inheritance

```
int main( )
{
        Total tm;
        tm.input( );
        tm.show( );
        return 0;
}
```

OUTPUT:

Enter internal marks : =56
Enter external marks : =35
Internal marks : =56
External marks : =35

Total Marks : =91

Example : 2 Ambiguity in multiple Inheritance

```
class A
{
protected :
int num;
public :
void show( )
{
cout<<“num A=”<<num<<endl;
}
};
```

```
class B
{
protected :
int num;
public :
void show( )
{
cout<<“num B=”<<num<<endl;
}
};
```

Example : 2 Ambiguity in multiple Inheritance

```
class C : public A, public B
{
int num;
public :
C( )// involving constructor
{
        A : :num=20;
        B : :num=30;
        num=40;
}
void show( )
{
cout<<"num C="<<num<<endl;
}
};
```

```
int main( )
{
C o1;
o1.A : :show( );
o1.B : :show( );
o1.show( );
return 0;
}
```

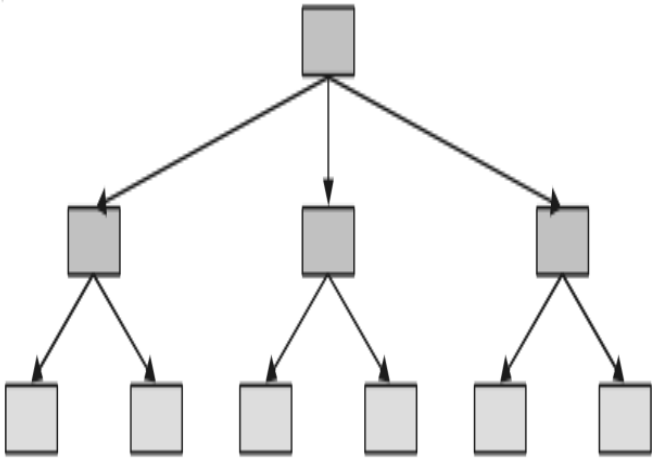
OUTPUT:

Num A: 20

Num B : 30

Num C: 40

Hierarchical Inheritance



```
class A
{

};

class B : public A  class C :  public A  class D :  public A
{ };                { };                 { };
```

- Multiple classes share the same base class. That is number of classes inherits the properties of one common base class. The derived classes again may become base class for other classes.

Example : Hierarchical Inheritance

```
class Father
{
float amount;
public :
Father( )// usage of constructor
{
        amount=50000;
}
float getamount( )
{
        return amount;
}
};
```

Example : Hierarchical Inheritance

```
class Son1  :public Father
{
float amount_son1,total;
public :
Son1( )//usage of constructor
{
        amount_son1=30000;
}
void show( )
{
cout<<"\tGot from father := "<<getamount( )<<endl;
cout<<"\tOwn Investment := "<<amount_son1<<endl;
total = getamount( ) + amount_son1;
cout<<"\tTotal Investment := "<<total<<endl;
}
};
```

Example : Hierarchical Inheritance

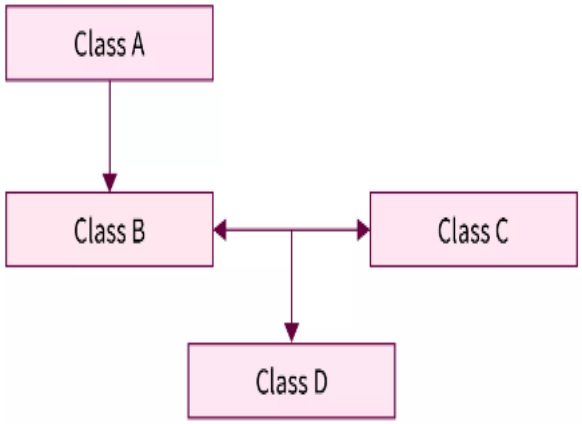
```
class Son2 :public Father
{
float amount_son2,total;
public :
Son2( )//usage of constructor
{
        amount_son2=40000;
}
void show( )
{
cout<<"\tGot from father := "<<getamount( )<<endl;
cout<<"\tOwn Investment := "<<amount_son2<<endl;
total = getamount( ) + amount_son2;
cout<<"\tTotal Investment := "<<total<<endl;
}
};
```

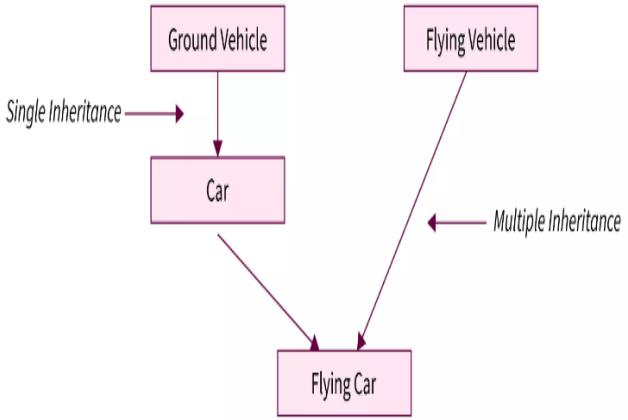
Example : Hierarchical Inheritance

```
int main( )
{
        Son1 S1;
        cout<<"\t Son 1"<<endl;
        S1.show( );
        Son2 S2;
        cout<<"\t Son 2"<<endl;
        S2.show( );
        return 0;
}
```

Son 1
Got from father : = 50000
Own Investment : = 30000
Total Investment : = 80000
Son 2
Got from father : = 50000
Own Investment : = 40000
Total Investment : = 90000

Hybrid Inheritance





- Combination of any of the above types of inheritance.
- It is also referred to as a **multipath inheritance** because many types of inheritances get involved.

Example : Hybrid Inheritance

```
class student
{
Protected:
        int rollno;
public :
        void get_num(int a)
        {
                rollno=a;
        }
        void put_num(void)
        {
                cout<<"Rollno:"<<rollno<<endl;
        }
};
```

Example : Hybrid Inheritance

```
class test : public student
{
Protected:
        float sub1, sub2;
public :
        void get_mark(float x, float y)
        {
                sub1=x;
                sub2=y;
        }
        void put_mark(void)
        {
                cout<<"Marks Obtained:"<<endl;
                cout<<"Sub1:"<<sub1<<endl;
                cout<<"Sub2:"<<sub2<<endl;
        }
};
```

Example : Hybrid Inheritance

```
class sports
{
protected:
        float score;
public :
        void get_score(float s)
        {
                score=s;
        }
        void put_score(void)
        {
                cout<<"Sports mark:"<<score<<endl;
        }
};
```

Example : Hybrid Inheritance

```
class result : public test, public sports
{
        float total;
public :
        void display(void);
};

void result:: display(void)
{
        total= sub1+sub2+score;
        put_num();
        put_mark();
        put_score();
        cout<<"Total score:"<<total<<endl;
}
```

Example : Hybrid Inheritance

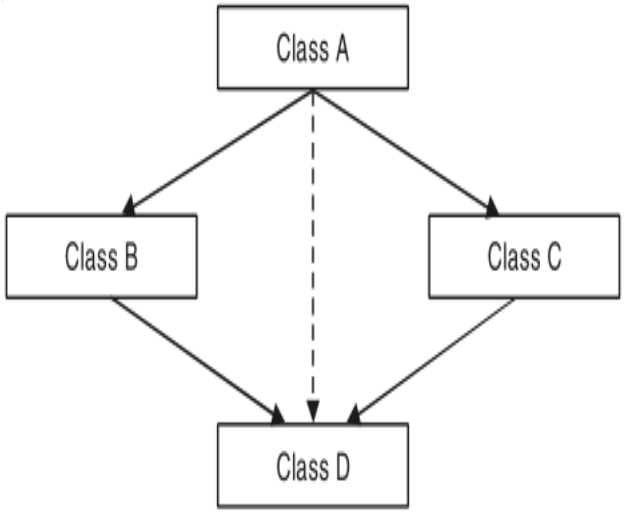
```
int main()
{
        result student1;
        student1.get_num(1111);
        student1.get_mark(60.0,70.0);
        student1.get_score(7.0);
        student1.display();
        return 0;
}
```

OUTPUT:

Roll no	**:**	**1111**
Marks Obtained	**:**	
Sub1	**:**	**60.0**
Sub2	**:**	**70.0**
Sports Mark	**:**	**7.0**
Total Score	**:**	**137.0**

Multipath Inheritance - Virtual Base Class

- Consider the situation where we have one class A. This class A is inherited by two other classes B and C. Both these classes are inherited in a new class D.
- This is as shown in figure given below.




```
class A
{
};
class B :public A
{
};
class C : public A
{
};
class D :public C, public B
{
};
```

Multipath Inheritance - Virtual Base Class

- Data members/ functions of class A are inherited twice to class D. One through class B and second through class C.
- When any data/ function members of class A is accessed by an object of class D, **ambiguity arises as to which data/function members would be called ?**
- One inherited through B or the other inherited through C.
- It confuses compiler and it flashes error message. To resolve this ambiguity when class A is inherited in both class B and class C, it is declared as virtual base class by placing the **keyword virtual.**

```
Class B : virtual public A        class C : public virtual A
{                                 {


};                                };
```

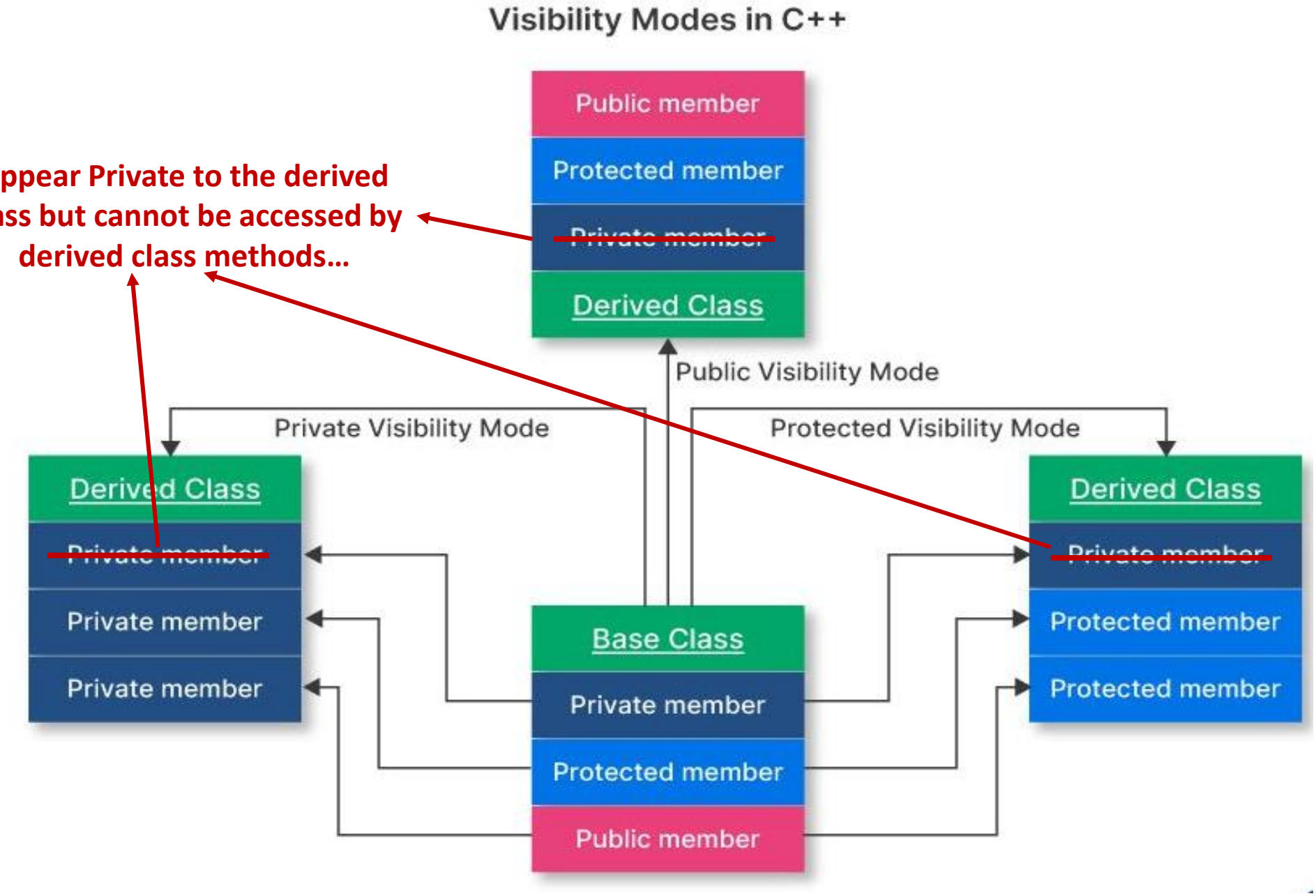
Multipath Inheritance - Virtual Base Class

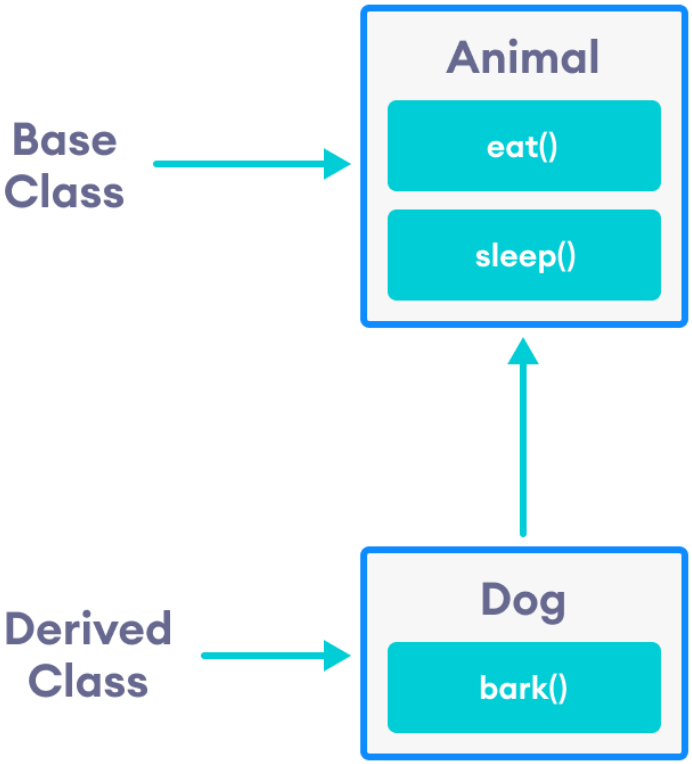
```
class A
{
public :
void show( )
{
cout<<"hello from show of A"<<endl;
}
};

class B :public virtual  A
{
};
class C :virtual public A
{
};
class D : public B, public C
{
};
```

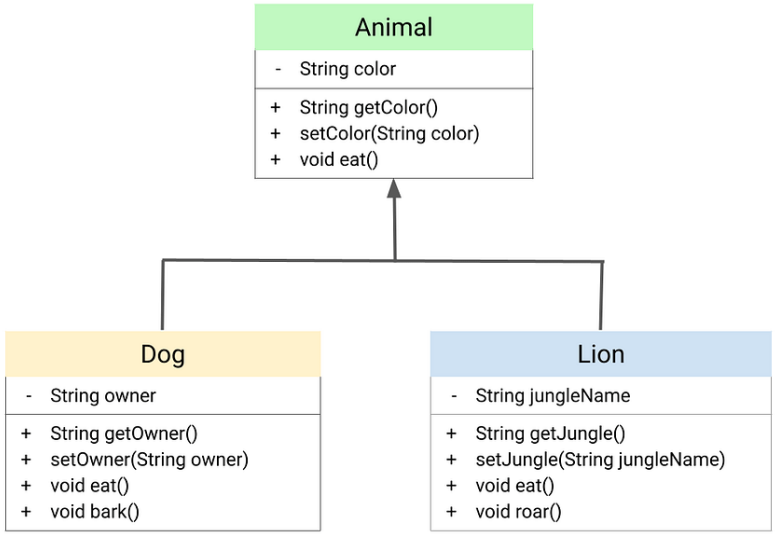
```
int main( )
{
D obj;
obj.show( );
return 0;
}
```

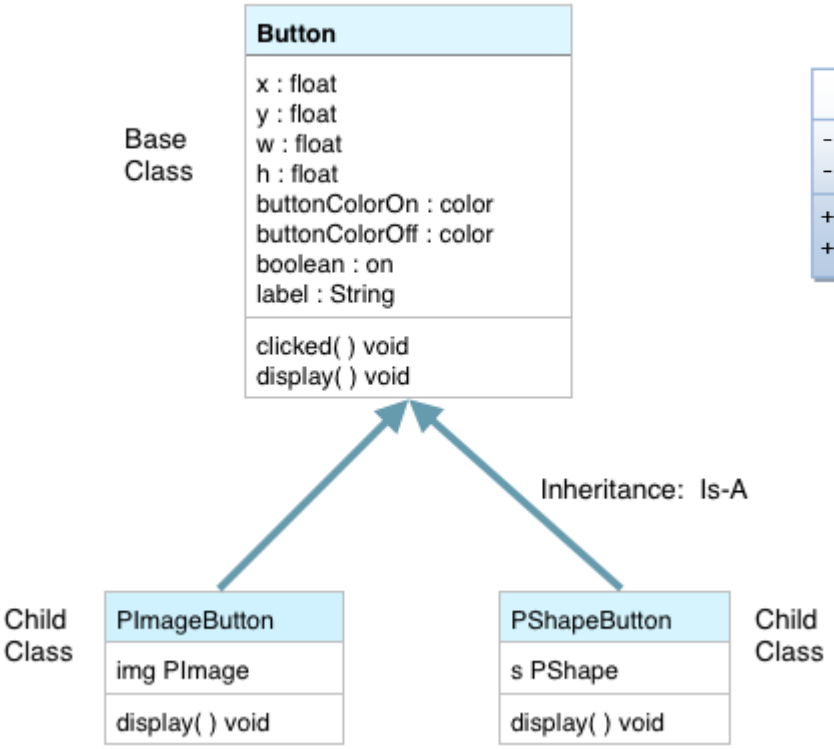
Only one copy of A will be inherited

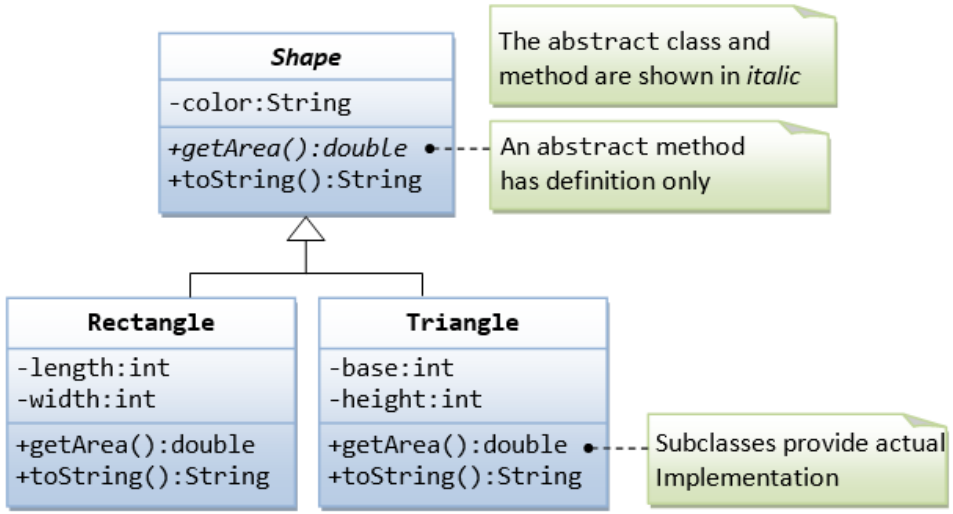
OUTPUT:
Hello from show of A

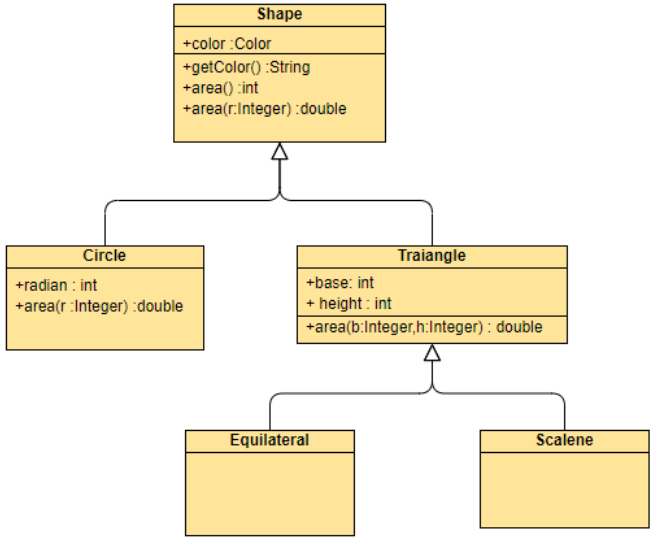

Visibility Modes in C++
Public member
Protected member
Private member
Derived Class
ppear Private to the derived
ss but cannot be accessed by
derived class methods...
Public Visibility Mode
Private Visibility Mode
Protected Visibility Mode
Derived Class
Private member
Private member
Private member
Base Class
Private member
Protected member
Public member
Derived Class
Private member
Protected member
Protected member

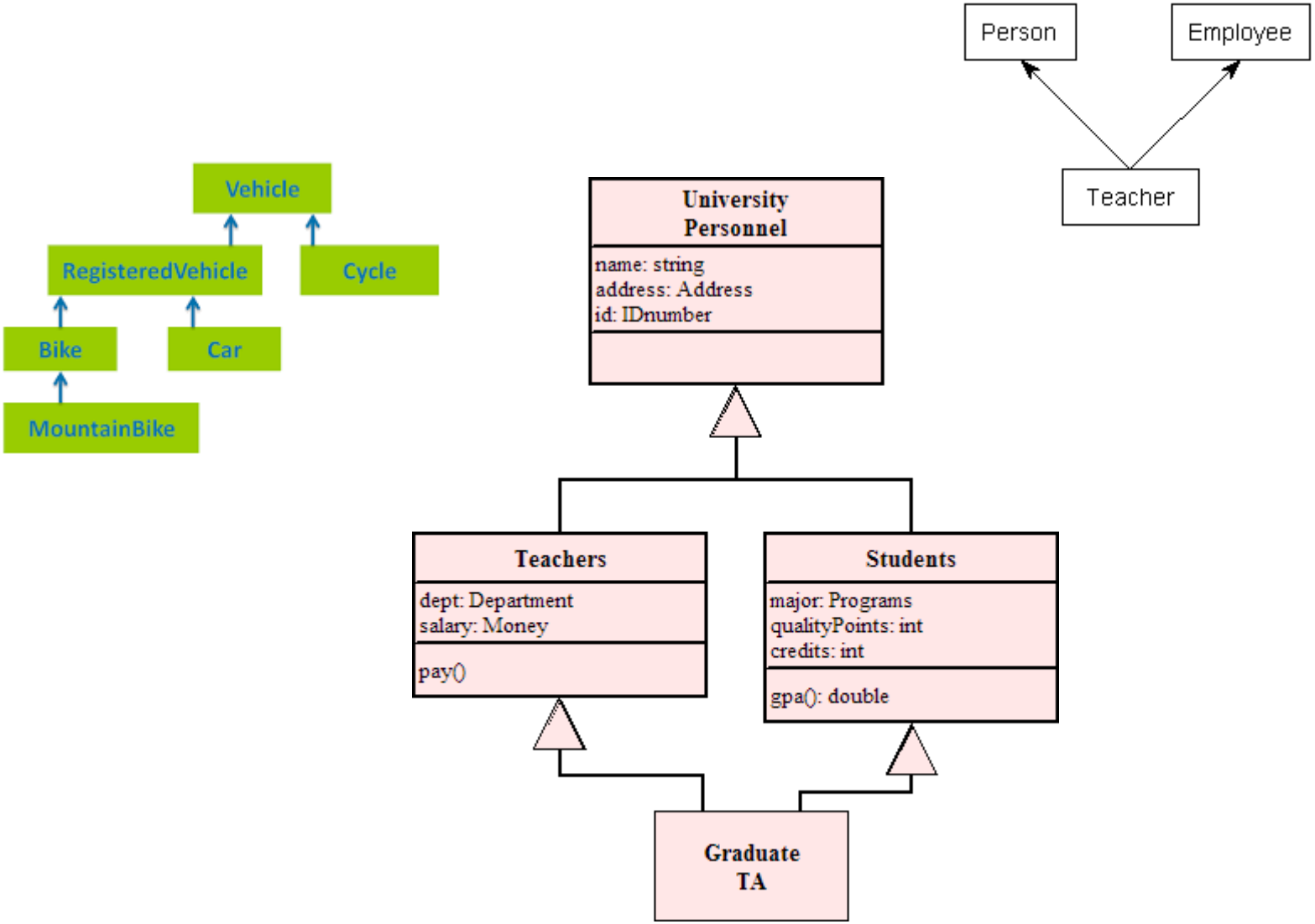

Base Class
Animal
eat()
sleep()
Derived Class
Dog
bark()

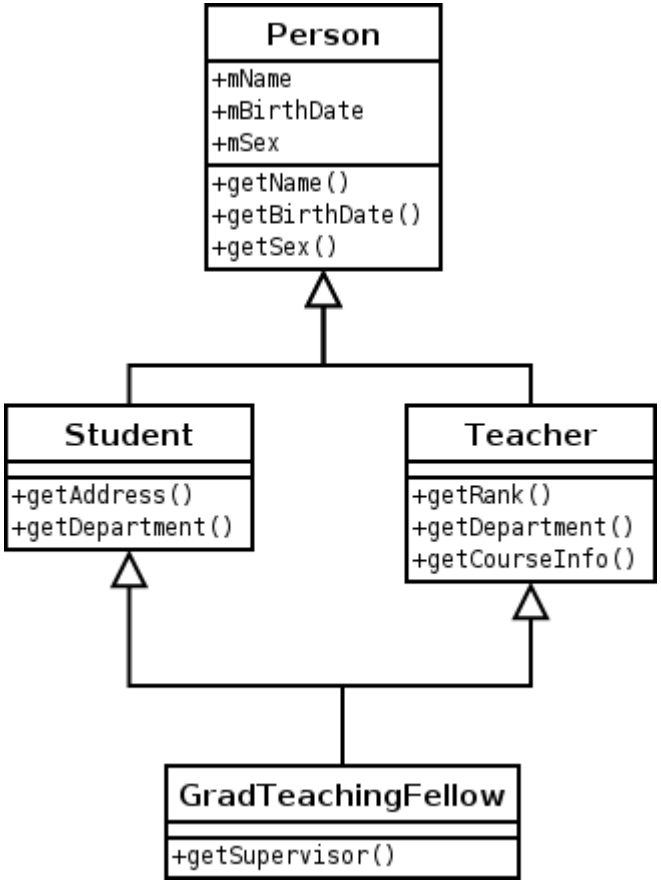

Animal
- String color
+ String getColor()
+ setColor(String color)
+ void eat()
Dog
- String owner
+ String getOwner()
+ setOwner(String owner)
+ void eat()
+ void bark()
Lion
- String jungleName
+ String getJungle()
+ setJungle(String jungleName)
+ void eat()
+ void roar()

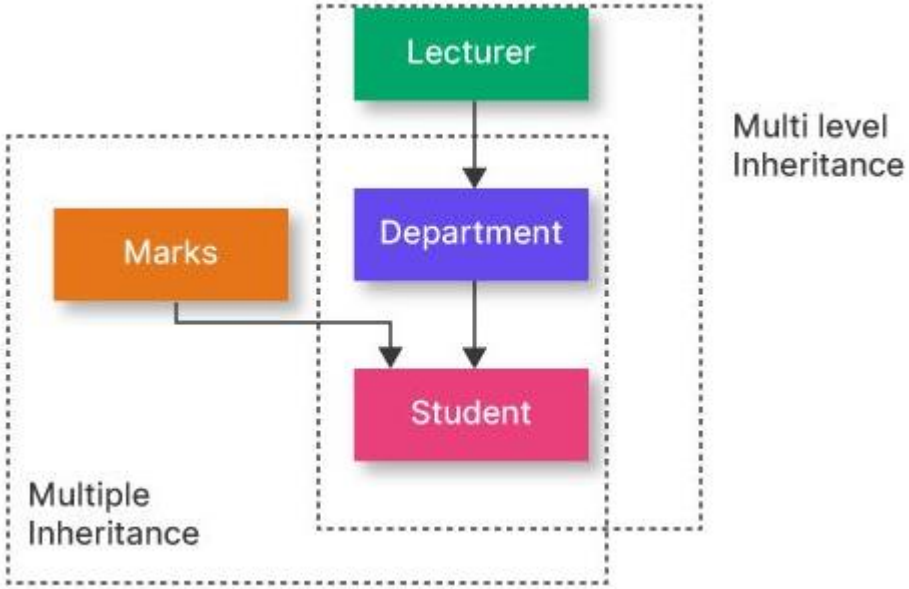

Button
x : float
y : float
w : float
h : float
buttonColorOn : color
buttonColorOff : color
boolean : on
label : String
clicked( ) void
display( ) void
Base Class
Inheritance: Is-A
PImageButton
img PImage
display( ) void
Child Class
PShapeButton
s PShape
display( ) void
Child Class


Shape
-color:String
+getArea():double
+toString():String
The abstract class and method are shown in italic
An abstract method has definition only
Rectangle
-length:int
-width:int
+getArea():double
+toString():String
Triangle
-base:int
-height:int
+getArea():double
+toString():String
Subclasses provide actual Implementation


Shape
+color :Color
+getColor() :String
+area() :int
+area(r:Integer) :double
Circle
+radian : int
+area(r :Integer) :double
Traiangle
+base: int
+ height : int
+area(b:Integer,h:Integer) : double
Equilateral
Scalene


Person
Employee
Teacher
Vehicle
RegisteredVehicle
Cycle
Bike
Car
MountainBike
University Personnel
name: string
address: Address
id: IDnumber
Teachers
dept: Department
salary: Money
pay()
Students
major: Programs
qualityPoints: int
credits: int
gpa(): double
Graduate TA


Person
+mName
+mBirthDate
+mSex
+getName()
+getBirthDate()
+getSex()
Student
+getAddress()
+getDepartment()
Teacher
+getRank()
+getDepartment()
+getCourseInfo()
GradTeachingFellow
+getSupervisor()


Lecturer
Department
Marks
Student
Multi level
Inheritance
Multiple
Inheritance