Object Oriented Programming

Bilal Khalid Dar

Inheritance

What is Inheritance

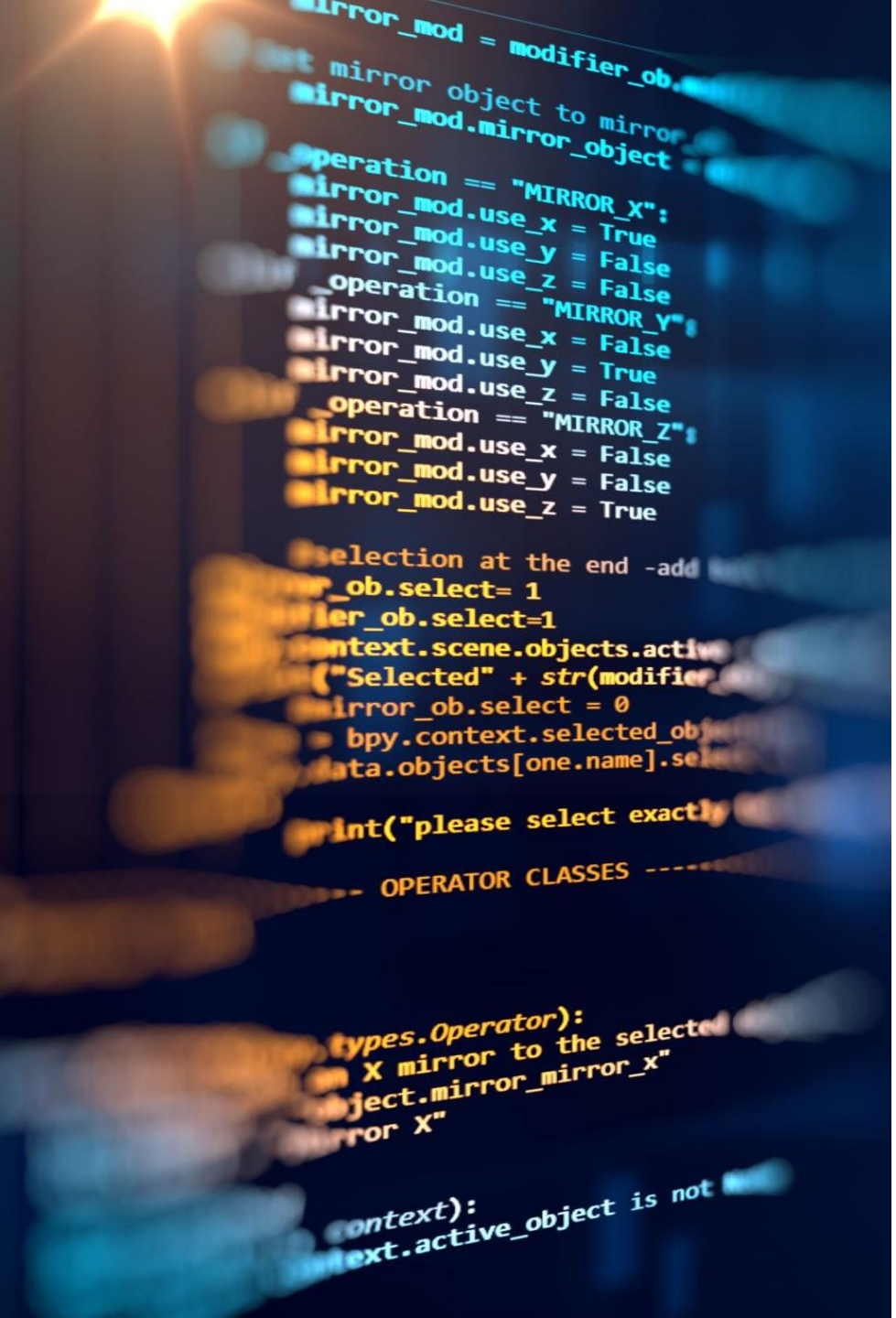
Inheritance is a mechanism in which one class acquires the property of another class.

For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class.

Hence, inheritance facilitates Reusability and is an important concept of OOPs.

Inheritance

- *Inheritance* allows a software developer to derive a new class from an existing one
- The existing class is called the *parent class,* or *superclass*, or *base class*
- The derived class is called the *child class* or *subclass*
- As the name implies, the child inherits characteristics of the parent
- That is, the child class inherits the methods and data defined by the parent class



Inheritance In C++

- Inheritance is a mechanism in which one class acquires the property of another class. In Java, when an “Is-A” relationship exists between two classes, we use Inheritance.
- Inheritance is important since it leads to the reusability of code.

is-a relationship

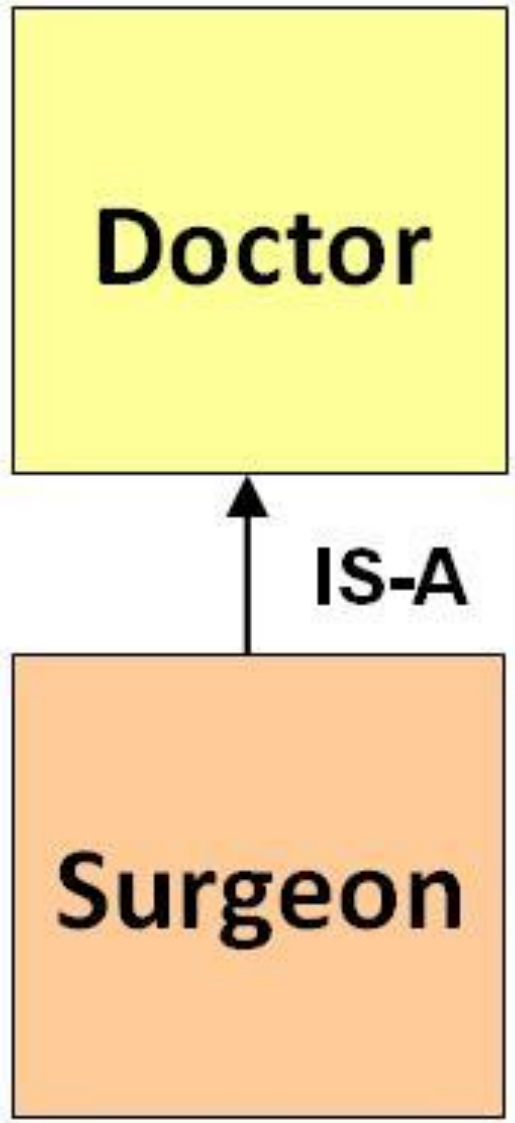
Car is a Vehicle

Orange is a Fruit

Surgeon is a Doctor

Dog is an Animal

Example



Important Terminologies

Super Class: The class whose features are inherited is known as superclass(or a base class or a parent class).

Sub Class: The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

How to do Inheritance in C++

```
class  <derived_class_name> : <access-specifier> <base_class_name>
{
        //body
}
```

How to do Inheritance in C++

```
1. class ABC : private XYZ
//private derivation
            {                  }
2. class ABC : public XYZ
//public derivation
            {                  }
3. class ABC : protected XYZ
//protected derivation
            {                  }
4. class ABC: XYZ
//private derivation by default
{            }
```

Example

```
class Animal {
}

class Mammal : public Animal {
}

class Reptile : protected Animal {
}

class Dog : private Mammal {
}
```

Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

Example

```
class Animal {
}

class Mammal : public Animal {
}

class Reptile : protected Animal {
}

class Dog : private Mammal {
}
```

Now, if we consider the IS-A relationship, we can say –

- Mammal IS-A Animal
- Reptile IS-A Animal
- Dog IS-A Mammal
- Hence: Dog IS-A Animal as well

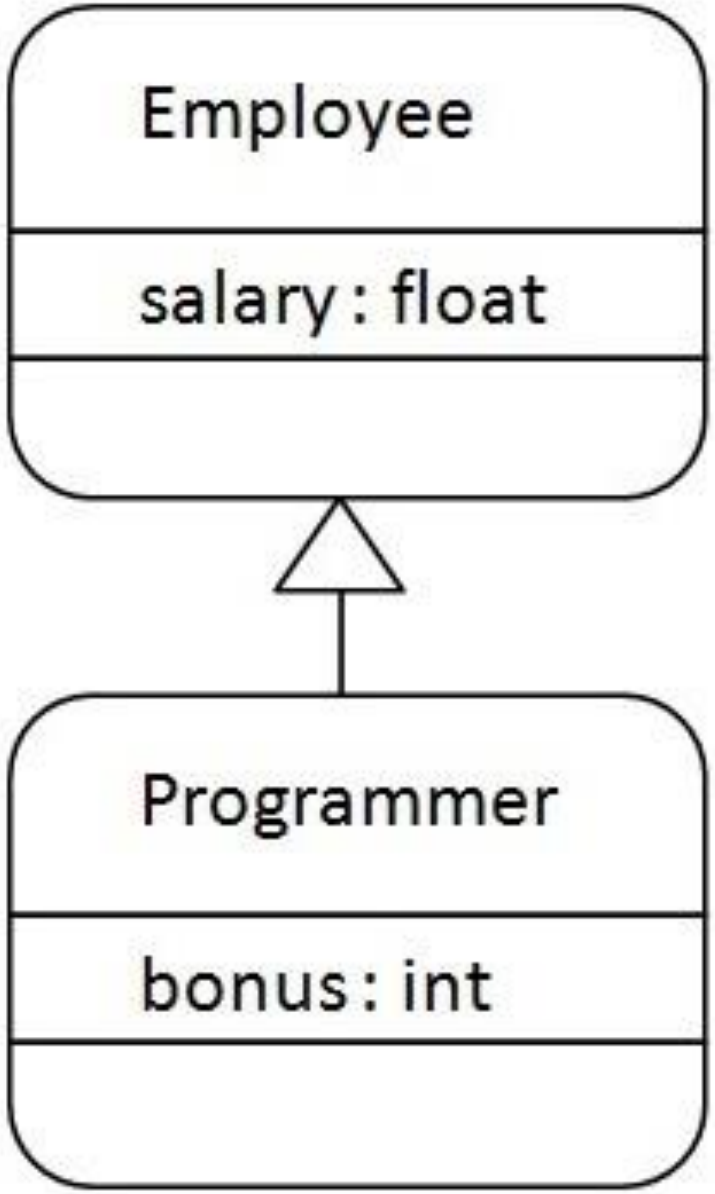
Example

```
class Animal {
}

class Mammal : public Animal {
}

class Reptile : protected Animal {
}

class Dog : private Mammal {
}
```

- Now, based on the above example, in Object-Oriented terms, the following are true −

- Animal is the superclass of Mammal class.
- Animal is the superclass of Reptile class.
- Mammal and Reptile are subclasses of Animal class.
- Dog is the subclass of both Mammal and Animal classes.

Example: Employee



Example: Accessing Super Class attributes in child class

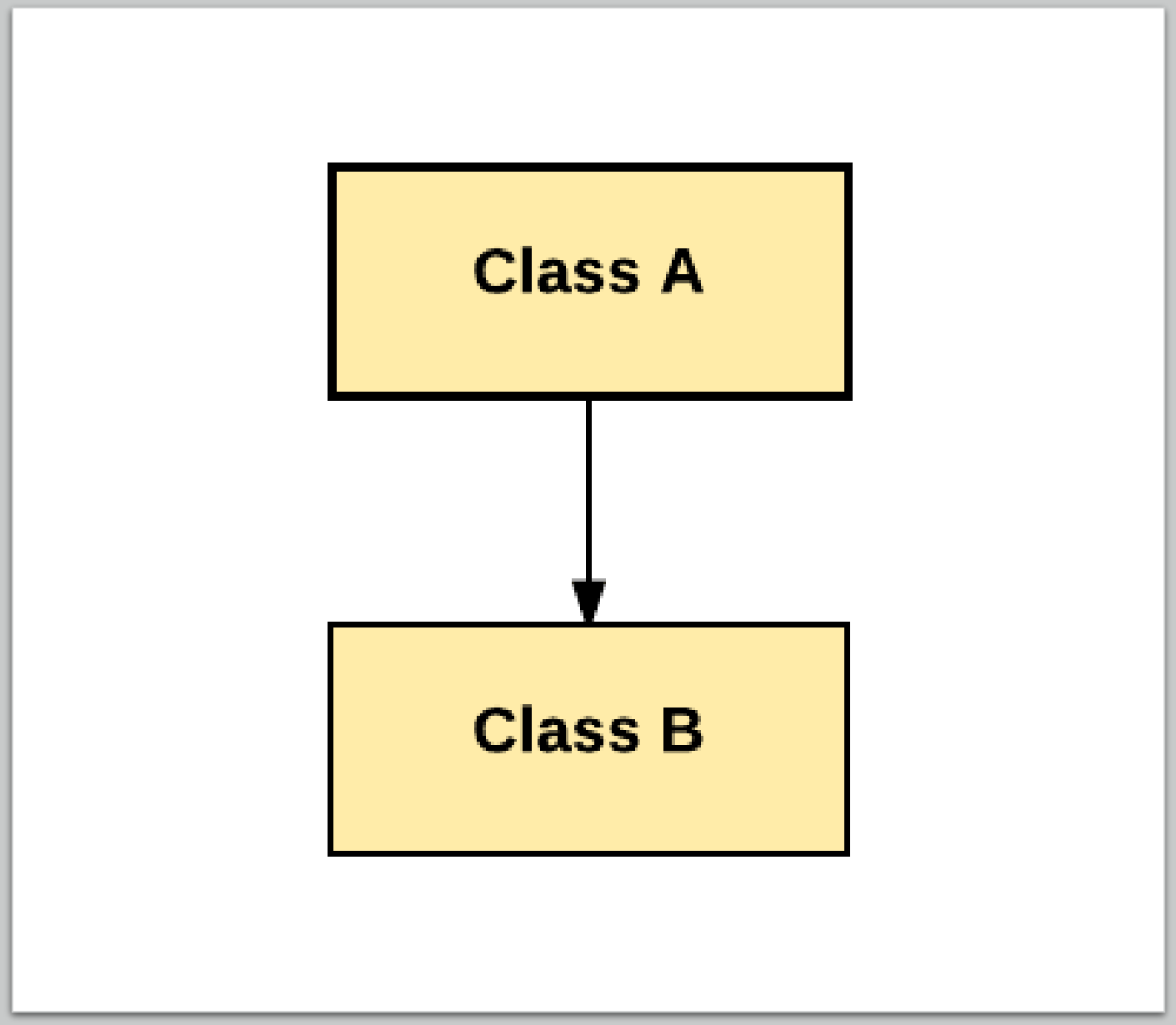
```
class A{
    public:
        int aa;
        A()
        {
            cout << "A constructor";
        }
        A(int a)
        {
            cout << "A parameter";
            aa= 10;
        }
};
class B: public A{
    public:
        B():A (5)
        {
            cout << "B constructor";
        }
};
main()
{
     B obj;
     obj.aa = 10;
     cout << obj.aa;
}
```

Types of Inheritance

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

Single Inheritance

- In Single Inheritance one class extends another class (one class only).



Example

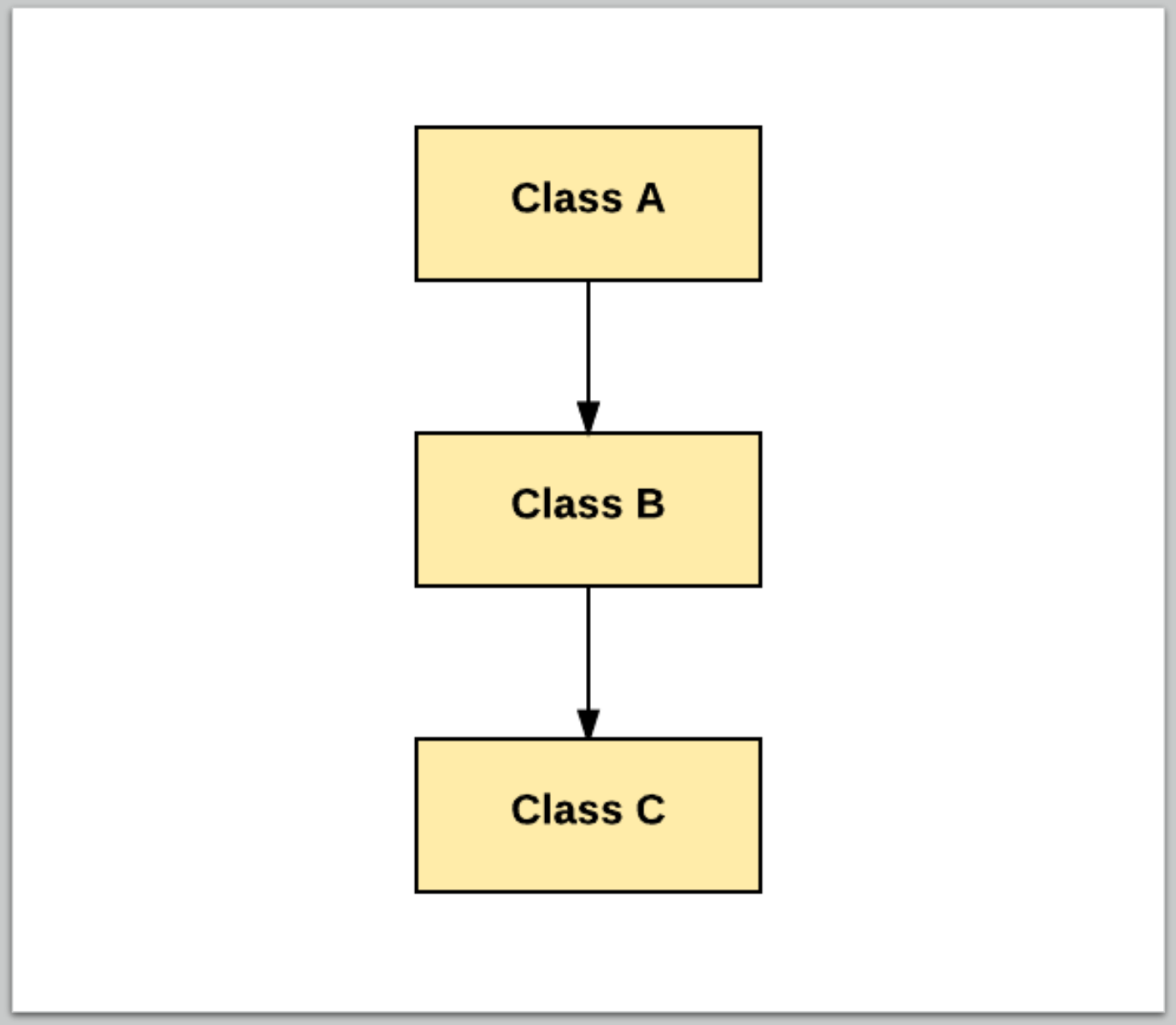
```cpp
// C++ program to explain
// Single inheritance
#include<iostream>
using namespace std;

// base class
class Vehicle {
  public:
    Vehicle()
    {
      cout << "This is a Vehicle\n";
    }
};

// sub class derived from a single base classes
class Car : public Vehicle {

};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base classes
    Car obj;
    return 0;
}
```

Output

```
This is a Vehicle
```

Multilevel Inheritance

- In Multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.



Example

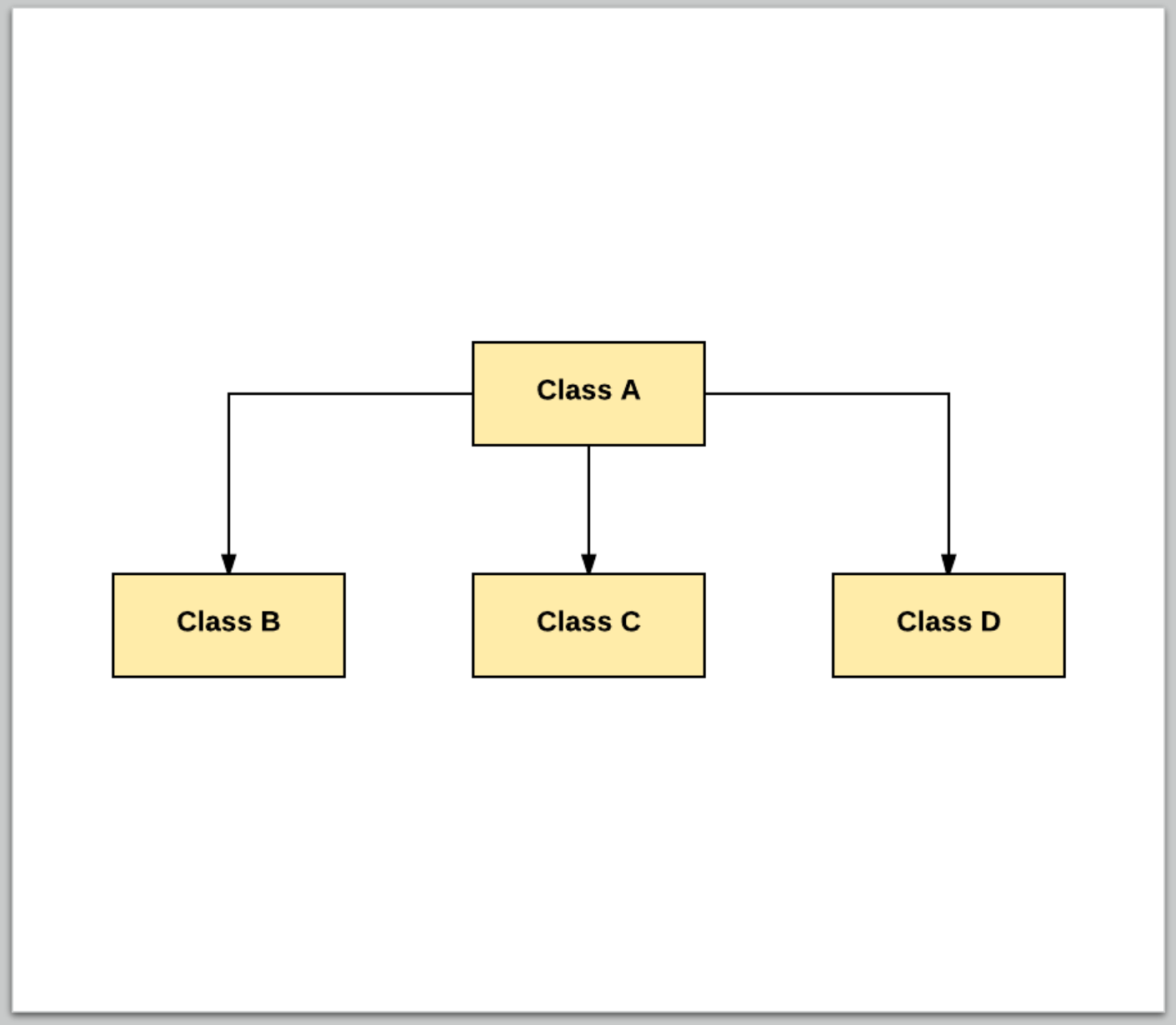
```cpp
// C++ program to implement
// Multilevel Inheritance
#include <iostream>
using namespace std;

// base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// first sub_class derived from class vehicle
class fourWheeler : public Vehicle {
public:
    fourWheeler()
    {
        cout << "Objects with 4 wheels are vehicles\n";
    }
};
// sub class derived from the derived base class fourWheeler
class Car : public fourWheeler {
public:
    Car() { cout << "Car has 4 Wheels\n"; }
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base classes.
    Car obj;
    return 0;
}
```

Output

```
This is a Vehicle
Objects with 4 wheels are vehicles
Car has 4 Wheels
```

Hierarchical Inheritance:

- In Hierarchical Inheritance, one class is inherited by many sub classes.



Example

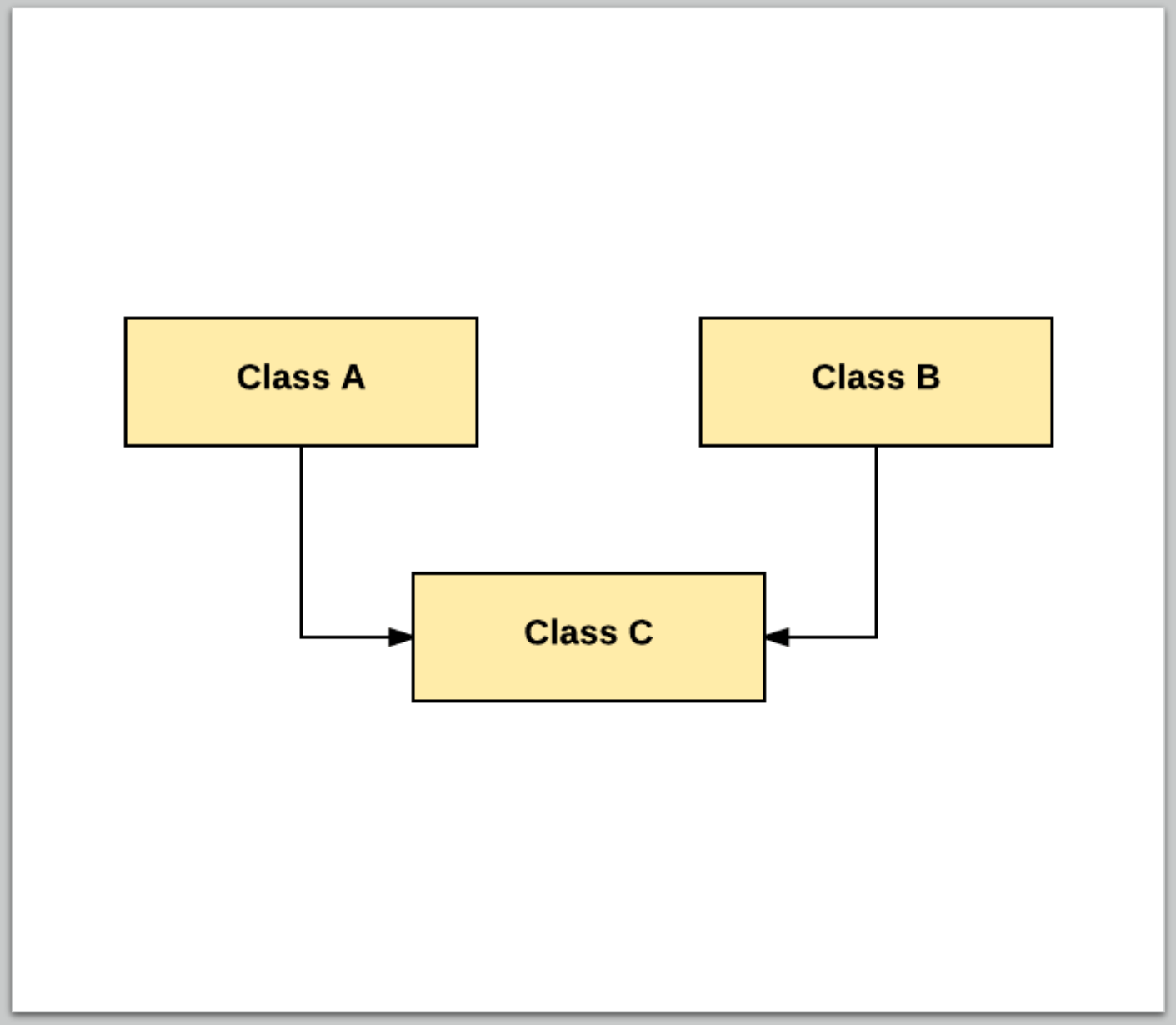
```cpp
// C++ program to implement
// Hierarchical Inheritance
#include <iostream>
using namespace std;

// base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// first sub class
class Car : public Vehicle {
};

// second sub class
class Bus : public Vehicle {
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base class.
    Car obj1;
    Bus obj2;
    return 0;
}
```

Output

```
This is a Vehicle
This is a Vehicle
```

Multiple Inheritance

- Multiple Inheritance is one of the inheritance in C ++ types where one class extending more than one class.



Code Example

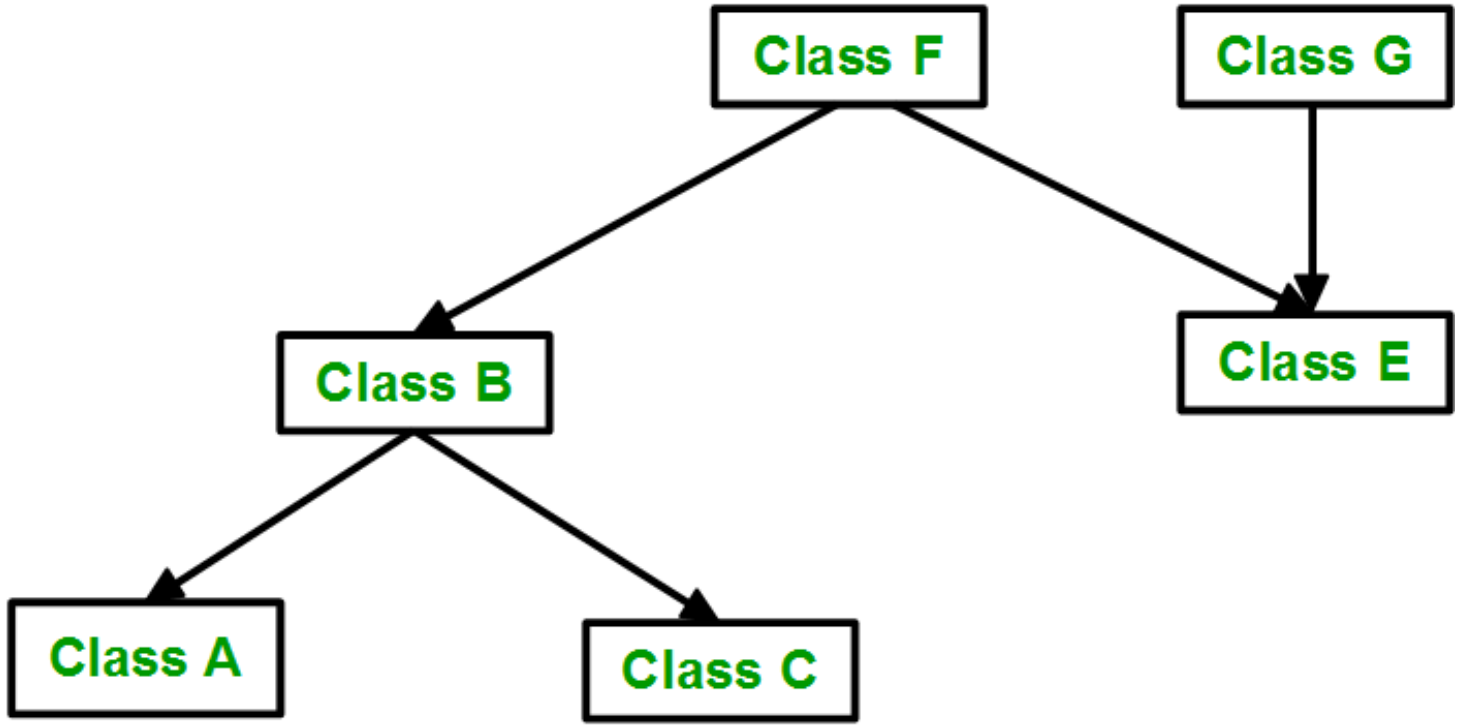
```cpp
// C++ program to explain
// multiple inheritance
#include <iostream>
using namespace std;

// first base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// second base class
class FourWheeler {
public:
    FourWheeler()
    {
        cout << "This is a 4 wheeler Vehicle\n";
    }
};

// sub class derived from two base classes
class Car : public Vehicle, public FourWheeler {
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base classes.
    Car obj;
    return 0;
}
```

Output

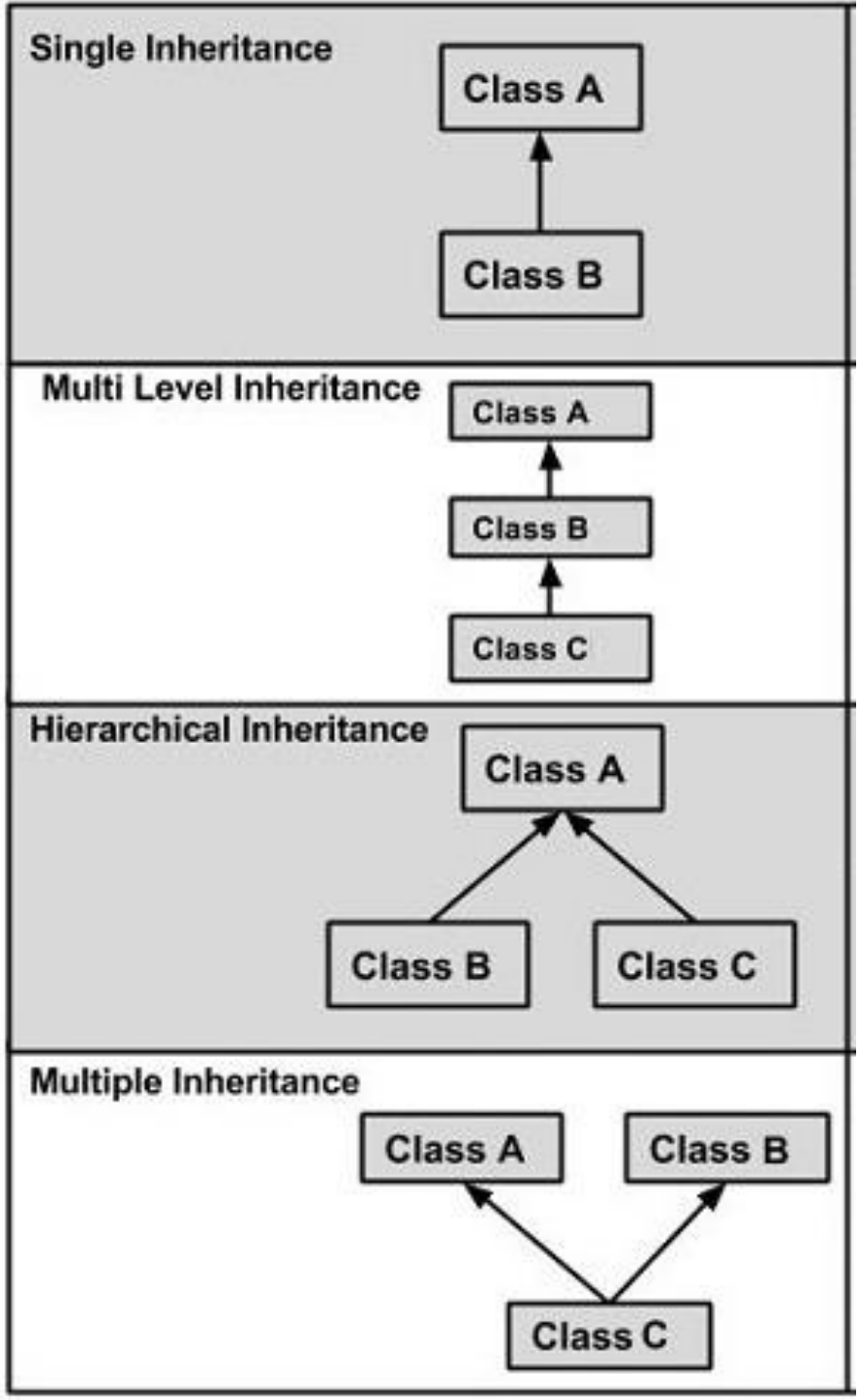
```
This is a Vehicle
This is a 4 wheeler Vehicle
```

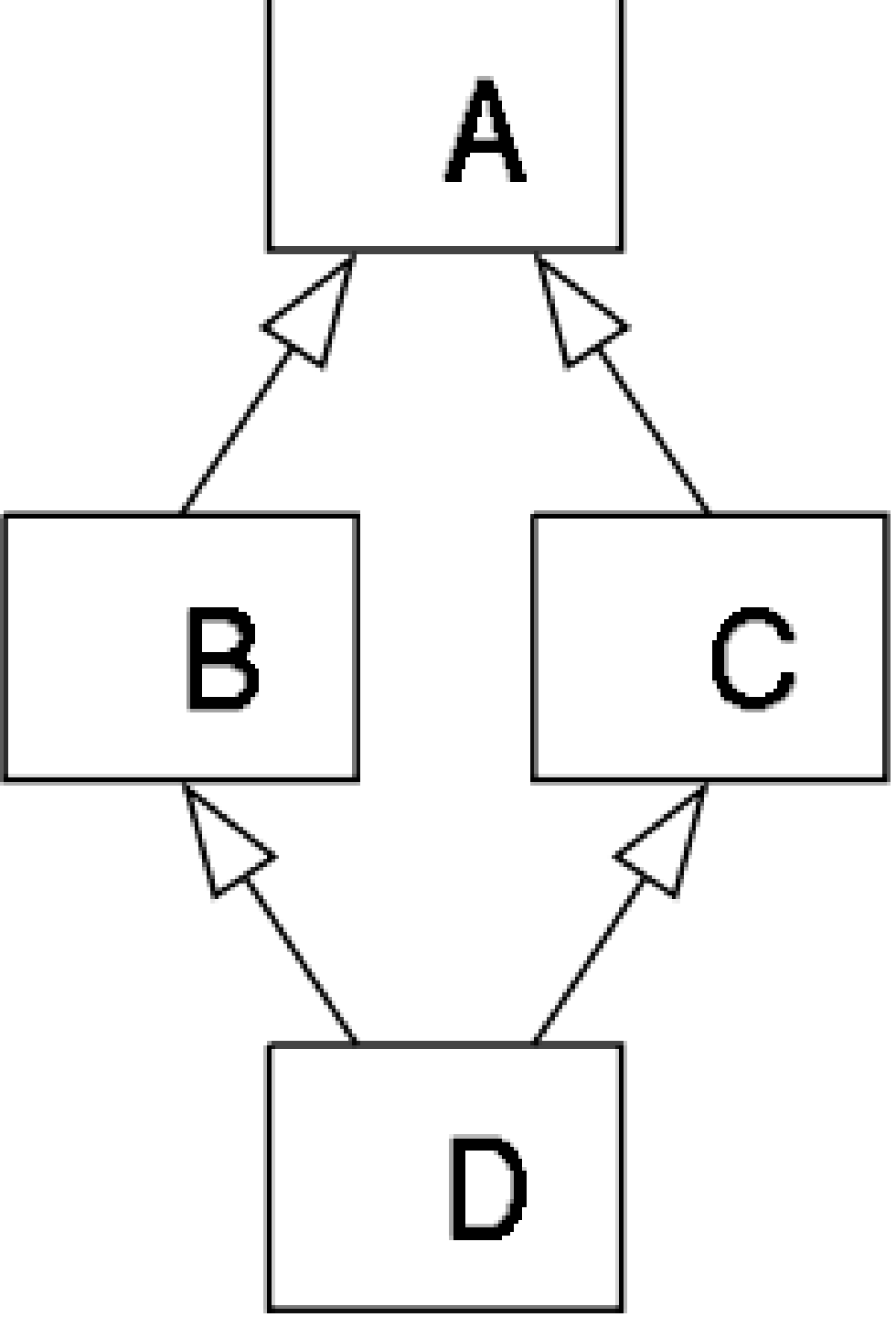
Hybrid(Virtual) Inheritance:

- Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance



Summary


Single Inheritance
Class A
Class B
Multi Level Inheritance
Class A
Class B
Class C
Hierarchical Inheritance
Class A
Class B
Class C
Multiple Inheritance
Class A
Class B
Class C



Why Multiple Inheritance is Dangerous?

We will answer it soon!!

Method Overriding

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Method Overriding

Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.

Method overriding is used for runtime polymorphism

Rules for C++ Method Overriding

The method must have the same name as in the parent class

The method must have the same parameter as in the parent class.

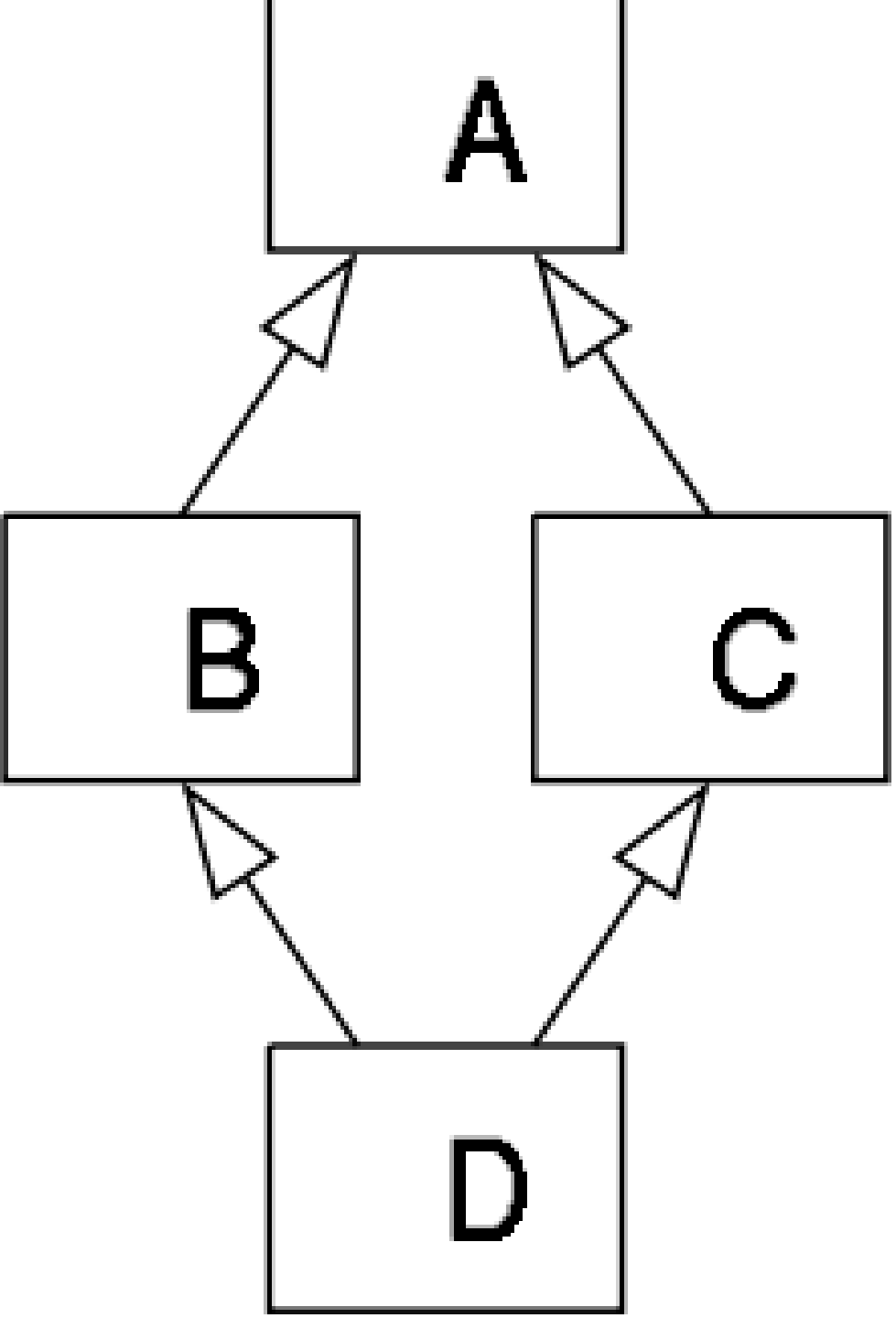
There must be an IS-A relationship (inheritance).

Example – Overriding

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

class Parent {
public:
    void Print()
    {
        cout << "Base Function" << endl;
    }
};

class Child : public Parent {
public:
    void Print()
    {
        cout << "Derived Function" << endl;
    }
};

int main()
{
    Child Child_Derived;
    Child_Derived.Print();
    return 0;
}
```

Output

```
Derived Function
```



Why Multiple Inheritance is Dangerous

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?


Exercise

Shapes

- Shape:
  - color, fields
  - draw() draw itself on the screen
  - calcArea() calculates its own area.

Kinds of Shapes

- Rectangle
- Triangle
- Circle

Each could be a kind of shape (could be specializations of the shape class).

Each knows how to draw itself, etc.