C++ Object Oriented Programming

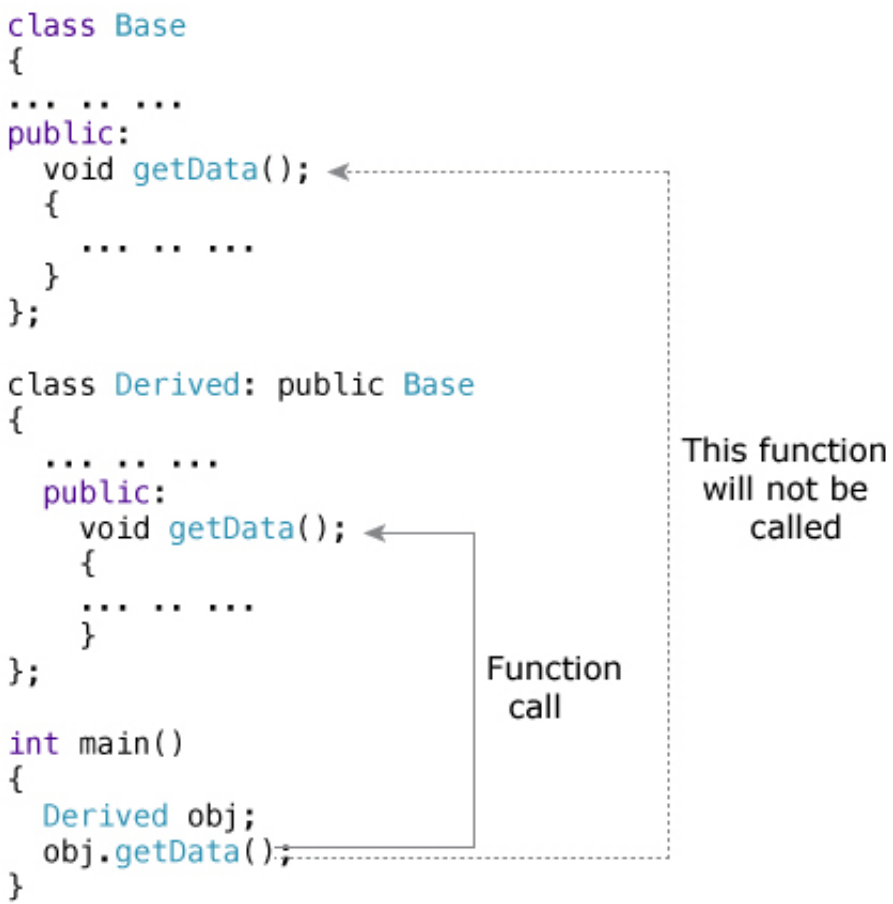
CS201-Data Structure

Function Overriding

- When a base class and a derived class contained a function with same name and parameter then the derived class function hides the base class function. This is called function overloading.
- It is the redefinition of base class function in its derived class with same signature i.e return type and parameters.

- It can only be done in derived class.

Function Overriding- Example

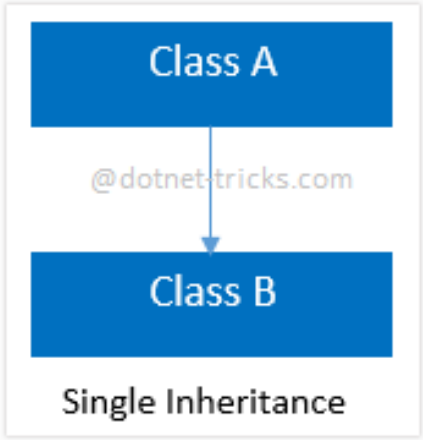


Function Overloading VS Function Overriding

1. **Inheritance:** Overriding of functions occurs when one class is inherited from another class. Overloading can occur without inheritance.
2. **Function Signature:** Overloaded functions must differ in function signature ie either number of parameters or type of parameters should differ. In overriding, function signatures must be same.
3. **Scope of functions:** Overridden functions are in different scopes; whereas overloaded functions are in same scope.
4. **Behavior of functions:** Overriding is needed when derived class function has to do some added or different job than the base class function. Overloading is used to have same name functions which behave differently depending upon parameters passed to them.

Types of Inheritance – (1) Single Inheritance

In this inheritance, a derived class is created from a single base class.

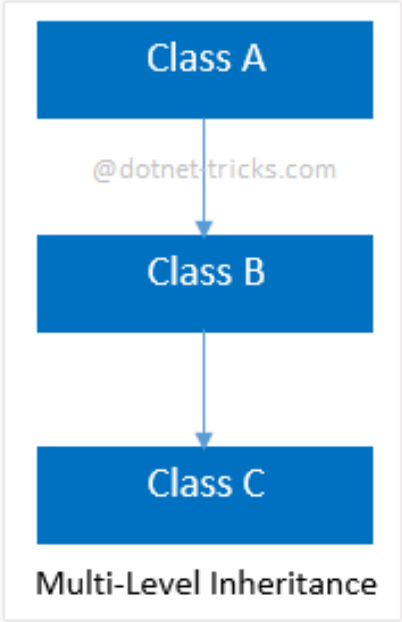


Single Inheritance

```
//Base Class
class A
{
 public void fooA()
 {
 //TO DO:
 }
}

//Derived Class
class B : A
{
 public void fooB()
 {
 //TO DO:
 }
}
```

(2) Multi-level inheritance

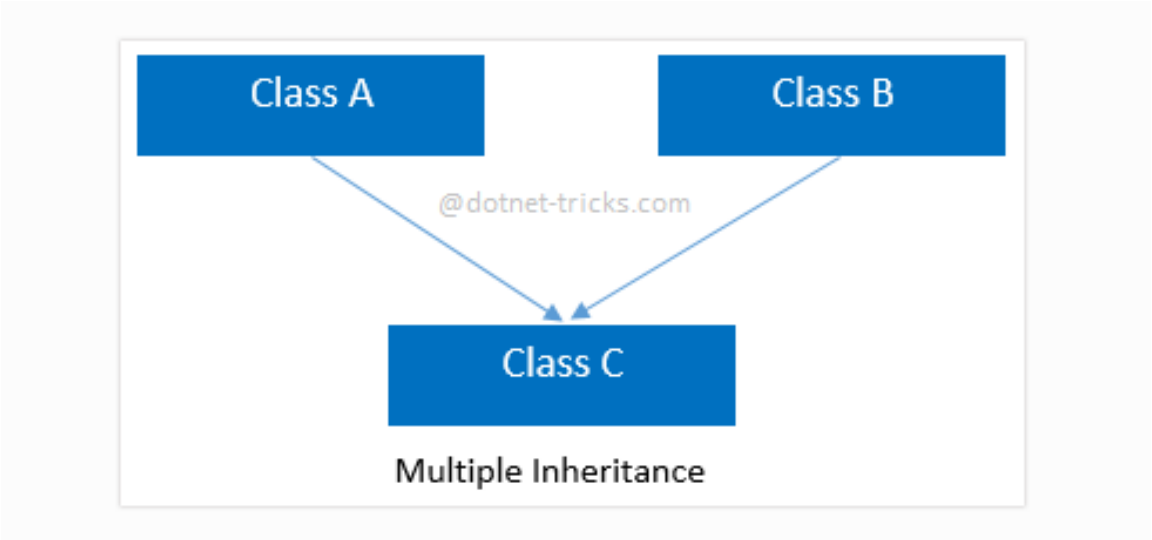
In this inheritance, a derived class is created from another derived class.



Multi-Level Inheritance

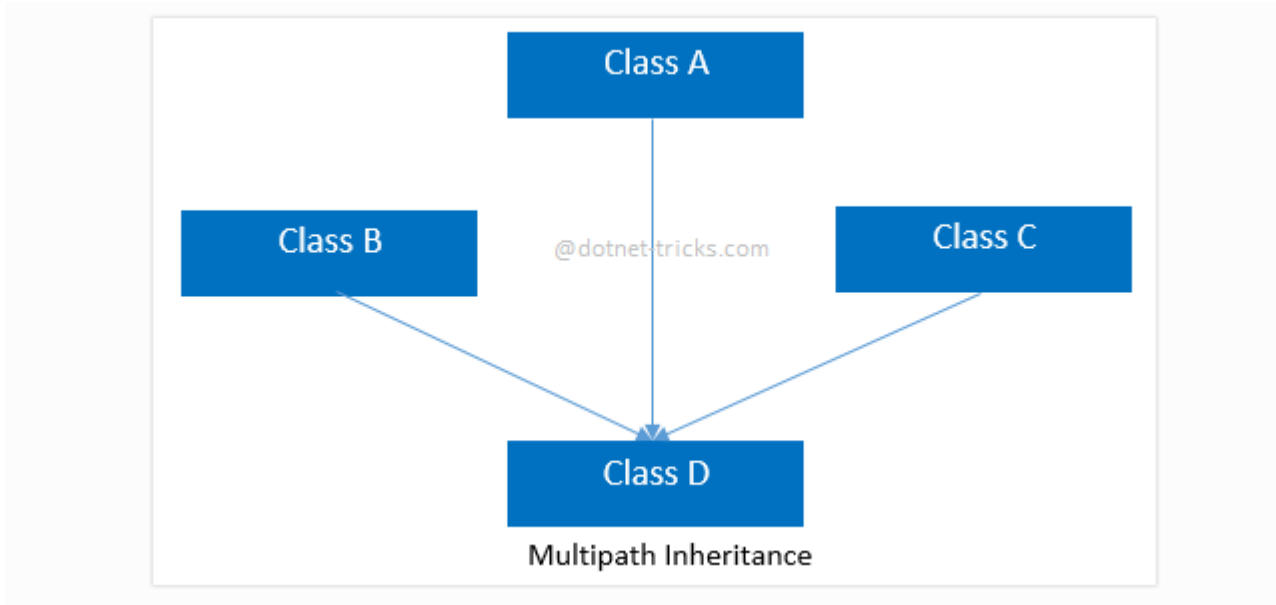
```
//Base Class
class A
{
 public void fooA()
 {
 //TO DO:
 }
}

//Derived Class
class B : A
{
 public void fooB()
 {
 //TO DO:
 }
}

//Derived Class
class C : B
{
 public void fooC()
 {
 //TO DO:
  }
}
```

(3) Multiple inheritance



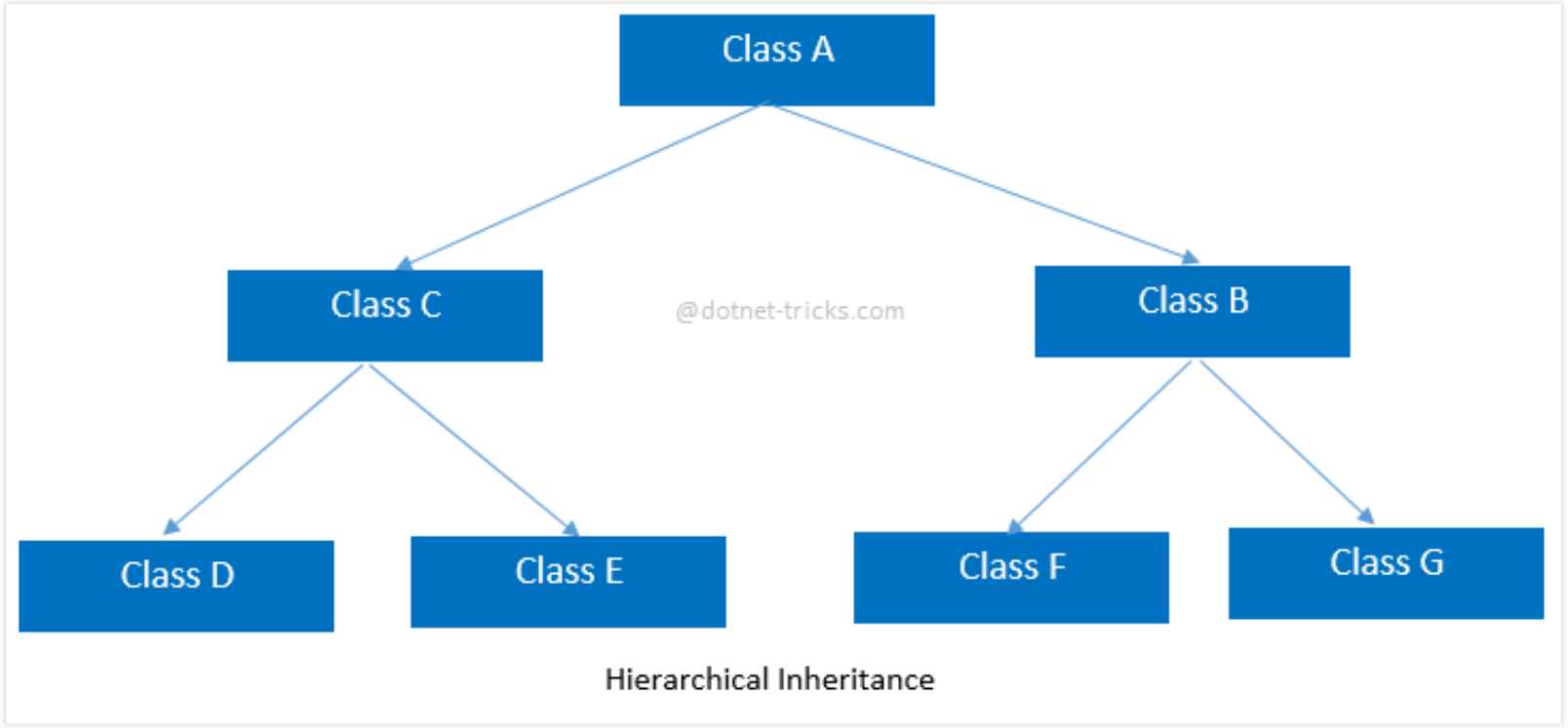

```
//Base Class
class A
{
 public void fooA()
 {
 //TO DO:
 }
}

//Base Class
class B
{
 public void fooB()
 {
 //TO DO:
 }
}

//Derived Class
class C : A, B
{
 public void fooC()
 {
  //TO DO:
  }
 }
```

(4) Multipath inheritance



(5) Hierarchical inheritance

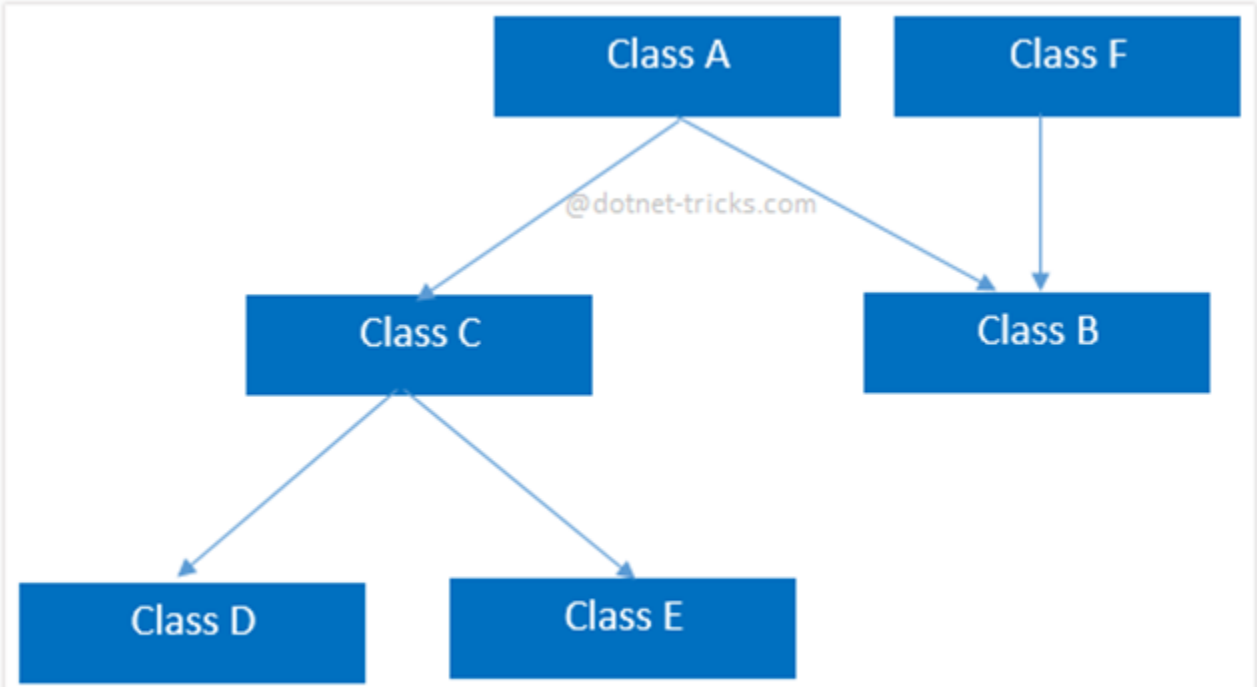
In this inheritance, more than one derived classes are created from a single base.



Hierarchical Inheritance

Hybrid inheritance

This is combination of more than one inheritance. Hence, it may be a combination of Multilevel and Multiple inheritance or Hierarchical and Multilevel inheritance or Hierarchical and Multipath inheritance or Hierarchical, Multilevel and Multiple inheritance.



Hybrid Inheritance – (a combination of Hierarchical and multiple)

Advantages of Inheritance

- Reduce code Redundancy
- Provides code reusability.
- Reduce source code size and improves code readability.
- Code is easy to manage and divided into parent and child classes.
- Support code extensibility by overriding the base class functionalities within child classes.

Disadvantages of Inheritance

- In inheritance base class and child classes are tightly coupled. Hence if you change the code of parent class, it will get affects to all the child classes.
- In class hierarchy, many data members remain unused and memory allocated to them is not utilized. Hence affect the performance of your program if you have not implemented the inheritance correctly.

Multiple inheritance:

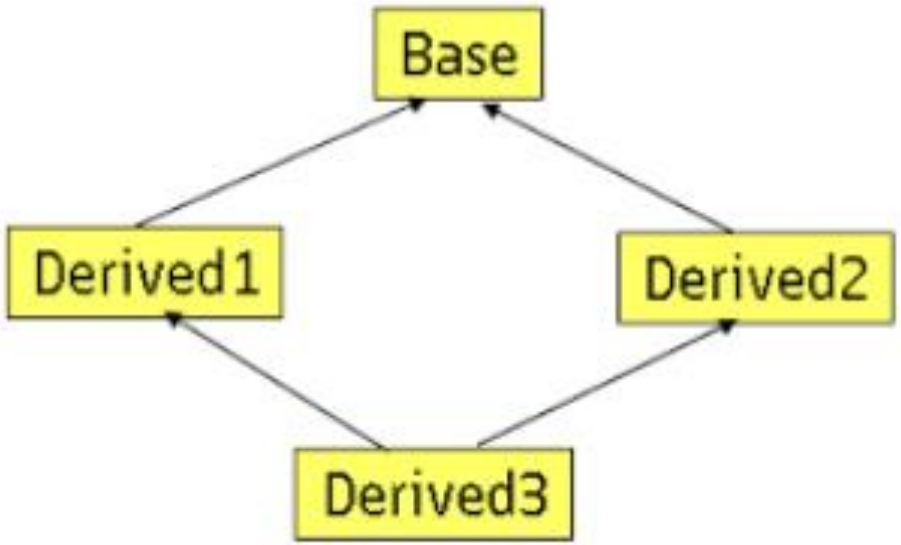
- Multiple inheritance is a feature of some object-oriented computer programming languages in which an object or class can inherit characteristics and features from more than one parent object or parent class.

- It is distinct from single inheritance, where an object or class may only inherit from one particular object or class.

Multiple inheritance issues-The diamond problem

- The "diamond problem" (sometimes referred to as the "deadly diamond of death") is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C. If there is a method in A that B and C have overridden, and D does not override it, then which version of the method does D inherit: that of B, or that of C?

The diamond problem



Virtual inheritance

- Virtual inheritance is a C++ technique that ensures only one copy of a base class's member variables are inherited by grandchild derived classes.

Virtual Functions

- A virtual function a member function which is declared within base class and is re-defined (Overriden) by derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class’s version of the function.

Virtual Functions

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve Runtime polymorphism
- Functions are declared with a virtual keyword in base class.

Virtual Functions

```cpp
// CPP program to illustrate
// concept of Virtual Functions
#include<iostream>
using namespace std;

class base
{
public:
    virtual void print ()
    { cout<< "print base class" <<endl; }

    void show ()
    { cout<< "show base class" <<endl; }
};

class derived:public base
{
public:
    void print ()
    { cout<< "print derived class" <<endl; }

    void show ()
    { cout<< "show derived class" <<endl; }
};

int main()
{
    base *bptr;
    derived d;
    bptr = &d;

    //virtual function, binded at runtime
    bptr->print();

    // Non-virtual function, binded at compile time
    bptr->show();
}
```

```
print derived class
show base class
```

Abstract class and Pure Virtual Function

- A pure virtual function (or abstract function) in C++ is a virtual function for which we don’t have implementation, we only declare it. A pure virtual function is declared by assigning 0 in declaration.

```
// An abstract class
class Test
{
    // Data members of class
public:
    // Pure Virtual Function
    virtual void show() = 0;

   /* Other members */
};
```

Abstract class and Pure Virtual Function

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

class Base
{
   int x;
public:
    virtual void fun() = 0;
    int getX() { return x; }
};

// This class ingerits 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
```

Abstract class and Pure Virtual Function-Facts

- 1) A class is abstract if it has at least one pure virtual function.
- 2) We can have pointers and references of abstract class type.
- 3) If we do not override the pure virtual function in derived class, then derived class also becomes abstract class.