05-03-18 ST-2

① eg-①

class X
member A
member B

class Y
member A
member B
member C

Common features in both classes

class X
member A, B } common features of both classes are abstracted & kept in a base class

class Y
member → (derived class)

eg-②

Class point
private : x, y;
Public :
get();
display();

class : circle
private : x, y, r;
public :
void get()
void display()
void area()

base

class : point
private : x, y
public :
void get()
void display()

derived

class : circle
private : r
public :
void get()
area()

Base class — derived class
employee — Director . . .
shape — rectangle, triangle, circle

Derived class declaration :

class derived classname : access specifier base classname { }

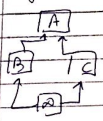
private - default
public
protected

Type of inheritance

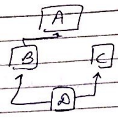
1. Single level — A → B
2. Multilevel Inheritance — A → B → C
3. Hierarchical — A (Univ) → B (student), C (teacher) → D, E
4. Multiple — A, B } more than one base class → C

Date __/__/__
Page ______

5. Hybrid



Hierarchial
multiple



(1) Single level

```
class base {
    int sem;
public:
    void bget() {
        cin >> sem;
    }
    void bdisplay() {
        Cout << sem;
    }
};

class derived : public base {

    Char x [20];
public:
    void dget() {
        Cin >> x;
    }
    void ddisplay() {
        Cout << x; } };
```

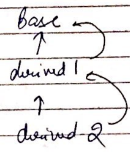
Derive class ka object bana hai

Date __/__/__
Page ______

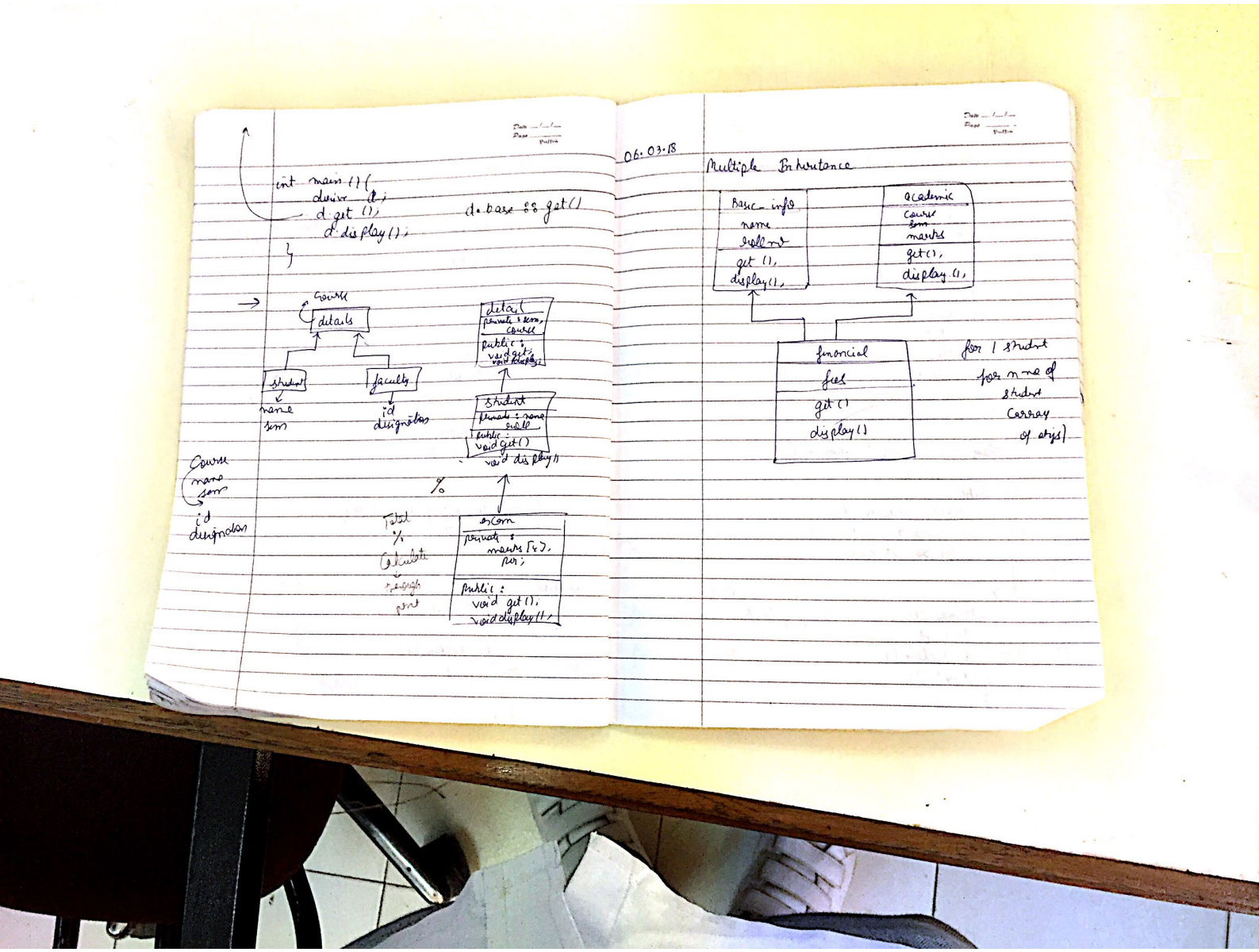
```
int main() {
    derive D;
    D.dget();
    D.ddisplay();
    D.bdisplay();
    D.bget();
}
```

Derived class → base class ki properties ko inherit kar sakti hai.

```
→ class base {
    int sem;
public:
    void get() {
        cin >> sem;
    }
    void display() {
        cout << sem;
    }
};
class derive : public base {
    char x [20];
public:
    void get() {
        base::get(); ←
        cin.getline (x, 20);
    }
    void display() {
        base::display(); ←
        cout << x;
    }
};
```



Or



Ambiguity in multiple Inheritance

```
class b1{
    int x;
public:
    void get(){
    }
};
class b2{
    int y;
public:
    void get(){
    }
};
class der : public b1, public b2{
    int z;
public:
    void display(){
        cin >> z;
        cout << z;
    }
};
int main(){
    der d;
    d.display();
    d.get();
}
```

b1::get();

07.03.18

Public / Private / Protected Inheritance in C++

```
class base{
public: int x;
protected: int y;
private: int z;
};
class derived : public base{
// x is public, y is protected & z is not
   accessible from derived.
};
class derive : protected base{
   x is protected, y is protected & z is not
   accessible
};
class derive : private base{
   x & y is private & z is also private};
```

Accessibility in Public Inheritance

for/ accessible for	private	protected	public
own class	Y	Y	Y
from 1st derived class	NO	Y	Y
from 2nd derived class	NO	Y	Y

If not specified it will be inherited privately.

In Protected Inheritance		
private	protected	public
Y	Y	Y
No	Y	Y
No	Y	Y

In Private Inheritance		
Y	Y	Y
N	Y	Y
N	N	N

Member access Control →

Accessing the public data →
1. by member function of that class
2. non mf of that class
3. mf of a friend class
4. mf of derived class if derived publicly.

```
① class example {
  public:
    int a;
    void get () {
      cin >> a;
    }
    void display () {
      ++a;        → valid (check)
      cout << a;
    }
};
```

② base = example

```
int main () {
  example e;
  e.a++;   → valid (check)
  cout << e.a;
}
```

③
```
class derive : public example {
  public:
    void display () {
      ++a;
      cout << a;
    }
};
```
} valid

when base class is inherited privately
```
class derive : private base {
  public:
    void get () {
      cin >> a; }
    void display () {
      ++a;        → check ? No
      cout << a;
    }
};
int main () {
  derive d;
  d.get ();      // error base :: get ()
  d.display ();  // base :: value a,
  cout << d.a;   // error
}
```
will not be allowed again

Accessing the private data →

* by member function of that class
* member function of friend class in which it's declared.

The public mf of drive class cannot access the private data member of base class irrespective of whether the derived class has been inherited publicly or privately.

```
class base {
    int a;
};
class drive : public/private base {
    public:
    void display (){
        ++a;   // Invalid
    }
};
int main (){
    drive d;
    d.get ();
    d.display ();
    cout << d.a;  → check? NO
}
```

If data member is private & → public/private will not be accessible

We will get permission on 1st drive class.

Accessing protected data member :

* by member function of its own class
* friend of class
* member function of derived class, irrespective of derived class inherited public/protected/private

```
class base {
    protected:
        int a;
};
class drive : public/private base {
    public:
        void display (){
            ++a;  // valid  Yes valid
            cout << a;
    }};
class drive2 : private drive {
    public:
    void data(){                    for private
        ++a;  ——→ Invalid check;
        cout << a;
    }
```

first bas then derive

Inheritance with Constructor

Case 1 → Constructor with no Parameter in both base & derived class

```
class base{
  public:
    base(){
    cout << " In base no Parameter Constructor ";
    }
};
class derive : public base{
  public:
  derive (){
    cout << " In Derive no Parameter Constant "; }
};
int main(){
    Clrscr();
    derive d;
    getch();
}
```

Constructor order → Base then derive } Execution

Case 2 → Constructor with no parameter in base & parametrized constructor in derived class

```
class base{
  public→
      base(){
      cout << " Base no Parameter Constant ";
      }
};
class derive : public base{
  int a, b;
  public:
   derive (int x, int y){
       a = x;
       b = y;
     cout << " In drive Const with 2 Parameter ";
    }
};
main(){
    clrscr();
   derive d(10, 15);
     getch();
     return 0;
}
```

Const = Constructor

Date __/__/__
Page ____ Puffin

Case 3 → Parametrized Constructor in both base & derived class

```
class base{
    int a;
  public:
    base (int a1) {
      a = a1;
      cout << "Base Single Parameter Const";
    }
    void show () {
      cout << "\n a = " << a;
    }
};
class derive : public base{
    int b;
  public:
    derive (int bb, int aa): base(aa){
        b = bb;
        cout << "Derived 2 Parameter Const";
    }
    void show () {
        base :: show ();
        cout << "b = " << b;
    }
};
```

argument that we want to assign

derive (int bb, int aa) base{
b = bb;
a = aa;

Execution → Down to up [Derive class ka constructor tab pahle derive class ka constructor ko pass jayega]

Print - up to down

Date __/__/__
Page ____ Puffin

```
int main(){
    clrscr();
    derive d (10, 20);
    d. show ()
    getch ();
    return 0;
}
```

Case 4 → Constructor with no Parameter in multiple inheritance.

```
class base1{
  public:
    base1 () {
    cout << "Base1 no Parameter Constant";
  }
};
class base2 {
  public:
  base 2 () {
  cout << "Base 2 no Parameter Constructor";
  }
};
class derived : public base1, public base2 {
  public:
    derived () {
      cout << "derived no Parameter const";
    }
};
```

base1 pehle chalega

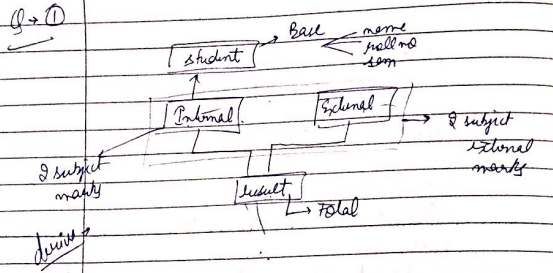
```
int main(){
    clrscr();
    derived d;
    getch();
    return 0;
}
```

Can 5 Parameterized Constructors in multiple Inheritance

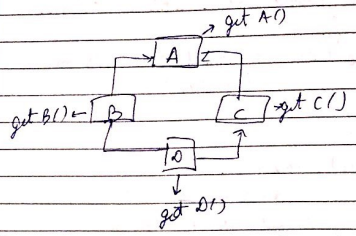
```
class base1{
private:
int x;
public:
base1(int x1){
    x = x1;
cout << "base1 single Parameter Constructor
    executed";
    cout << "x = " << x;
}
};
class base2{
    private:
        int y;
Public:
    base2(int y1){
        y = y1;
cout << "\n base 2 single Parameter Constructor
        Executed";

cout << "y = " << y;

}};
class derived : public base1, public base2{
public:
    derived(int xx, int yy): base2(yy), base1(xx)
{
cout << "\n derived contructor in the 2 Parameter,
        Executed";
}
};
int main(){
    clrscr();
    derived d(10,20);
    getch();
    return 0;
}
```

called first

Q → ①



Q → ②



If B also enter a and c also enter a then is error as same value is updated

whereas D can't find which path it should follow.

virtual is a keyword:

class B : public virtual A { }

Class C : virtual public A { }

class userdetails

① Input AdhNo, alt Mob no, landline no, email ID through constructor

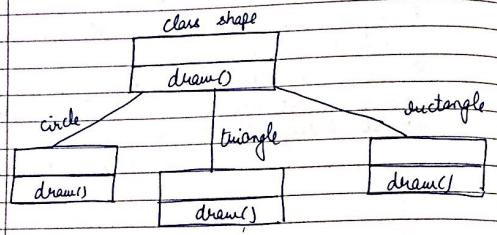
② Time friend function

→ static validate Email → check whether email ID is valid or not ("a", ".com")

→ static find Duplicate → else print "invalid"
→ check whether

12.03.18

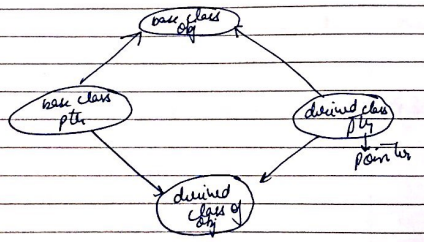
Polymorphism



Classification of polymorphism:

① Compile time or static binding eg ⇒ overloading function, operator overloading, template ⇒

② Runtime or dynamic (virtual function)




```
# class base {
  public:
  virtual void display(){
      cout << " Base class"
  }
};
class drive1 : public base {
  public:
    void display(){
      cout << " Derived class1 ";
};
class drive2 : public base {
    public:
      void display(){
      cout << " Derived 2 "; }
};
int main(){
  base * ptr;
  drive1 d1;
  drive2 d2;
  ptr = &d1;
  ptr -> display(); //base class print
  ptr = &d2;
  ptr -> display(); // base class print
}
```

function name should be same.

ptr will not take any address at compile time

But if we use virtual keyword in the base class it will call the derived class function only

class shape {

protected :
int a, b;
public :
void get() {
cin >> a >> b;
}
virtual void display() {
cout << " Base class "; }

virtual void display() = 0;

this is of no use so make it pure virtual virtual void display() = 0;

this means empty }

};
class rectangle : public shape {
public :
void display() {
cout << (a * b);
}
};
class triangle : public shape {
public :
void display() {
cout << (a * b)/2 ; }

};

int main() {
shape * s;
rectangle r;
r.get();
triangle t;
t.get();
s = &r;
s -> display();
s = &t;
s -> display();
}

class polygon {
protected :
int ht, wt;
public :
void get (int w, int h) {
wt = w;
ht = h;
}
virtual int area() = 0; // Pure virtual fun
void display() {
cout << this -> area(); }
};

Date / /
Page
Puffin

13.03.18

#

```
class rectangle : public polygon {

public:
    int area(){
        return (wt * ht);
    }
};

class triangle : public polygon{
    public:
        int area(){
            return (wt * ht)/2;
        }
};
int main(){
    rectangle r;
    triangle t;
    polygon * p1 = &r;
    p1.get(4,5);
    p1.display();
```

```
    polygon * p2 = &t;
    p2 -> get(7,8);
    p2 -> display();
}
```

* ST → 2 *