INHERITENCE

By J.V.PATIL

INTRODUCTION TO INHERITANCE

- Reusability is another important feature of OOP.
- It is always nice if we could reuse something that already exists rather than trying to create the same all over again.
- It would not only save time and money but also reduce frustration and increase reusability.
- For instance, the reuse of a class that has already been tested, debugged and used many times can save us the effort of developing and testing the same again.
- C++ supports the concept of reusability.
- The C++ classes can be reused in several ways. Once a class has been written and tested, it can be adapted by other programmers to suit their requirements.
- This is basically done by creating new classes, reusing the properties of existing ones.

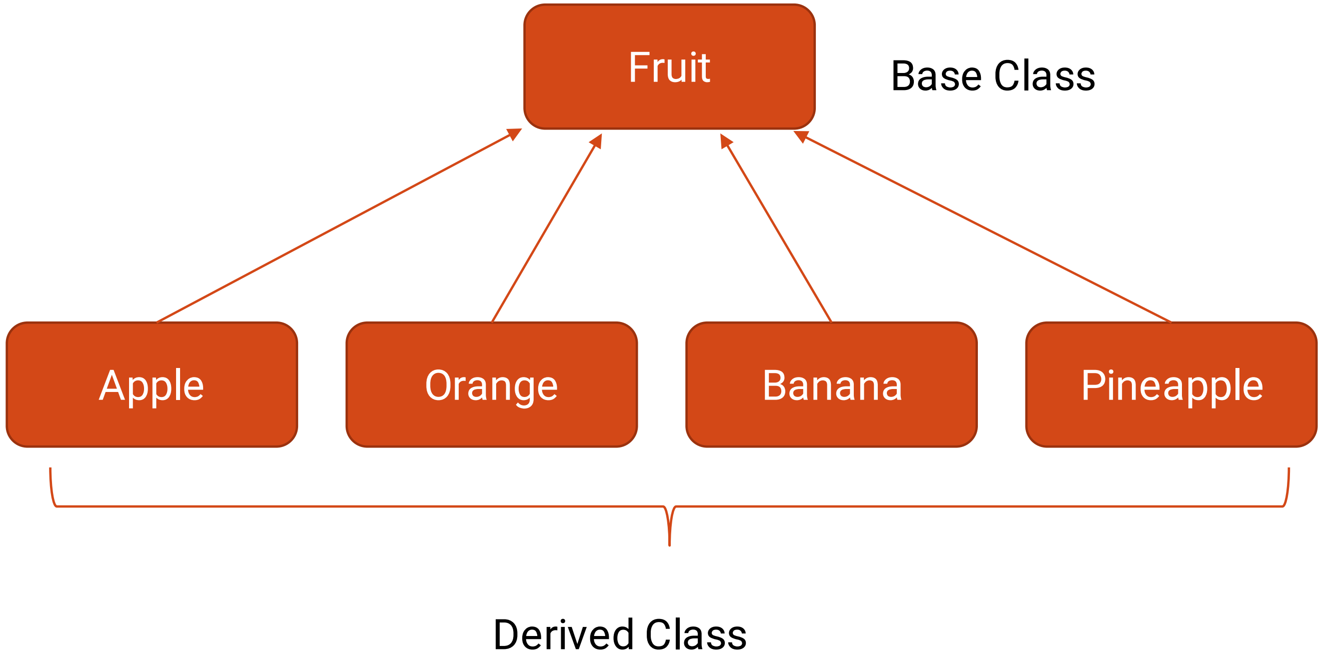
- The mechanism of deriving a new classes (called as derived class) from

INTRODUCTION TO INHERITANCE

- **Base Class**: A base class is a class in Object-Oriented Programming language, from which other classes are derived. The class which inherits the base class has all members of a base class as well as can also have some additional properties. The Base class members and member functions are inherited to Object of the derived class. A base class is also called **parent class** or **superclass**.

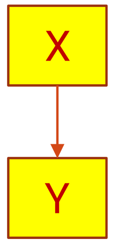
- **Derived Class**: A class that is created from an existing class. The derived class inherits all members and member functions of a base class. The derived class can have more functionality with respect to the Base class and can easily access the Base class. A Derived class is also called a **child class** or **subclass**.

For example:-

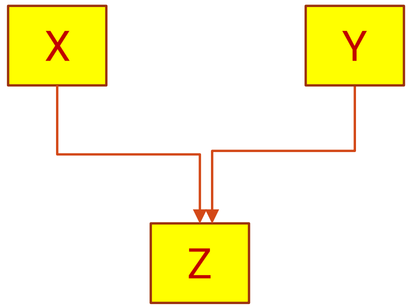

Fruit
Base Class
Apple
Orange
Banana
Pineapple
Derived Class

- In OOP, it is possible for a base class to have one or many derived classes and for one derived class to have one or many base classes. This gives different forms of inheritance.

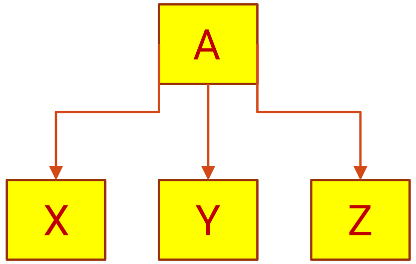
1. Single Inheritance: A derived class with only one base class is called single inheritance.



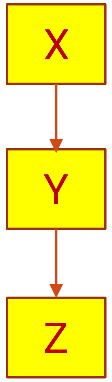
2. Multiple Inheritance: A derived class with more than one base class is called multiple inheritance.



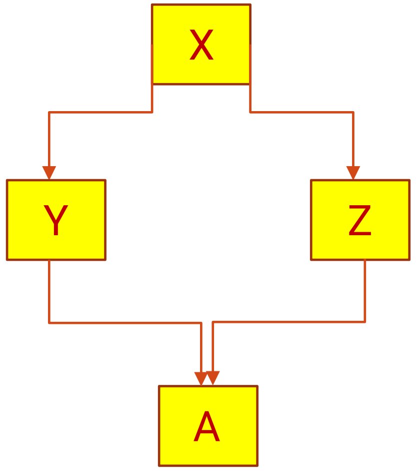
3. Hierarchical Inheritance: If one base class is inherited by more than one derived classes, it is called hierarchical inheritance.



4. Multilevel Inheritance: The mechanism of deriving a class from another derived class is known as multilevel inheritance.



5. Hybrid Inheritance: One class is inherited by many derived classes which are again base class for other derived classes. This form of inheritance is called hybrid inheritance.


$$\begin{tikzcd} & X \arrow[dl] \arrow[dr] & \\ Y \arrow[dr] & & Z \arrow[dl] \\ & A & \end{tikzcd}$$

DEFINING DERIVED CLASSES

- When a class inherits another, the member of the base class becomes members of the derived class.
- The Syntax of derived class definition:

```
class derived_class_name: visibility_mode base_class_name
{
    //Members of derived class
};
```

- The colon (:) indicates that the derived_class_name is derived from the base_class_name.
- The visibility mode is optional. The default visibility mode is private.
- Visibility mode specifies whether the features of the base class are privately derived or publicly derived.

DEFINING DERIVED CLASSES

- For example:

```
class ABC: private xyz        //private derivation
{
// body of derived class
};
class ABC: public xyz         //public derivation
{
//body of derived class
};
class ABC:protected xyz       //protected derivation
{
//body of derived class
};
```

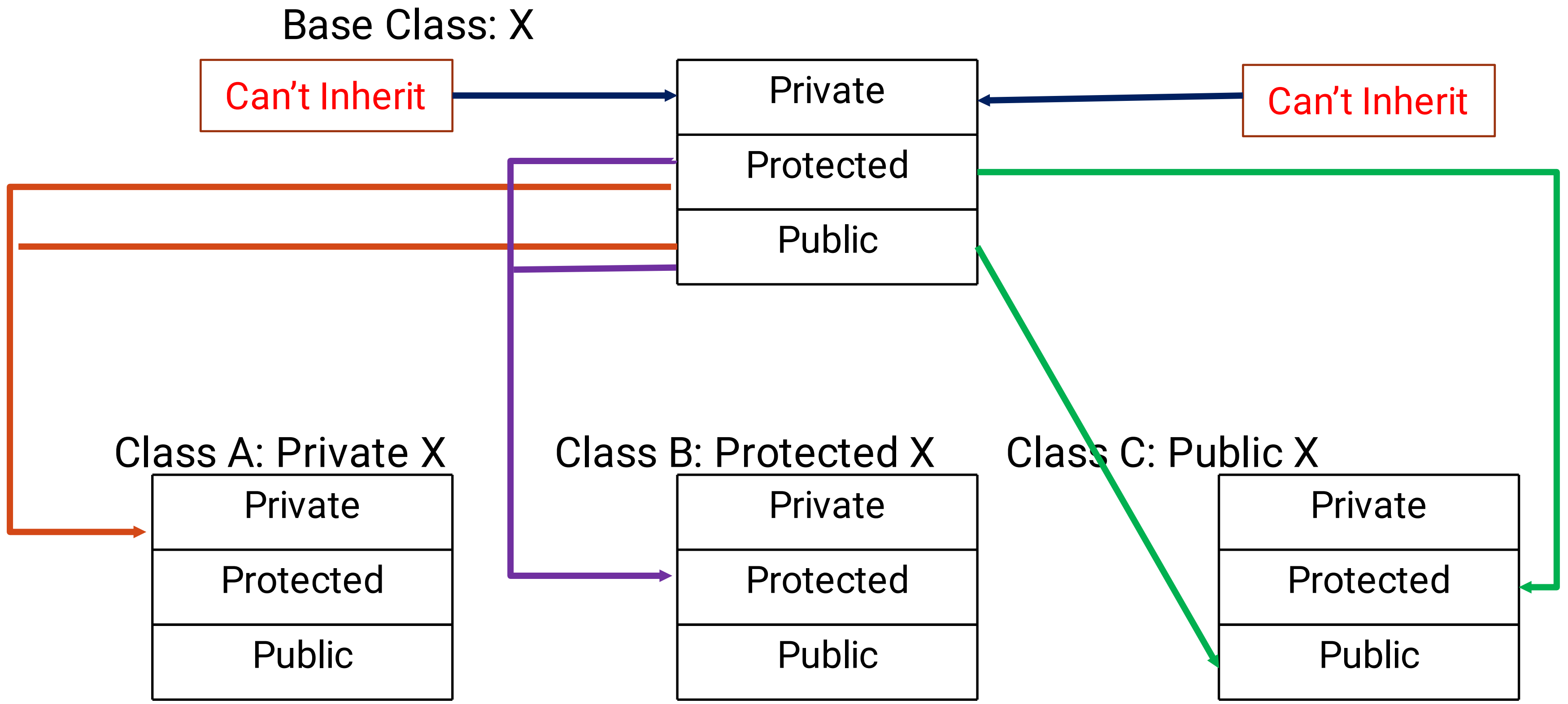
VISIBILITY MODES AND EFFECTS

- An important concept in inheritance is that of finding out when a member function in the base class can be used by objects of the derived class. This is called accessibility or visibility.
- A visibility mode is a feature in inheritance that decides the accessibility of inherited features of a derived class, i.e. whether the inherited features will be accessible from outside the class or not.
- It also decides whether the inherited features can be further inherited by another class or not.
- Access modifiers or access specifiers in a class are used to set the accessibility of the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions.
- The visibility mode can be private or public or protected. The visibility mode decides whether the inherited features of base class will be public or private or protected members of derived class.

VISIBILITY MODES AND EFFECTS

- Public Visibility Mode: If visibility mode is public, then the public members of base class will go in the public section and the protected members will become part of the protected section of the derived class. Private members of base class cannot be inherited.
- Private Visibility Mode: If the visibility mode is written as private then all public and protected members of base class will become private members of the derived class.
- Protected Visibility Mode: If the visibility mode is protected then all members inherited will be protected members of the derived class.

VISIBILITY MODES



VISIBILITY MODES

Base class visibility	Derived Class Visibility		
	Public derivation	Private derivation	Protected derivation
Private	Not inherited	Not inherited	Not inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

Table : Visibility of inherited members

TYPES OF INHERITANCE

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

SINGLE INHERITANCE

- When one base class is derived by one child class, the it is known as single inheritance.
- The existing base class is known as direct base class whereas, they newly created class is called as singly derived class.
- Syntax:

```
class  base_class_name
{
    //body of base class
};
class derived_class_name : visibility_mode base_class_name
{
    //body of derived class
};
```

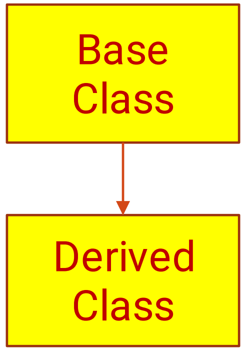
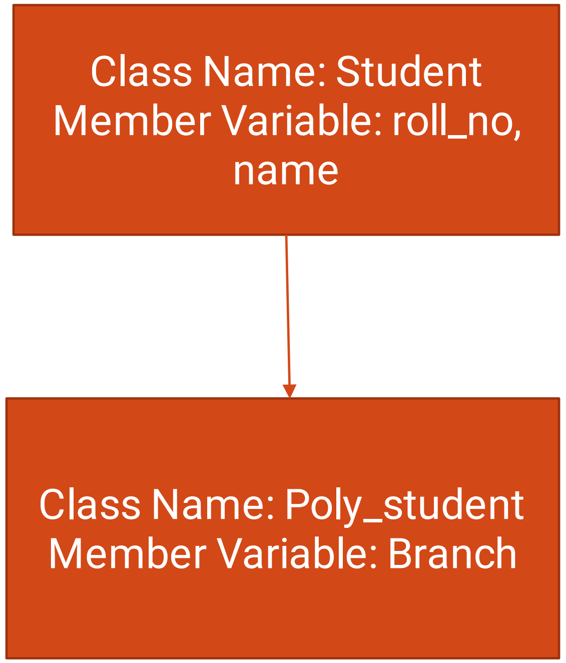


Fig. Single Inheritance

SINGLE INHERITANCE EXAMPLE

- Assume suitable member functions.



MULTIPLE INHERITANCE

- When more than one base classes are inherited by a derived class, then such type of inheritance is called as multiple inheritance.
- Syntax:

```
 class base_class1
 {
     //body of base_class1        };
 class base_class2
 {
     //body of base_class2            };
 …..
 class base_classn
 {
     //body of base_classn        };
 class derived_class_name : visibility_mode base_class1, visibility_mode base_class2, …,visibility_mode base_classn
 {
     //body of derived class            };
```

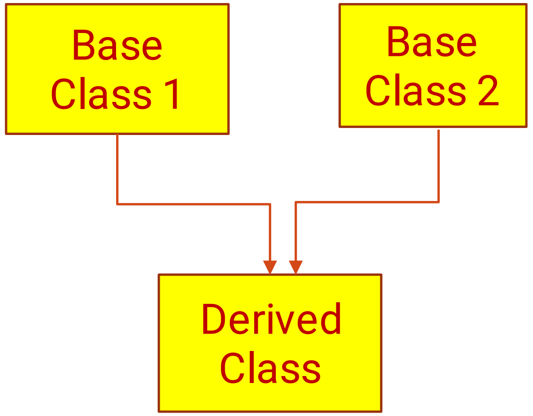
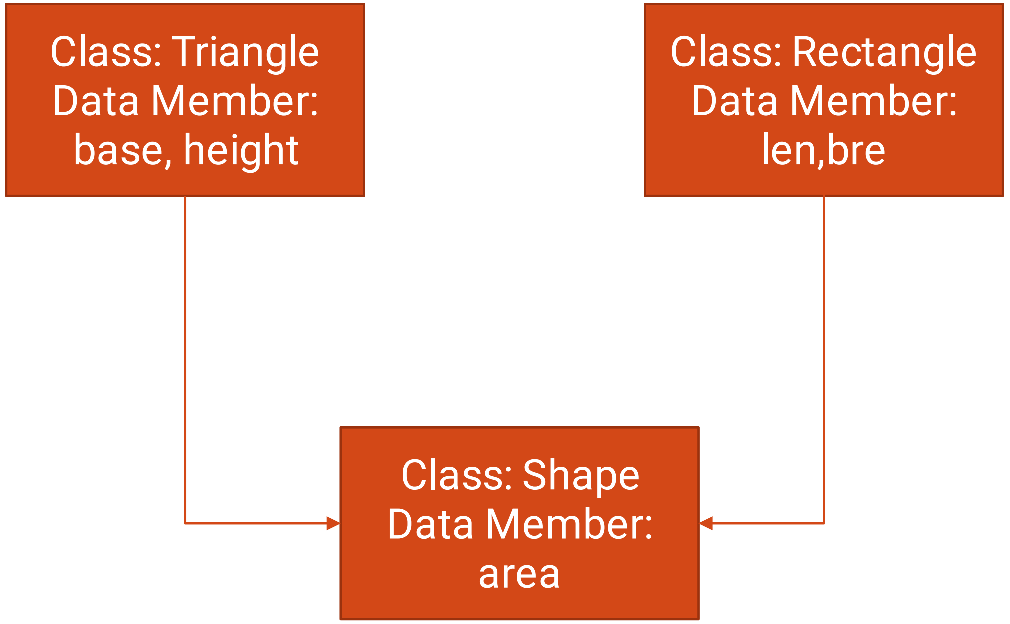


Figure. Multiple Inheritance

MULTIPLE INHERITANCE EXAMPLE



MULTILEVEL INHERITANCE

- When a base class is derived by a child class which further derived by another child class, then it is known as multilevel inheritance.
- Syntax:

```
class base_class_name
{
    //body of base class
};
class intermediate_class_name: visibility_mode base_class_name
{
    //body of intermediate class
}
class child_class_name : visibility_mode intermediate_class_name
{
    //body of child class name
}
```

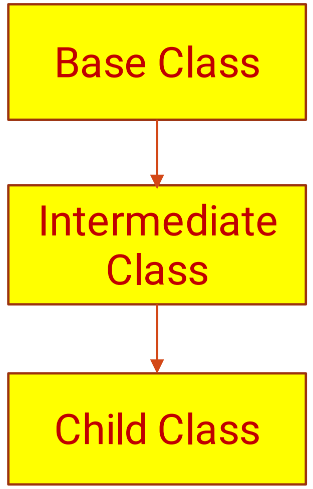
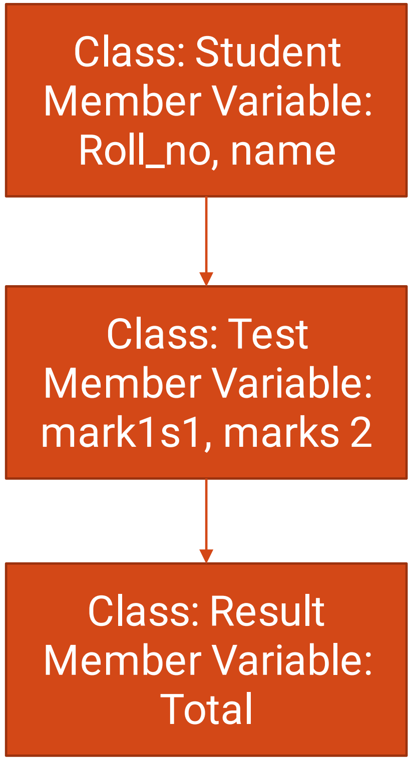


Fig. Multilevel Inheritance

MULTILEVEL INHERITANCE EXAMPLE

- Assume suitable member functions.



HIERARCHICAL INHERITANCE

- When one base class is inherited by more than one derived classes, it is known as hierarchical inheritance.
- The base class will include all the features that are common to the subclasses.
- A subclass can be constructed by inheriting the properties of the base class.

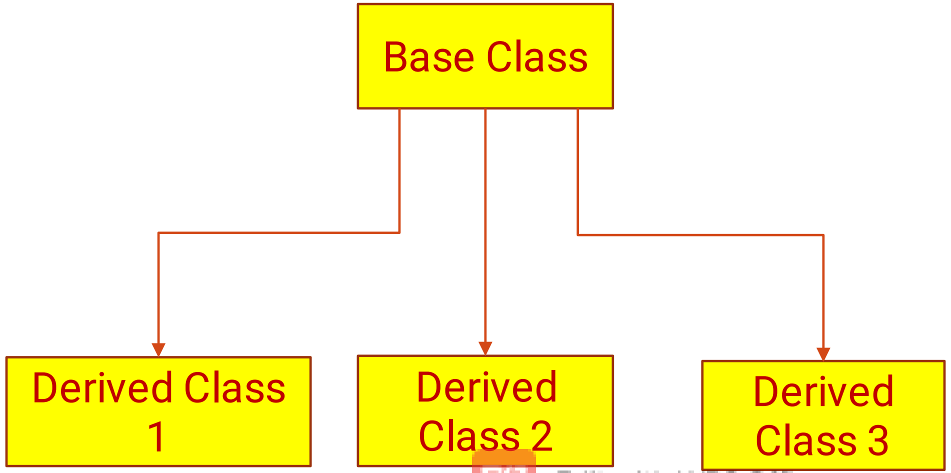


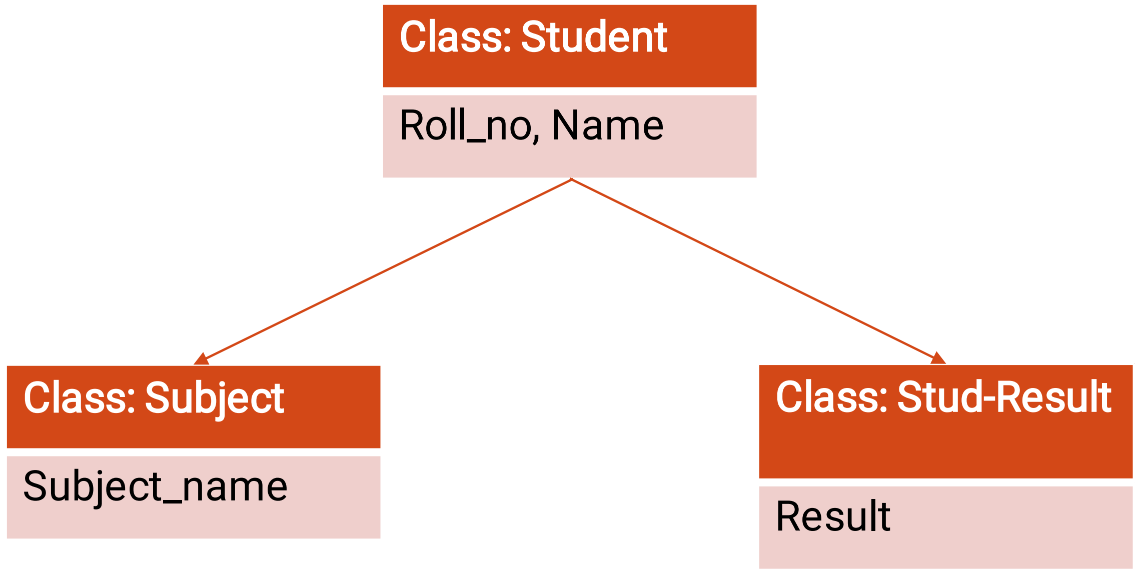
Fig. Hierarchical Inheritance

HIERARCHICAL INHERITANCE

- Syntax

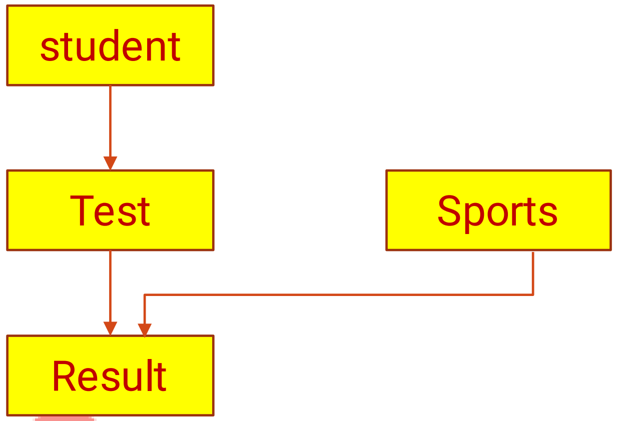
```
class base_class_name
{
    //body of base class
};
 class derived_class1: visibility_mode base_class_name
{
    //body of derived_class1
};
 class derived_class2: visibility_mode base_class_name
{
    //body of derived_class2
};
 class derived_class3: visibility_mode base_class_name
{
    //body of derived_class3
};
```

HIERARCHICAL INHERITANCE EXAMPLE

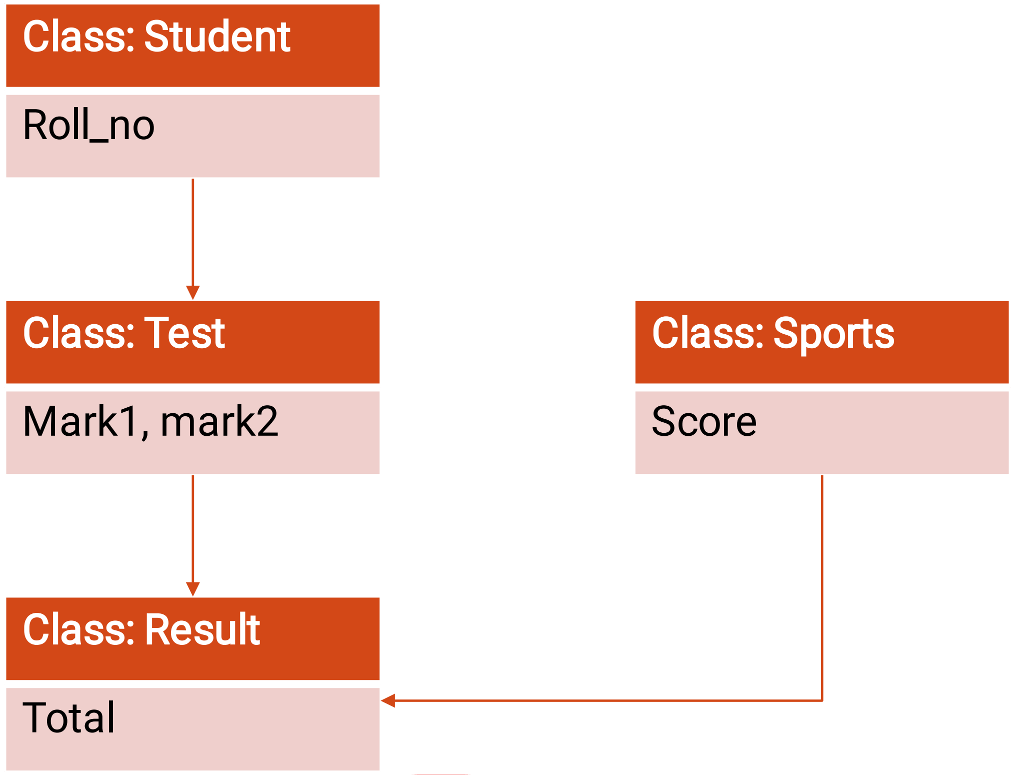


HYBRID INHERITANCE

- Sometimes, we may require to combine two or more types of inheritance to design a program the result is known as hybrid inheritance.
- Example: Consider an example where we have three classes student, test and result.
- But authority wants that some weightage should be given for sports to finalise the result.



HYBRID INHERITANCE EXAMPLE



VIRTUAL BASE CLASS

- If it is a situation where multilevel, multiple and hierarchical inheritance, are required(as shown in following diagram) then here child class has two direct base classes (parent1 and parent 2) which themselves have a common base class (grandparent).
- Here child inherits grandparent via parent1 and parent 2.
- It can also inherits grandparent directly(as shown in fig by dotted line).
- As child inherit grandparents via two paths, child would have duplicate sets of members inherited from grandparent which introduces ambiguity. This ambiguity is avoided by making common base class as virtual base class.

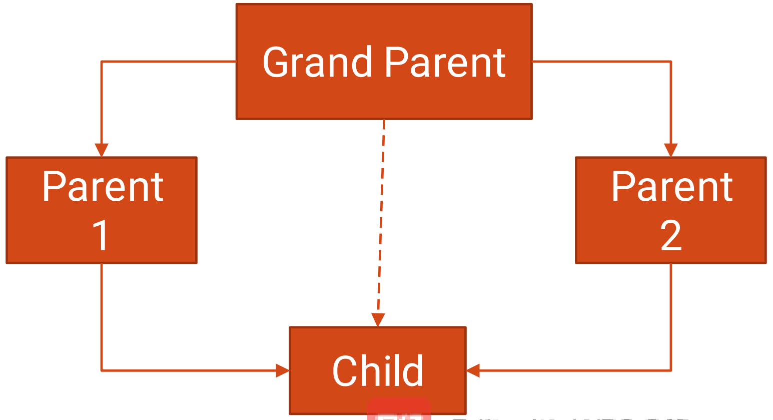


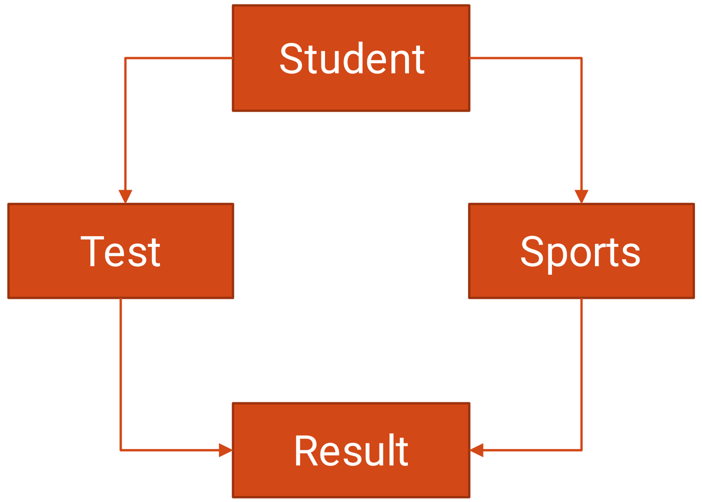
Fig. Multipath Inheritance

VIRTUAL BASE CLASS

- Inheritance by the ‘child’ might pose some problems. All the protected and public members of ‘grandparent’ are inherited into child twice, first via parent1 and again via parent2. This means child would have duplicate sets of the members inherited from grandparent. This introduces ambiguity and should be avoided.
- The duplication of inherited members due to these multiple paths can be avoided by making the common base class as virtual base class while declaring the direct or intermediate base classes which is shown as follow:

```
class Grandparent
{
};
class Parent1 : virtual public Grandparent
{
};
class Parent2: virtual public Grandparent
{
};
class child: public Parent1, public Parent2
{
};
```

VIRTUAL BASE CLASS EXAMPLE

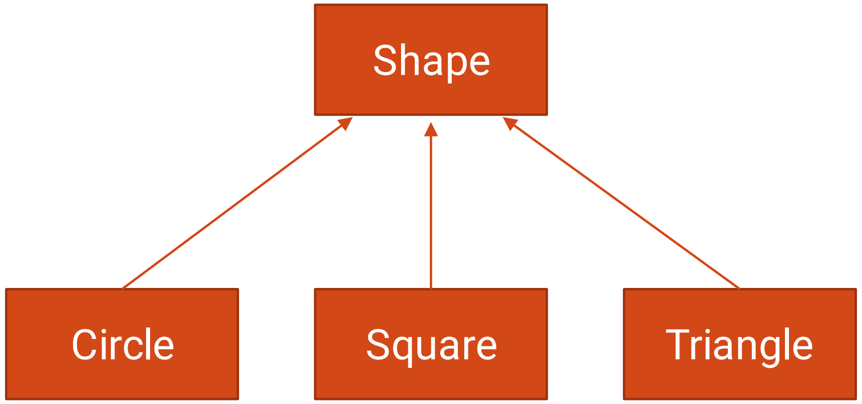


ABSTRACT CLASSES

- A class that contains at least one pure virtual function is said to be abstract. A pure virtual function is a member function i.e., declared in an abstract class, but defined in a derived class.
- An abstract class does not create any object and it contains one or more functions for which there is no definitions.
- A class used only for deriving other classes is called the abstract class. An abstract class is designed only to act as base class, i.e. to be inherited by other classes.
- A class containing a pure virtual function is called an abstract class. It is called abstract because we cannot define objects of a class containing a pure virtual function. It exists only for the purpose of defining classes which are derived from it.
- A pure virtual function is one which **must be overridden** by any non-abstract derived class. This is indicated in the declaration with the syntax **"=0"** in the member function's declaration.

ABSTRACT CLASSES

- An abstract class is used only to derive other classes. For example: Shape is abstract base class which derives square, circle and rectangle.



CONSTRUCTORS IN DERIVED CLASSES

- Constructors are used to initialize objects.
- As long as a base class constructor takes any arguments, the derived class need not have constructor function.
- However, if any base class contains a constructor with one or more arguments, then it is mandatory for the derived class to have a constructor and pass the arguments to the base class constructor.
- When both the derived and base classes contain constructors, the base constructor is executed first and then constructor in the derived class is executed.
- Since derived class takes the responsibility of supplying initial values to its base classes, we supply the initial values that are required by all the classes together when a derived class object is declared.
- Arguments are passes to the base class constructor.
- The constructor of the derived class receives the entire list of values as its arguments and passes them on to the base constructors in the order in which they are declared in the derived class.

CONSTRUCTORS IN DERIVED CLASSES

- The base constructor are called and executed before executing the statements in the body of the derived constructor.
- Syntax of defining a derived constructor:

```
Derived Constructor(arglist1, arglist2,……,arglistN, arglistD):
Base1 (arglist1);
Base2(arglist2);
……..
BaseN(arglistN);
{
//Body of derived construtor
}
```

CONSTRUCTORS IN DERIVED CLASSES

- The header line of derived constructor function contain two separate parts separated by colon (:).
- The first part provides the declaration of the arguments that are passed to the derived constructor and the second part lists the function calls to the base constructor.
- Base1(arglist1), Base2(arglist2) ,…… are the function calls to the base constructors Base1(),Bsae2(), …. and therefore arglist1, arglist2, …. etc represents the actual parameters that are passed to the base constructor.
- Arglist1 through arglistN are argument declaration for the base constructor base1 through baseN.
- ArglistD provides the parameter that are necessary to initialize the members of derived class.
- The constructor for virtual base classes are invoked before any non-virtual base classes.

THANK YOU!!!