INHERITANCE IN C++

WHAT IS AN INHERTANCE?

- Inheritance is the process by which new classes called derived classes are created from existing classes called base classes.
- The derived classes have all the features of the base class and the programmer can choose to add new features specific to the newly created derived class.
- The idea of inheritance implements the **is a relationship**. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.

WHAT IS AN INHERTANCE? contd...

MAMMAL

All mammals have certain characteristics.

Dog *is a* mammal. It has all features of mammals in addition to its own unique features

Cat *is a* mammal. It has all features of mammals in addition to its own unique features

FEATURES /ADVANTAGES OF INHERITANCE

- Reusability of Code
- Saves Time and Effort
- Faster development, easier maintenance and easy to extend
- Capable of expressing the inheritance relationship and its transitive nature which ensures closeness with real world problems .

SYNTAX

To create a derived class from an already existing base class the syntax is:

```
class derived-class: access-specifier base-class
{

....

}
```

Where access specifier is one of public, protected, or private.

SYNTAX contd.....

For example, if the base class is *animals* and the derived class is *amphibians* it is specified as:

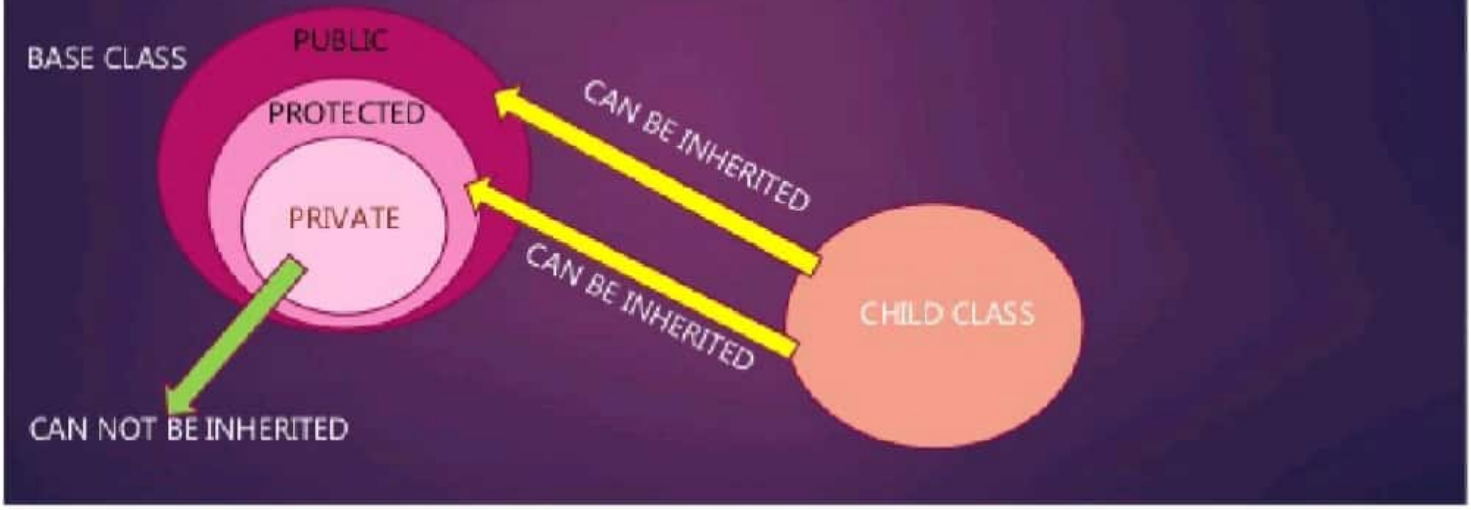
```
class animals  //base class
{

  .......
};

class amphibians : public animals
{                          //derived class

 .....
};
```

In this example class amphibians have access to both public and protected members of base class animals.

NOTE: A class can be derived from more than one class, which means it can inherit data and functions from multiple base classes. In that case a class derivation lists names of one or more base classes each separated by comma.

ACCESS CONTROL AND INHERITENCE

- A derived class can access all the protected and public members of its base class.
- It can not access private members of the base class.



ACCESS CONTROL AND INHERITENCE contd...

We can summarize the different access types according to who can access them in the following way:

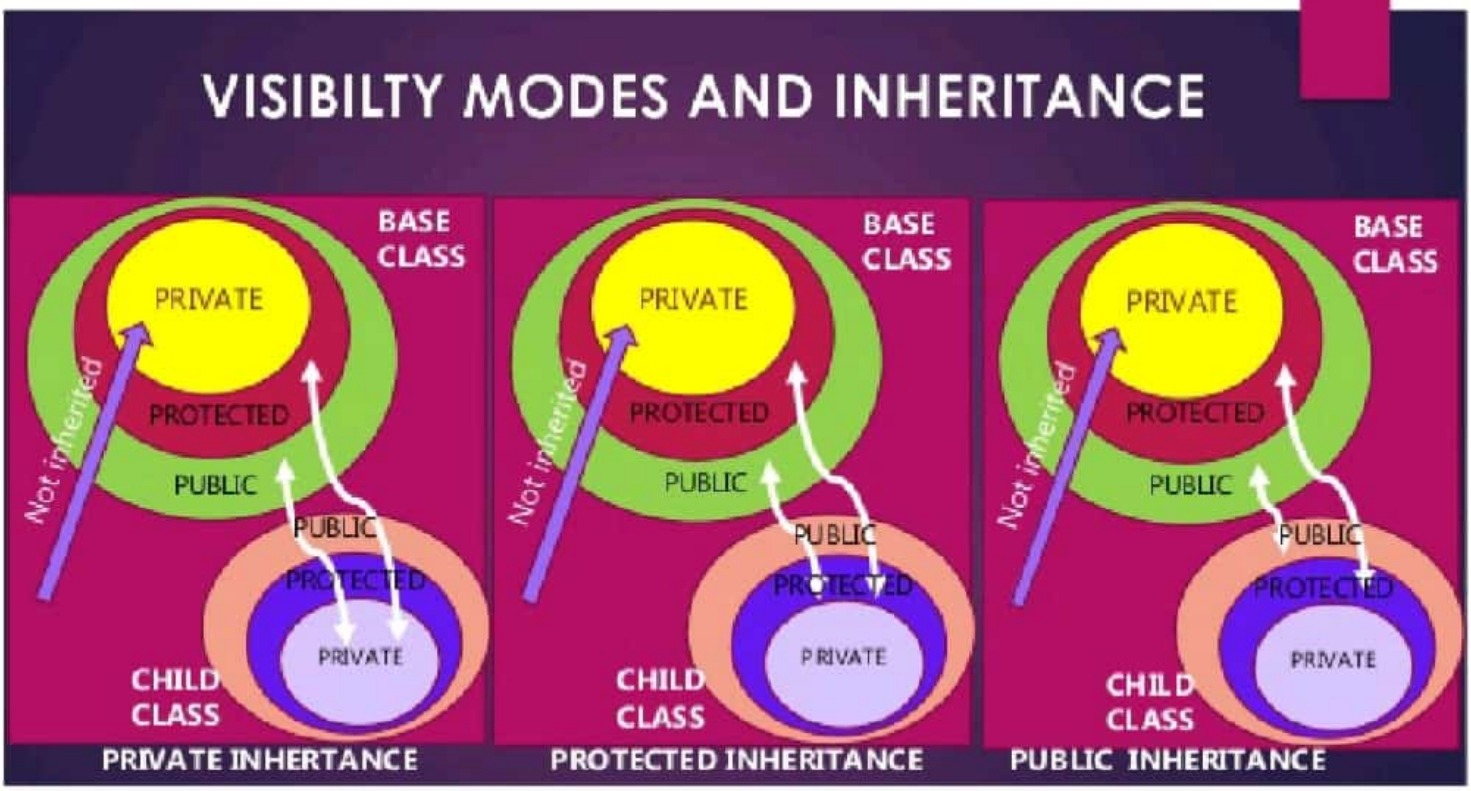
Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

NOTE: Constructors and destructors of the base class are never inherited.

VISIBILTY MODES AND INHERITANCE

A child class can inherit base class in three ways. These are:

Members of base class / Inheritance type	PRIVATE	PROTECTED	PUBLIC
PRIVATE	NOT INHERITED	Become private of child class	Become private of child class
PROTECTED	NOT INHERITED	Become protected members of child class	Become protected members of child class
PUBLIC	NOT INHERITED	Become protected members of child class	Become public members of child class



PRIVATE INHERITANCE

In private inheritance protected and public members of the base class become the private members of the derived class.

```
class base
{
private:
int a;
void funca();
protected:
int b;
void funcb();
public:
int c;
void funcc();
}
```

Private inheritance

```
class child : private base
{
private:
int x;
void funcx();
protected:
int y;
void funcy();
public:
int z;
void funcz();
}
```

```
class child
{
private:
int x;
void funcx();
int b;
void funcb();
int c;
void funcc();
protected:
int y;
void funcy();
public:
int z;
void funcz();
}
```

Protected members inherited from base class

Public members inherited from base class

New child class after inheritance

PROTECTED INHERITANCE

In protected inheritance protected and public members of the base class become the protected members of the derived class.

```
class base
{
private:
int a;
void funca();
protected:
int b;
void funcb();
public:
int c;
void funcc();
}
```

Protected inheritance

```
class child : protected base
{
private:
int x;
void funcx();
protected:
int y;
void funcy();
public:
int z;
void funcz();
}
```

```
class child
{
private:
int x;
void funcx();
protected:
int y;
void funcy();
int b;
void funcb();
int c;
void funcc();
public:
int z;
void funcz();
}
```

Protected members inherited from base class

Public members inherited from base class

New child class after inheritance

PUBLIC INHERITANCE

In protected inheritance protected members become the protected members of the base class and public members of the base class become the public members of the derived class.

```
class base
{
private:
int a;
void funca();
protected:
int b;
void funcb();
public:
int c;
void funcc();
}
```

Public inheritance

```
class child : public base
{
private:
int x;
void funcx();
protected:
int y;
void funcy();
public:
int z;
void funcz();
}
```

```
class child
{
private:
int x;
void funcx();
protected:
int y;
void funcy();
int b;
void funcb();
public:
int z;
void funcz();
int c;
void funcc();
}
```

Protected members inherited from base class

Public members inherited from base class

New child class after inheritance

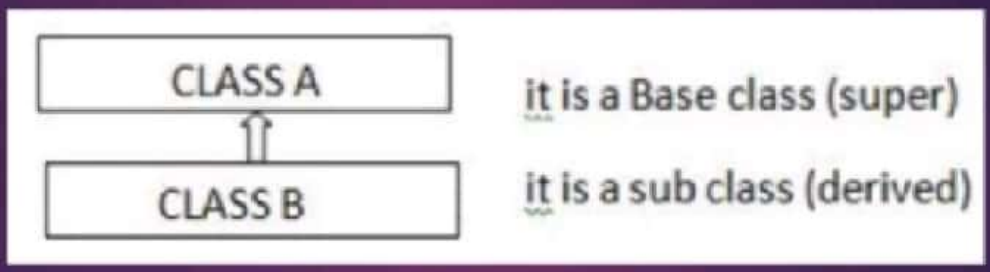
TYPES OF INHERITANCE

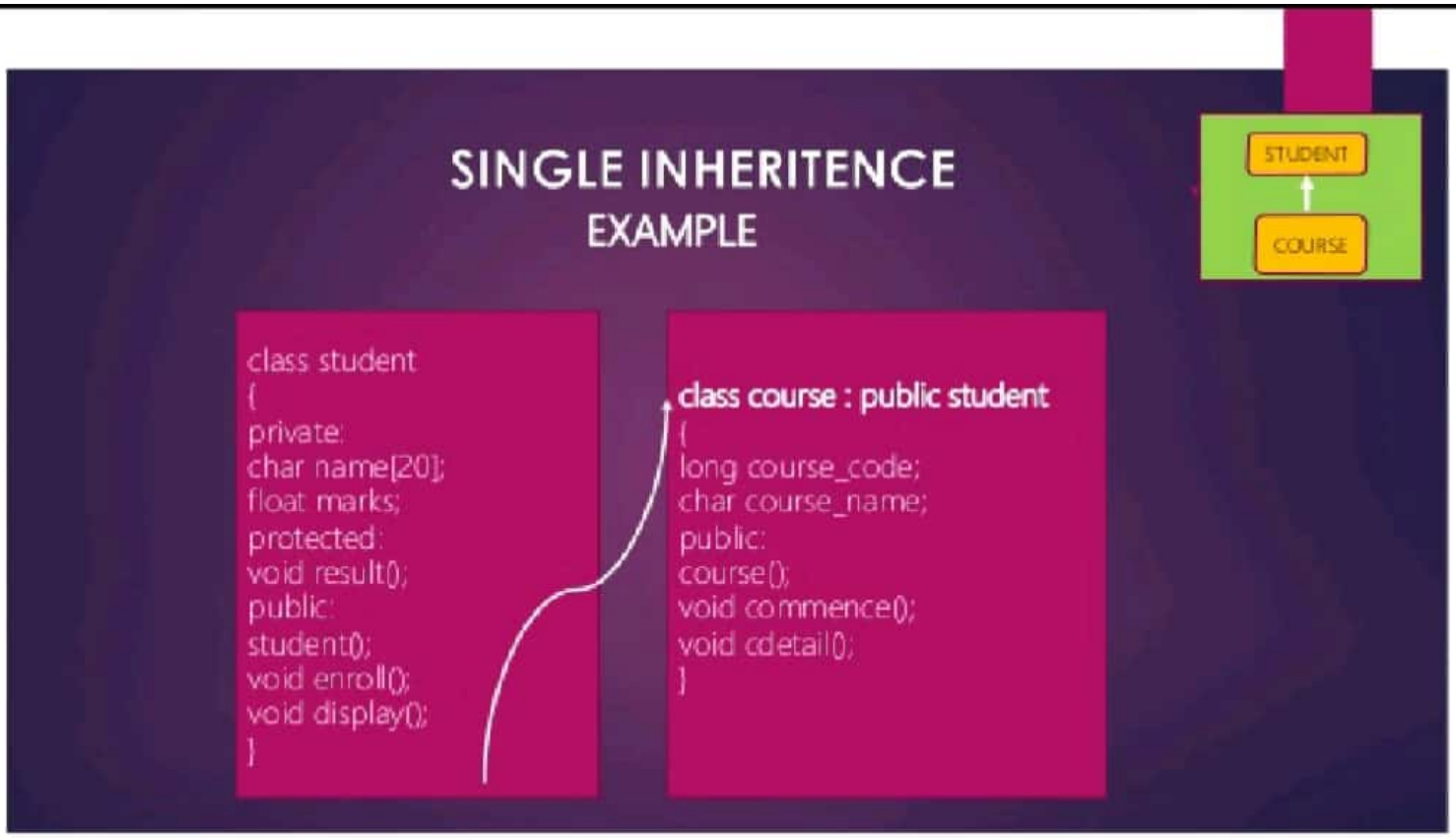
There are five different types of inheritance:

1. Single Inheritance
2. Multiple Inheritance
3. Multilevel Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

SINGLE INHERITENCE

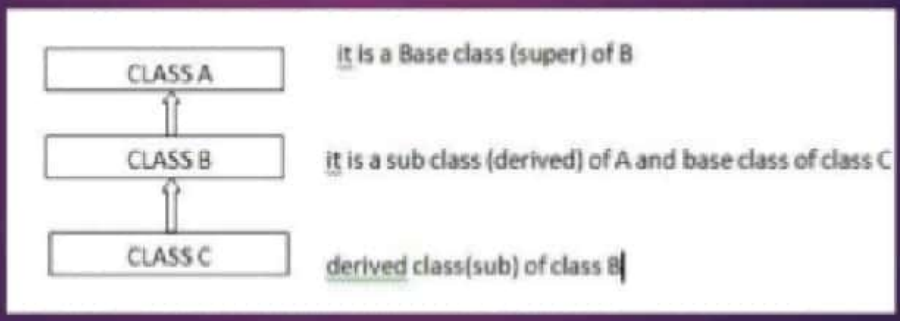
Single inheritance is the one where you have a single base class and a single derived class.

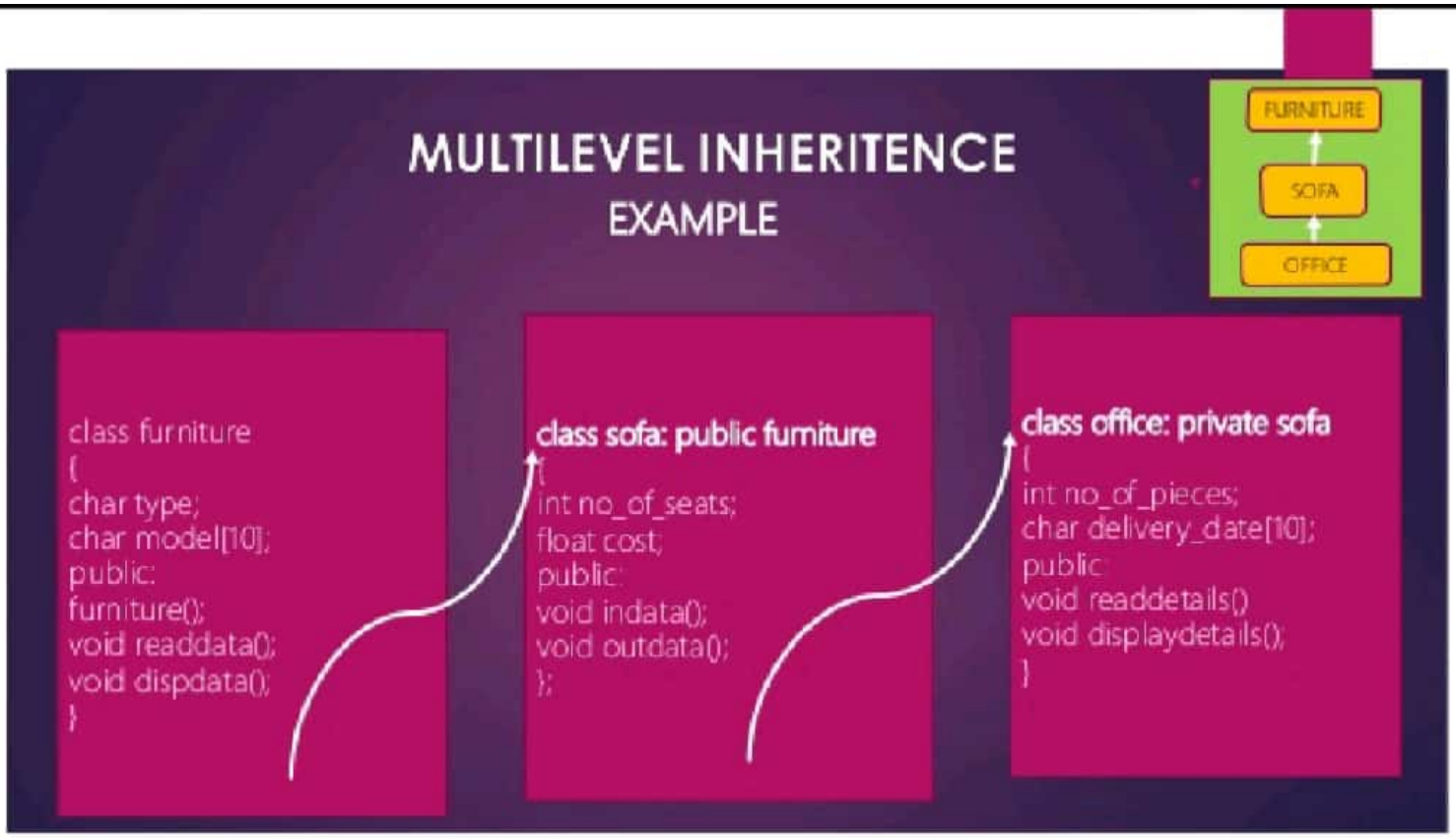




MULTILEVEL INHERITENCE

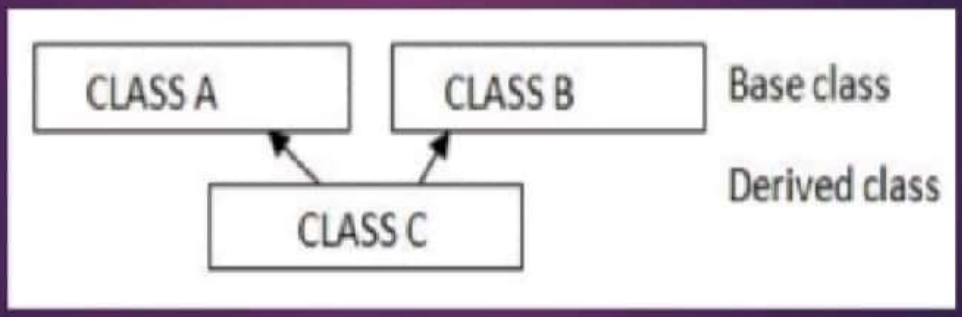
In Multi level inheritance, a subclass inherits from a class that itself inherits from another class.

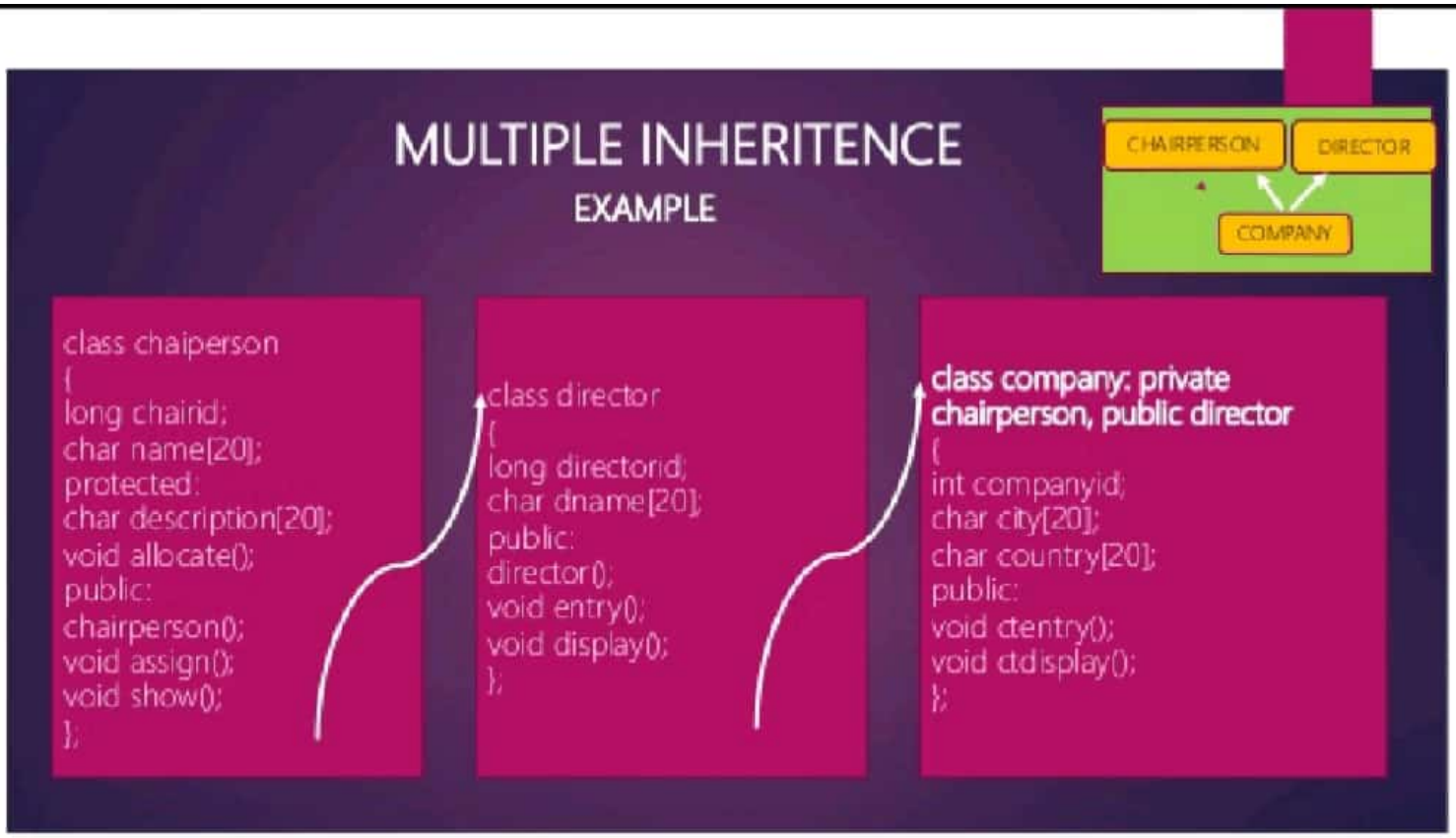




MULTIPLE INHERITENCE

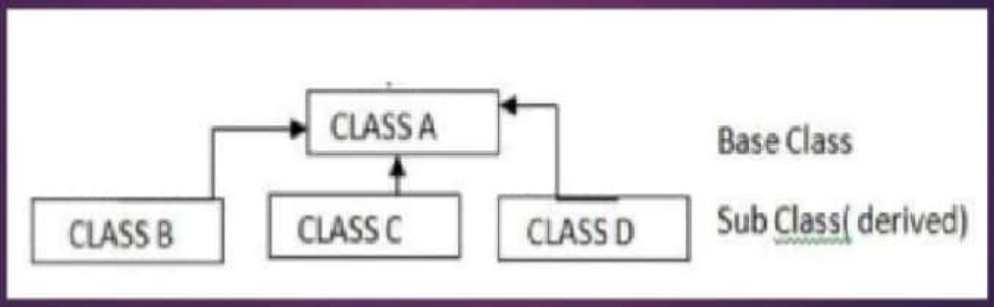
In Multiple inheritances, a derived class inherits from multiple base classes. It has properties of both the base classes.





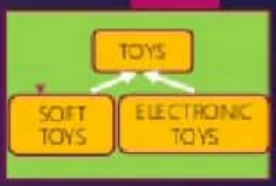
HIERARCHICAL INHERITENCE

In hierarchical Inheritance, it's like an inverted tree. So multiple classes inherit from a single base class.



HIERARCHICAL INHERITENCE

EXAMPLE



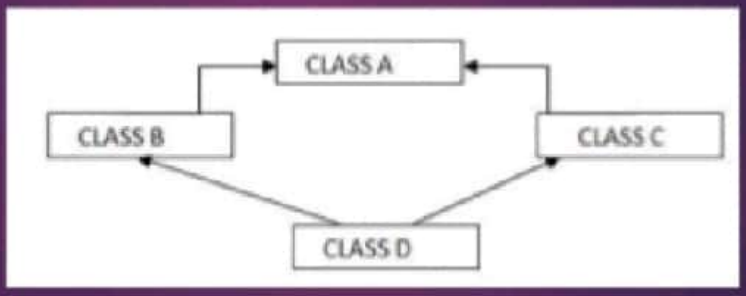

```
class toys
{
char tcode[5];
protected:
float price;
void assign(float);
public:
toys();
void tentry();
void tdisplay();
};
```

```
class softtoys: public toys
{
chat stname[20];
float weight;
public:
softtoys();
void stentry();
void stdisplay();
};
```

```
class electronictoys: public
toys
{
char etname[20];
int no_of_batteries;
public:
void etentry();
void etdisplay();
};
```

HYBRID INHERITENCE

It combines two or more types of inheritance. In this type of inheritance we can have a mixture of number of inheritances.



CONSTRUCTORS AND DESTRUCTORS IN BASE AND DERIVED CLASSES

- Derived classes can have their own constructors and destructors.
- When an object of a derived class is created, the base class's constructor is executed first, followed by the derived class's constructor.
- When an object of a derived class goes out of scope, its destructor is called first, then that of the base class.

IMPROTANT POINTS TO NOTE

- **Calculating the size of the object of the child class:**
  - While calculating the size of the object of the child class, add the size of all data members of base class including the private members of the base class and the child class.
  - If child class is inheriting from multiple base classes, add the size of data members of all base classes and the child class.
  - In case of multilevel inheritance the size of all base classes(directly /indirectly) inherited by child class is added to the size of child class data members

- **Members accessible to the object of the child class:**
  Only public members of the new modified child class(after inheritance) are accessible to the object of the child class.

- **Members accessible to the functions of the child class:**
  All members: public, protected, private, of the new modified child class(after inheritance) are accessible to the functions of the child class.

PASSING ARGUMENTS TO BASE CLASS CONSTRUCTOR

If a base class has parametrized constructor then it is the duty of child class to pass the parameters for base class constructor also at the time of creation of object.

```
class student
{
private:
char name[20];
float marks;
protected:
void result();
public:
student(char nam[20], float mar);
void enroll();
void display();
}
```

Base class constructor

```
class course : public student
{
long course_code;
char course_name[20];
public:
course(long cc, char cn[20],char nam[20], float mar ) :
        student(char nam[20], float mar);

void commence();
void cdetail();
}
course c1(01,"CS","Naman", 460);
```

Child class constructor

Base class constructor parameters