CodeChef SCOE Chapter

OOP Cheat Sheet

Summary:

Object oriented programming is a way of solving complex problems by breaking them into smaller problems using objects. Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.

Few principles of Object Oriented Programming are -

- Classes
- Objects
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Classes and Objects:

Classes -

1. A class is the building block that leads to Object-Oriented programming.
2. It is like a blueprint for an object.
3. It is a user-defined data type which has data members and member functions
4. For Example:

Consider the Class of Cars.There may be many cars with different brands but all of them will share some common properties. In this case, speed limits, mileage are the data members and increasing speed, applying brakes are member functions.

Syntax:

```
class ClassName
{
        Access specifier:       //can be private,public or protected

        Data members;           // Variables to be used

        Member Functions() {}   //Methods to access data members

};                      // Class name ends with a semicolon
```

Objects -

1. An Object is an instance of a Class.
2. When an object is created (i.e. when the class is instantiated) memory is allocated.

Syntax:

```
ClassName ObjectName;
```

Data Abstraction:

1. Data Abstraction refers to providing only essential information to the outside world, hiding the background details.

2. There are two ways to achieve Data abstraction .

a) **Abstraction using classes** :- A class is used to group data members and member functions using access specifiers. Class can decide which data member is visible to the outside world.

Example :

```
                #include<iostream.h>

                using namespace std;
```

```
      class Sum

      {

   // private variables which are only accessible

          //  to member function of class

        private:

              int  x,y,z;

        public:

              void addTwoNumbers()

              {

                     cout<<"Enter two numbers";

                     cin>>x>>y;

                     z=x+y;

                     cout<<"Sum of two numbers is "<<z<<"\n";

              }

      };

      int main()

      {

              // object

              Sum sm;

              sm.addTwoNumbers();

              return 0;

      }
```

b) **Abstraction using header files** :- Abstraction can also be achieved through header files. Example: Consider sqrt() function in C++ which is present in the math.h header file. Whenever we are required to find the square root of a number we call the sqrt() function without knowing the actual implementation in the math.h header file.

Example :

```
 #include <iostream>

#include<math.h>

using namespace std;

int main()

{

   int Num=16;

   // sqrt is inbuilt function in math.h header file

   // this leads to abstraction

   int SquareRootOfNum=sqrt(Num);

   cout<<SquareRootOfNum<<"\n";

   return 0;

}
```

3. **Real Life example of Abstraction** :

Consider the man driving a car. Whenever he feels to increase the speed , he will press the accelerator. Whenever he feels to decrease the speed , he will press the brake. But he does not know the inner mechanism of the brake and accelerator. This is called abstraction.

Data Encapsulation:

1. Data encapsulation is a concept that binds together data and member functions that manipulate data. Encapsulation also keeps data secure from outside interference.

2. Data encapsulation leads to data abstraction or data hiding. This makes encapsulation an important concept in object oriented programming.

Example :

```
#include<iostream>

using namespace std;

class Encapsulation

{

        Private:

        // data hidden from outside world

                        int x;

        public:

                        void set(int a)// function to set value of variable x

                        {

                                x=a;

                        }

                        int get()// function to return value of variable x

                        {

                                return x;

                        }

};
```

In the above program the variable x is made private. This variable can be accessed and manipulated only using the functions get() and set() which are present inside the class

Abstract Class :

1. Abstract class is a class that is used as base class. Abstract class contains at least one pure virtual function.
2. If we do not override pure virtual function in derived class, then derived class also becomes abstract class.
3. Example :

```
class  AB
{
      public:
          // pure virtual function.
        virtual void f() =0;


};
```

■ ■ ■

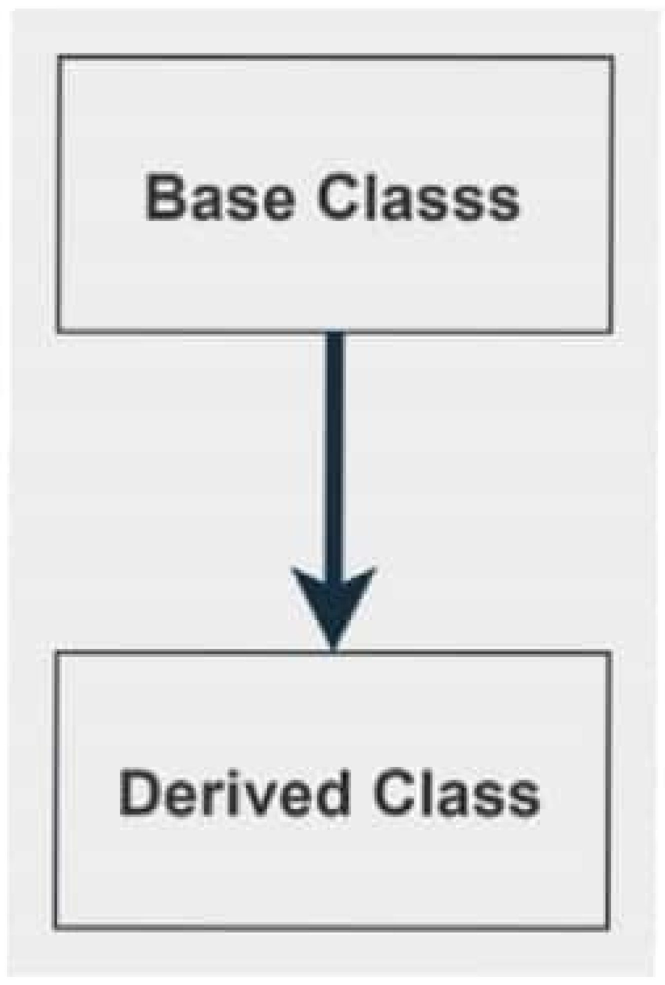
Inheritance:

Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. The class which inherits the members of another class is called derived class and the class whose members are inherited is called base class

Types of inheritance in C++:

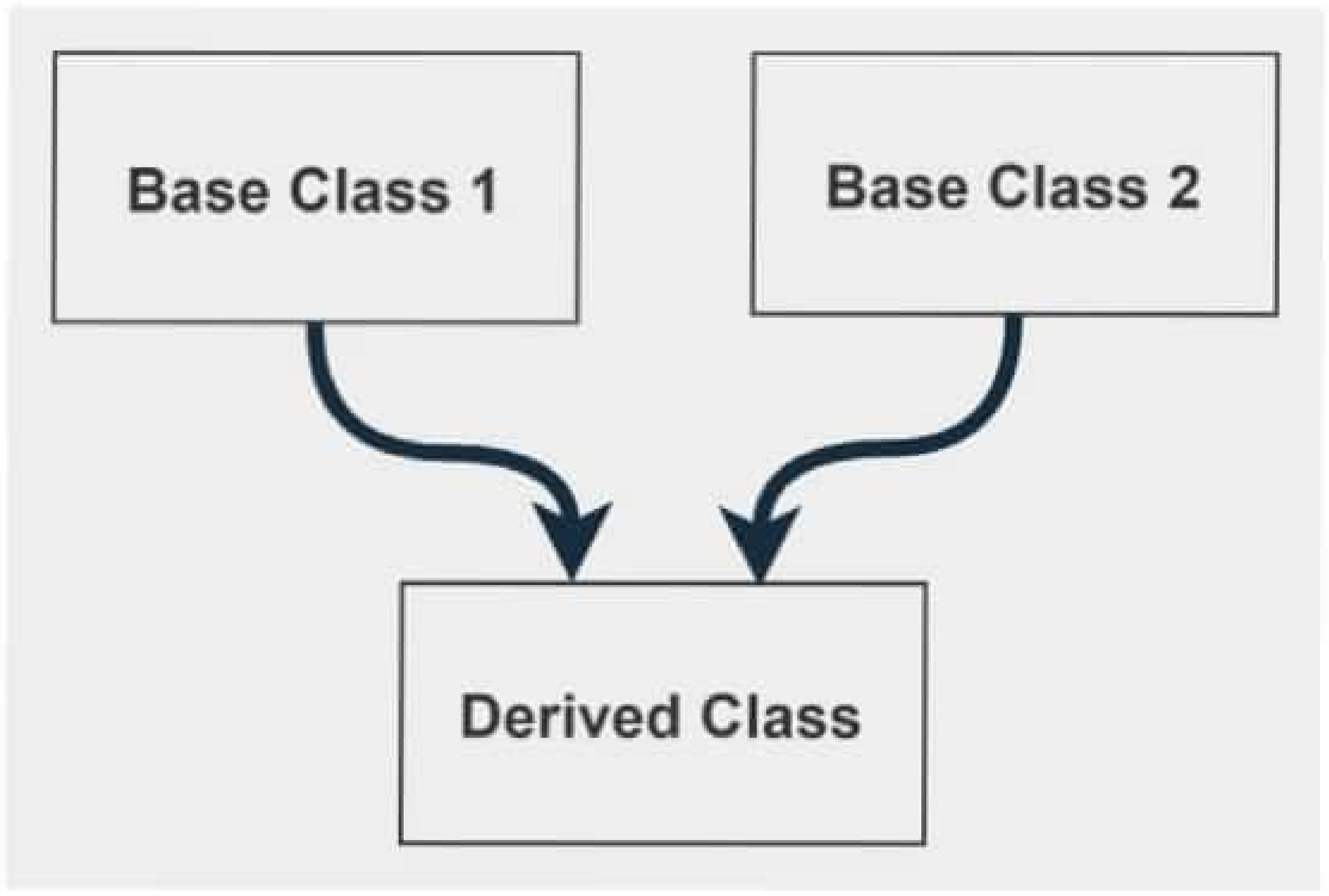
1. Single Inheritance
2. Multiple Inheritance
3. Multilevel Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

Single Inheritance - One derived class can inherit property from only one base class.



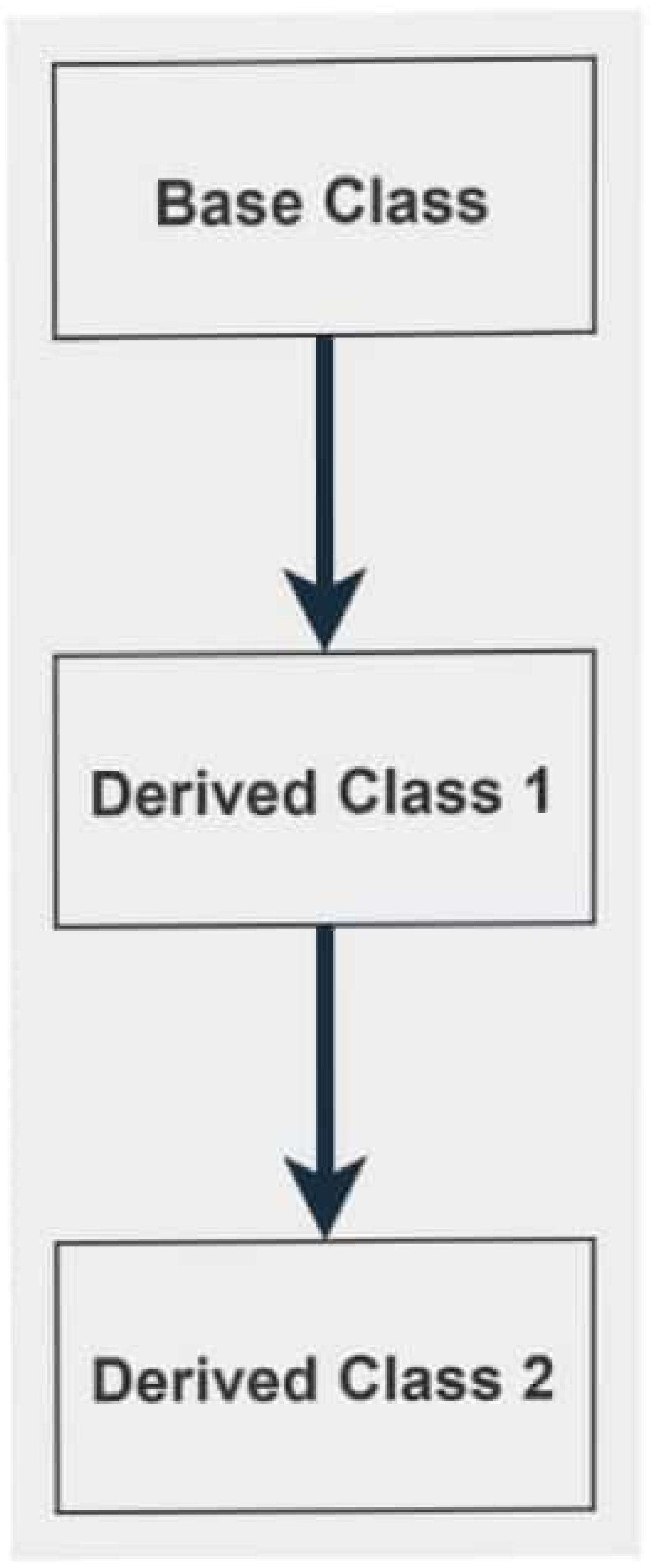
```
class DerivedClass: access_mode BaseClass
{
        //body of Derived class
};
```

Multiple Inheritance - A single derived class can inherit property from more than one base class.



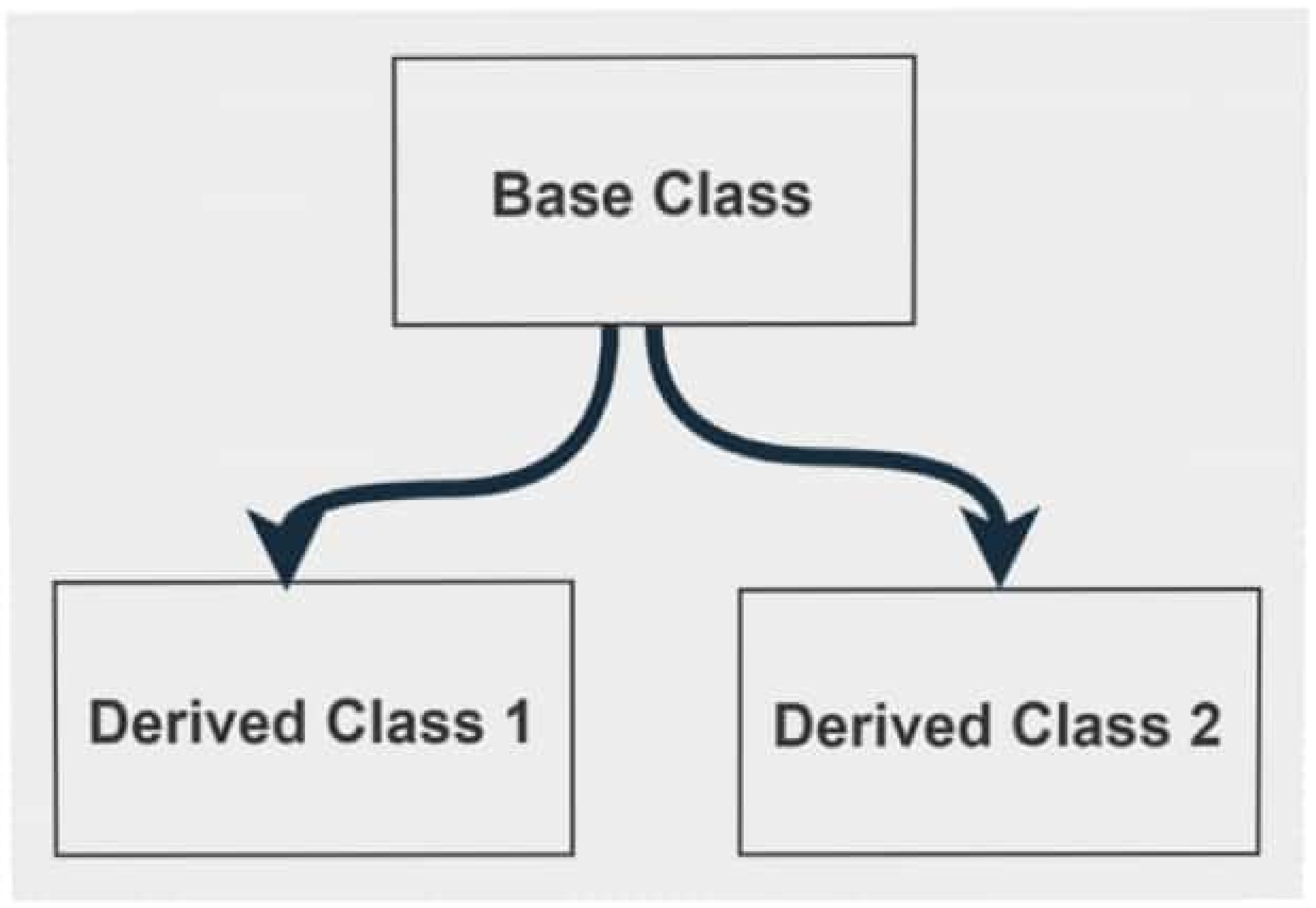
```
class Derived: access_mode Base1, access_mode Base2
{
        //body of Derived class
        //inherit property from Base1 & Base2
};
```

Multilevel Inheritance - The derived class inherits property from another derived class



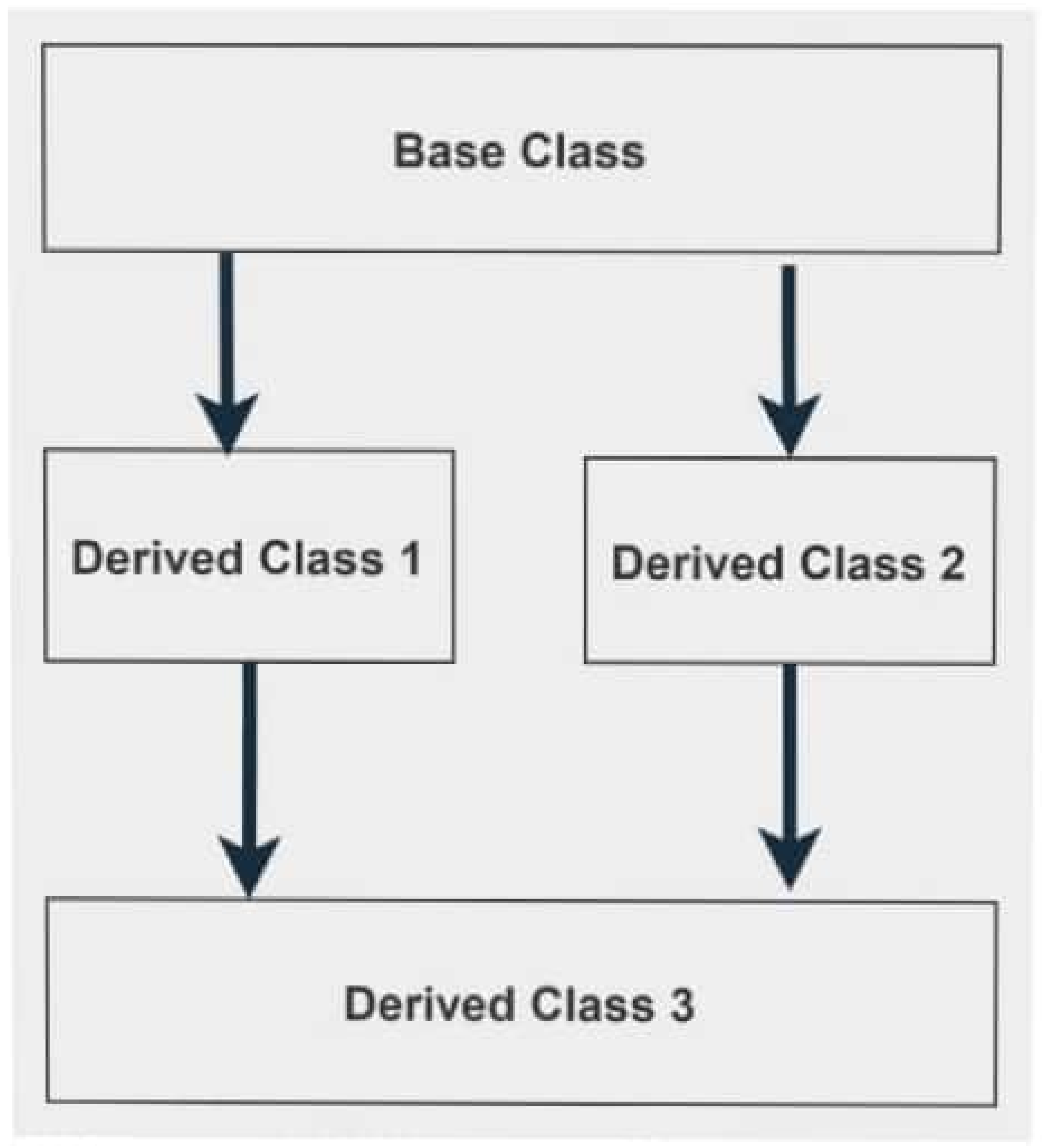
```
class Derived1 : access_mode Base
{
        //body of Derived1 class which inherit property from base class
};
Class Derived2: access_mode Derived1
{
        //body of Derived2 class which inherit property from Derived1 class
};
```

Hierarchical Inheritance - More than one (multiple) derived classes inherit property from a single base class.



```
class Derived1 : access_mode Base
{
        //body of Derived1 class which inherit property from base class
};
Class Derived2: access_mode Base
{
        //body of Derived2 class which inherit property from Base class
};
```

Hybrid Inheritance - It is a combination of both multilevel and hierarchical inheritance.

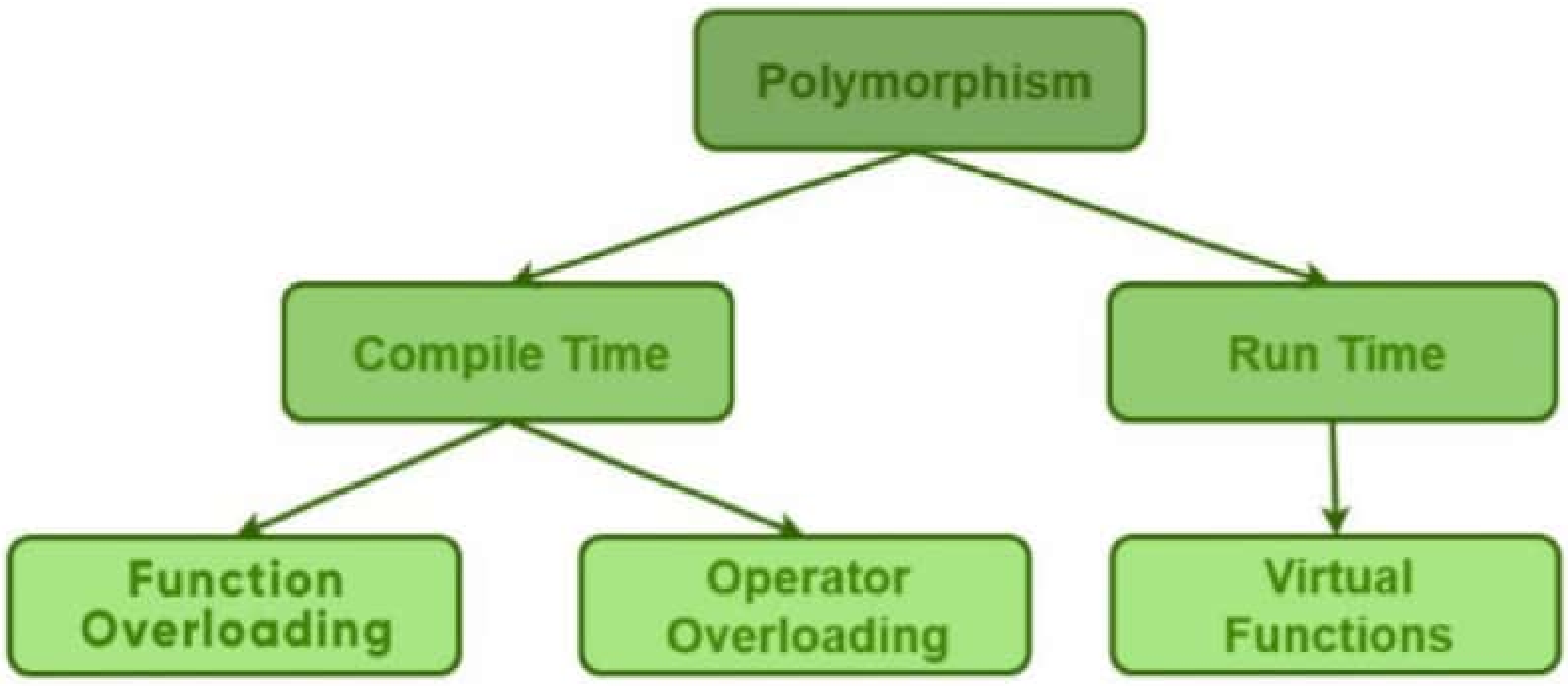


```
class Derived1 : access_mode Base
{
        //body of Derived1 class which inherit property from the base class
};
Class Derived2: access_mode Base
{
        //body of Derived2 class which inherit property from Base class
};
Class Derived3: access_mode Derived1 , access_mode Derived2
{
        //body of Derived3 class which inherits property from both Derived1
        and Derived2 class.
};
```

■ ■ ■

Polymorphism:

The word polymorphism means having many forms. That is, the same entity (function or operator behaves differently in different scenarios.



Compile time polymorphism: This type of polymorphism is achieved by function overloading or operator overloading.

Example of Function-overloading (Compile time polymorphism):

```
#include <bits/stdc++.h>
using namespace std;
class Chef
{
public:
   void func(int x)
   {
      cout << "value of x is " << x << endl;
   }
   void func(int x, int y)
   {
      cout << "value of x and y is " << x << ", " << y << endl;
   }
};

int main()
{

   Chef.obj1;
   obj1.func(7);
   obj1.func(85, 64);
   return 0;
}
```

Example of Operator overloading (Compile time polymorphism):

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

class Complex
{
private:
   int real, imag;
public:
   Complex(int r = 0, int i =0)  {real = r;   imag = i;}

   Complex operator + (Complex const &obj)
   {
       Complex res;
       res.real = real + obj.real;
       res.imag = imag + obj.imag;
       return res;
   }
   void print() { cout << real << " + i" << imag << endl; }
};

int main()
{
   Complex c1(10, 5), c2(2, 4);
   Complex c3 = c1 + c2;
   c3.print();
}
```

Runtime polymorphism: This type of polymorphism is achieved by Function Overriding.

Example of Function-overriding (Runtime Polymorphism):

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

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

class Derived : public Base
 {
  public:
   void print()
   {
       cout << "Derived Function" << endl;
   }
 };

int main()
{
   Derived derived1;
   derived1.print();
   return 0;
}
```