Object Oriented Programming System

→ Object oriented programming is a programming paradig-m that revolves around the concept of object, which can contain data and functions to manipulate the data.

* Features of oops -

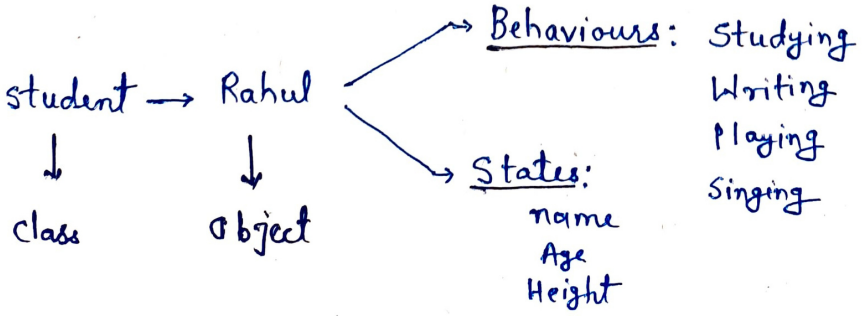
1. Inheritance
2. Abstraction
3. Encapsulation
4. Polymorphism

* Class -

A class is a blueprint or a template for creating objects that defines a set of variables, methods and properties that are common to all objects of that class.

* Object -

An object is an instance of a class or an entity which gets created using class and it represents the state and behaviour.



* Important points to remember -

→ Class is a group of objects.

→ Class is a design of objects.

→ Many objects can be created from same class

→ Object consumes memory equal to the sum of size of all data members.

→ Member function don't occupy memory.

→ Member functions are called depending on objects.

→ Dot(.) operator is used for accessing members of objects.

* Inheritance -

Inheritance means designing an object or a class by reusing (acquiring) the properties of the existing class and object.

Eg:- A old style television is transformed with extra features into slim and smart television where it re-used the properties of old television.

* Abstraction -

It means hiding internal details and showing the required things.

Eg:- Consider a man driving a car, while driving he focus on using of steering, gear, and acceleration. He does not require to know the inner mechanism of the car.

* Encapsulation -

It is a process of wrapping the data and the function together. It can assume as a protective wrapper that stops random access of code defined outside the wrapper.

Eg:- Complete television is single box, where all the mechanism are hidden inside the box. All are capsuled.

* Polymorphism -

Polymorphism is a concept in which we can execute a single operation in different ways.

* Data hiding -

Data hiding is an important concept for building secure software.

→ By hiding data, a programmer can avoid mishandling of data by the code outside the class.

→ For hiding data, we use 'private' keyword. (Default)

→ So, when the data is made private, we can't access the data outside the class.

→ It is accessible only within the class.

Eg:- Actual operation of the television is performed in the circuitry which is done by pressing a button. So the circuitry is data and operations are methods where the data is hidden inside the box.

Eg:-

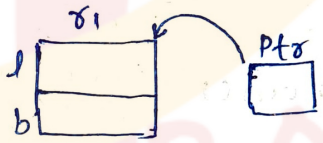
```
class Rectangle
{
  Public :
      int length;
      int breadth;

      int area()
      {
         return length * breadth;
      }
      int perimeter()
      {
         return 2*(length + breadth);
      }
};

void main()
{
   Rectangle r1, r2;
   r1.length = 10;
   r1.breadth = 5;
   cout << r1.area();

   r2.length = 15;
   r2.breadth = 2;

   cout << r2.area();
}
```

Data members

Functions

* Pointer to an object

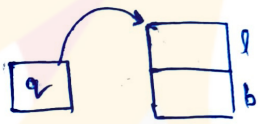
```
class Rectangle
{
  Public :
     int l;
     int b;
     int area()
     {
        return l*b;
     }
     int perimeter()
     {
        return 2*(l*b);
     }
};
int main()
{
   Rectangle r1;
   Rectangle *Ptr;
     Ptr = &r1;
     Ptr -> l = 10;
     Ptr -> b = 5;
   cout<< ptr -> area();
   cout << Ptr -> perimeter();
}
```



(It will create in stack)

* Create an object in heap (using 'new' keyword).

```
int main()
{
    Rectangle *Ptr = new Rectangle;  // Dynamic Allocation

    Ptr->l = 10;
    Ptr->b = 5;

    cout << Ptr->area();
    cout << Ptr->perimeter();

    Rectangle *q = new Rectangle;
    - - - -
    - - - -
    - - - -

}
```



Note:

Rectangle r; // Object is created in stack

Rectangle *p = new Rectangle; // Heap

or, Rectangle *p = new Rectangle();

* Accessor(get) and Mutator (set)-

→ These are used for avoid mishandling of data.

Accessor - Used for knowing the value of data members.

Mutators - Used for changing value of data members.

Eg:-

```
class Rectangle
{
  Private:
      int length;
      int breadth;
  Public:
     void setLength (int l)
      {
          if (L>0)
           length = l;
          else
            length = 0;
      }

     void setBreadth (int b)
      {
          if (b>0)
            breadth = b;
         else
            breadth = 0;
     }
```

Mutator function

```
    int getLength ()
    {
        return length;
    }
  int getBreadth()
    {
        return breadth;
    }
 int area()
    {
        return length * breadth;
    }
 int perimeter()
    {
        return 2*(length + breadth);
    }
};
int main()
{
    Rectangle r1;
    r1. setLength (10);
    r1. setBreadth(5);

    cout << r1. area ();
    cout << r1. perimeter();

    cout << r1. getLength();

    cout << r1. getBreadth();
}
```

Accessor Function