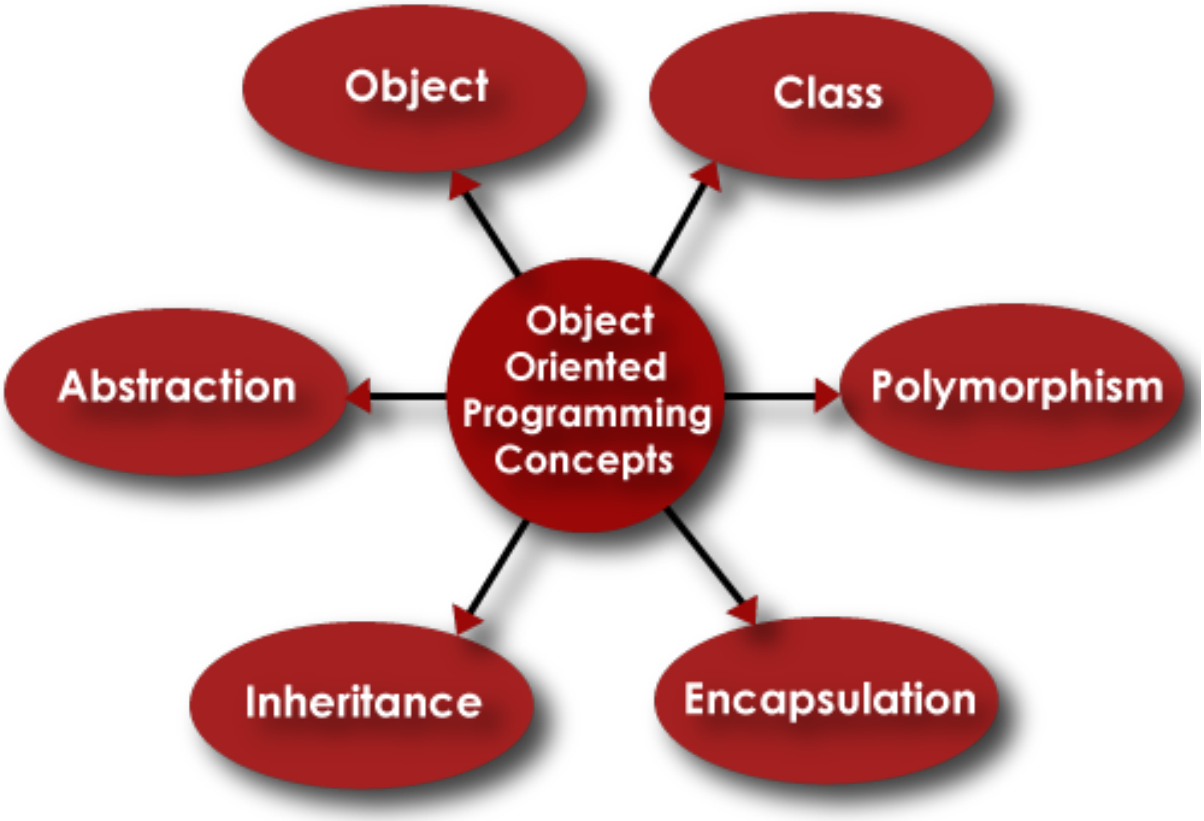
Object Oriented Programming Concepts




1. Class
2. Objects
3. Encapsulation
4. Data hiding
5. Inheritance
6. Polymorphism

1. Class. – Blue print / Extension of structure / is a keyword

```
class Person
{
        int age;                // data member
        char name[15];          // data member

        void display()  // member functions
```

```
        {
        ……
        }
};
```

Make use of Access Specifiers – Public, Protected, Private.

**By default – private**

In C++, there are three access specifiers:

- `public` - members are accessible from outside the class
- `private` - members cannot be accessed (or viewed) from outside the class
- `protected` - members cannot be accessed from outside the class, however, they can be accessed in inherited classes

2. **Object** : is instance of a class

To access data mem & mem functions we require VARIABLE, this variable is called OBJECT

```
Class Emp
{      int a,b;      };
Emp e;  // e is object, using this we can access members
```

Example:

```
class Emp
{
int id;
char name[12];
void read()
{
   cin>>id;
   cin>>name;
}

void display()
{
cout<< id << name <<endl;
}
};

int main()
{
```

```
    Emp e;
    e.read();
    e.display();

}
```

**3. Encapsulation – binding of data & functions into a single unit**

**[By default – Private – hence data is binded]**

```
class Emp
{
int id;
char name[12];
void read()
{
    cin>>id;
    cin>>name;
}

void display()
{
cout<< id << name <<endl;
}
};
```

ORRRRRRRR

```
class Emp
{

private:
int id;
char name[12];

public:
void read()
{
    cin>>id;
    cin>>name;
}

void display()
{
cout<< id << name <<endl;
}
};
```

```
int main()
{
    Emp e;
    e.read();
    e.display();

}
```

**Data Hiding & Abstraction is it same????**

**Abs—used to create user defined data types with properties but will not show details by using private,ignore inessential details.- use – hide complexity**

**D.Hiding – protects data against unauthorized access – using private. Used to encap the data,**

4. **Abstraction. – Hide the internal/background details from outside users**

```
class A
{

    private:
            int a=10;   // abstracted - hidden
    public:
    void display()
    {
    cout<< a<<endl;
    }
    };

int main()
{
    A obj;
    obj.display();

}
```

One more example is header file

#include<math.h>

Sqrt() when used we shud use math.h

Math.h is hidden from user.

5. Inheritance: one class extends the properties of another class

```
Class A
{
……
}

Class B: public A

{
……
}
```

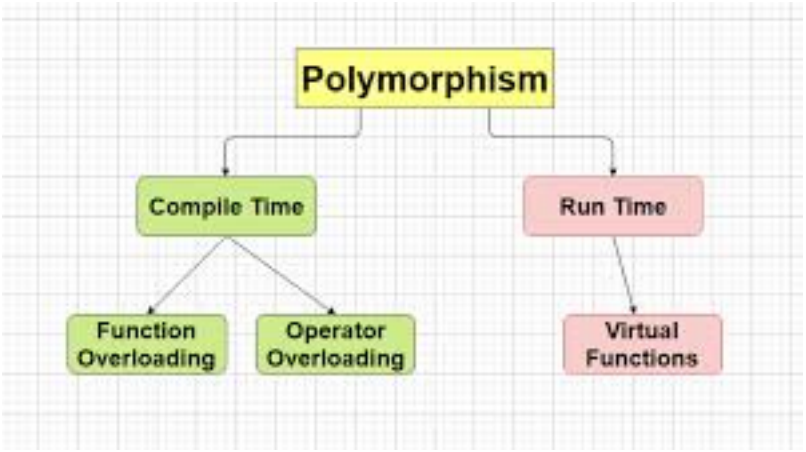
Class A is called **Parent/Super/Base class**
Class B is called **Child/sub class**

Uses : 1. Reusability. 2. Extensibility

6. Polymorphism: One object behaves as different objects in different situation

   One Person → a. Student in college
   b. Intern at Company
   c. Passenger in College bus
   d. Son/Daughter at home etc

   Two Types: 1. Compile time
   2. Run Time




1) Compile /Static Polymorphism
   2 ways: Func – Overloading & Func Overriding

   **a)** Func Overloading: **2 or more functions can have same name but different parameters**

Ex:

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

Class Add
{
public:
void sum(int a, int b)
{
cout<<a+b<<endl;
}
void sum(int a, int b, int c)
{
cout<<a+b+c<<endl;
}

        int main()
        {
        Add obj;
        obj.sum(10,20);
        cout<<endl;
        obj.sum(10,20,30);
        }
```

   **a)** Func/method overriding: **2 or more functions can have same name with same parameters**

**Mandate: Use inheritance concept must be der.**
**Reason: cannot be done within a class.. therefore we use**

**inheritance – (Super class & Child Class)- During execution its confusing**
**Hence we require – Derived class & Base Class**

Example:

```
Class Base
{
public:
void show()
{
cout<<"Base class"<<endl;
}
};
class Derived: public Base
{
public:
void show().        // overriding done here same
{
      cout<<"derived class"<<endl;
}
};
```

```
int main()
{
Base b;
Derived d;
b.show();
d.show();
}
```

Note:

1.if using same name & same parameter use – overriding – use super & child class

2. if using same name & differen parameter – use overloading – same/single class is enough

**Operator Overloading**

It is a type of polymorphism in which operator is overloaded to give user defined meaning.

Ex:

a.add 2 numbers – we use + operator
b. concatenate two strings use + operator

So we are using both + & + … this is called operator overloaded.

Example:

```
Class Rec
{
public:
Rec()
{
l=0;
b=0;
}
void operator++()
{
l+=2;
b+=2;
}
void display()
{
cout<<l<<endl;
cout<<b<<endl;
}
};


int main()
{
Rec R;
//before increment
R.display();
++R;

//after increment
R.display();
```

```
return 0;
}
```

output:

```
0
0
2
2
```