

22-06-2024

## PRINCIPLES OF OOP'S:-

1. class
2. object
3. Encapsulation
4. Abstraction
5. Polymorphism
6. Inheritance
7. Dynamic Binding
8. Message Communication

} [Object based]  
vanilla Based script  
Java Script

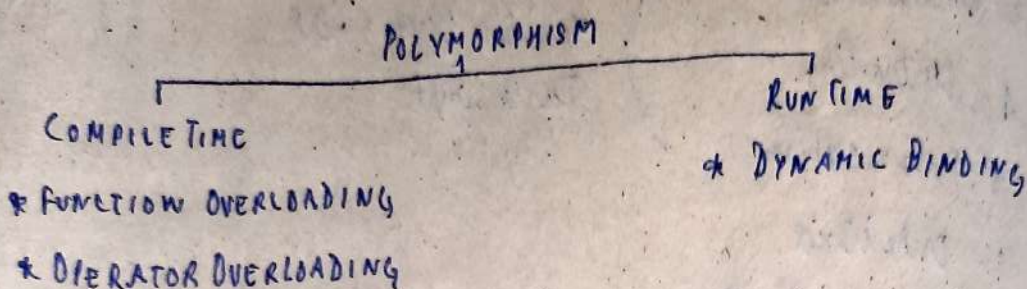
24-06-2024

1. CLASS:- It is a collection of similar objects.  
A class is a factory which produces different objects of similar datatype.
2. OBJECT:- An instance of a class is called an object.  
An object is a collection of ~~mem~~ properties i.e collection of members & methods ( ).
3. ENCAPSULATION:- It is a ~~process~~ mechanism that the members & the methods bounded together to protected from unauthorized access.
4. ABSTRACTION:- Creating new datatype through encapsulation is called abstraction.  
The datatype which is created by abstraction is called Abstracted Datatype [ADT]



The simplest example for an abstracted datatype is a "Class"

5. POLYMORPHISM:- Poly means many (or) more  
Morphism means Action



6. INHERITENCE:- It is a process by which 1 class object acquires the properties of another class object.

Q:- WAP to add complex numbers:-

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

```
class complex
```

```
{
```

```
private: int a, b;
```

```
public: void get()
```

```
{
```

```
cout << "Enter a, b";
```

```
cin >> a >> b;
```

```
}
```

```
void add (complex p, complex q)
```

```
{
```

```
a = p.a + q.a;
```

```
b = p.b + q.b;
```

```
}
```

2 3

5 6

7 + 9i

```
void put()
```

```
{
```

```
    cout << a << "i" << b;
```

```
}
```

```
}
```

```
main()
```

```
{
```

```
    complex p, q, r;
```

```
    p.get();
```

```
    q.get();
```

```
    r.add(p, q);
```

```
    r.put();
```

```
}
```

O/p:-

Enter a, b: 2 3

Enter a, b: 5 6

7 + i9

this :- This is a pointer which always refers "current address".

→ Using namespace std;

```
#include <iostream>
```

```
class complex
```

q, p → object

```
{    private : int a, b;
```

```
    public : void get (int a, int b)
```

```
{
```

```
        this -> a = a;
```

```
        this -> b = b;
```

```
}
```

```
void add (complex p, complex q)
```

```
{
```

```
    a = p.a + q.a;
```

```
    b = p.b + q.b;
```

```
}
```



```
void put ( )
```

```
{
```

```
cout << a << "+" << b;
```

```
}
```

```
};
```

```
main ( )
```

```
{
```

```
Complex p, q, r;
```

```
p.get (5, 6);
```

```
q.get (3, 2);
```

```
r.add (p, q);
```

```
r.put ( );
```

```
}
```

o/p:- 8+18i

ARGUMENTS  
 $\overline{R}$   
(1,  $\omega$ )

similar  
operations

**METHOD OVERLOADING:-** Reusing same name for different methods, with different number of arguments [or] with different type of arguments with (or) without same return type is called Method overloading.

It provides opportunity of giving same name for similar operation methods.



Ex:- using namespace std;  
#include <iostream>

class complex

{

private: int a, b, c;

float x, y, z;

public: void get (int a, int b)

{

this -> a = a; this -> b = b;

}

void get (float x, float y)

{

this -> x = x; this -> y = y;

}

void add()

{

c = a + b;

z = x + y;

}

void display()

{

cout << c << "Im" << z;

}

};

main()

{

complex p;

p.get (5, 6);

p.get (3.2f, 2.3f);

p.add();

p.display();

}

O/P:-

Enter a, b: 1 2

Enter a, b: 3 4

4 + i6



## STRUCTURE Using POINTERS :-

```
#include <stdio.h>
->int main ( )
```

C CODE

```
{
    struct book
    {
        int a;
    };
    struct book p = {5}, *q;
    q = &p;
    printf ("%d", p.a);
    printf ("%d", p->a);
}
```

O/P:-

5

5

(O/P)

```
-> #include <iostream>
```

C++ CODE

```
using namespace std;
```

```
main ( )
```

```
{
    struct book
    {
        int a;
    };
    struct book p = {5}, *q;
    q = &p;
    cout << p.a << endl;
    cout << q->a << endl;
}
```

O/P:-

5

5