1/04/2023

# OOPs

Full form is object oriented programming. In this programming technique in which everything is revolving around object. But now question comes to our mind that what is an object?

Object is an entity having 2 things namely state/properties and behaviour/methods.

↳ functions

It is used to solve real life problems & it becomes easy to understand things. Our code becomes readable, we can also reuse the code.

Class
int num;
string str;
bool ans;
} Pre-defined datatypes

Can we create our own datatype? The answer is yes and with the help of classes, we can create user defined data type.

↱ User defined datatype
Animal
↳ name (string)
↳ int age (integer datatype)

Note→ Actual entity → object
Blueprint / structure definition → Class

We can say that object is an instance of class.

Syntax to create class
class class_name {

};
↳ Semicolon to be used here.

Ex→ class Animal {

};

## Size of empty class

```
class Animal {
};
main() {
    cout << sizeof (Animal) ; → 1 is the output
}
```

Size of empty class comes out to be 1 byte here. If it was given 0 byte, then we won't be able to track its existence else we won't be able to track the existence if it was given 0 size.

Ex →
```
class Animal {
    int age;
    int weight;
};
sizeof (Animal); → 8 bytes
```

Ex →
```
class Animal {
    int age;
    int weight;
    char ch;
};
sizeof (Animal); → 12 bytes. Find out why?
```

## Creating an object of class

1) Static allocation

```
Animal ramesh;
↑ class name
       ↳ name of object.
```

If we want to access properties of ramesh, we will be using the . (dot) operator.

ramesh.age ;

Access modifiers → Protected (used in inheritance)
→ Defines the scope.

1) public → can be accessed outside & inside the class.
2) private → can be accessed inside the class.

Note → By default everything is marked as private.

```
class Animal {
        public : → Access modifier
        ----
        ----
};
```

Ex →
```
Class Animal {
        private :
            -----  } Can be accessed within class
            -----  }
        public :
            ---  } Can be accessed outside the
            ---  } class.
}
```

Calling member functions of class.

```
class Animal {
        public :
            void eat () {
                cout << " I am eating " ;
            }
};
```

In main function, write

```
Animal ramesh;
ramesh.eat();
```

O/p → I am eating

Terminologies

1) The state or properties declared in class is known as data members.
2) The behaviour defined in the class is known as member functions.

Note → If we want to access private member outside the class, we can do so with the help of getter & setter.

```
class Animal {
    private:
        int weight;
    public:
        void setWeight (int w) {
            weight = w;
        }
        Void getWeight () {
            cout << weight;
        }
};
```

Setter & Getter

In main function

```
Animal ramesh;
ramesh.setWeight (101); → Setter
ramesh.getWeight (); → Getter
```

Dyanamic memory allocation

There are 2 types of memory available namely stack and heap.

* Stack → smaller memory & static allocation is done here.
* Heap → larger memory & dyanamic allocation is done here.

Syntax of dyanamic memory allocation

1) int * a = new int ;
   (a: address stored in pointer. new int: Returns address)
2) int * arr = new int [5] ; → Array is dyanamically created & has size = 5.

Pointer arr is present in the stack memory pointing to the base address of the array which is present in heap memory.

Space allocated in heap won't be deallocated automatically. To deallocate the memory, delete keyword to be used else heap memory will get full.

1) delete a ; → To delete variable
2) delete [ ] left ; → To delete array

By the dyanamic memory allocation, we can create object also.

Animal * suresh = new Animal ;

```
(*suresh).age = 15;
(*suresh).name = "billi";
```
} 1st way

```
suresh → age = 15;
suresh → name = "billi";
suresh → eat();
```
} 2nd way

this keyword
Pointer to current object & hence arrow operator is used.

```
class Animal { int weight;
    public:
        void setWeight (int weight) {
            this → weight = weight;
        }
};
```

↑ Which weight, class or object?
↳ To vanish ambiguity

Note→ this can not be used outside the class
Also we can use it like,

```
(*this).weight = weight;
```
↳ dereference

Constructor
The job of constructor is object initialization.
It is a function having same name as that of class & does not have any return type

Syntax

```
class Animal {
    public:
    Animal() {
        cout << "Constructor called";
    }
};
```

The above is an example of default constructor. Whenever object is created, constructor will be called.

Parameterized constructor

```
class Animal { int age;
        public :
            Animal (int age) {
                this → age = age;
            }
};
```

In main function,

Animal a; → Default constructor is called

Animal b(50); → Parameterized constructor is called.

Note → If no constructor is written by us, there is a constructor already present

Copy constructor

```
class Animal { int age;
        public :                    ↗ Reason is mentioned below
        Animal (Animal &obj) {
            this → age = obj.age
            cout << "Copy constructor";
        }
};
```

In main function,

Animal a = b; → Copy constructor called

↑ 1st way

Now as it is pass by value, a copy is created & hence again copy constructor is called & this goes on and hence we need to pass it by reference.

2nd way ⇒ Animal animal1 (a);

We use copy constructor to create deep copy as by default shallow copy is being made. But now question comes to our mind that what is shallow & deep copy. (To discuss in next class)

Destructor

Used to free the memory.

1) Same name as class
2) No return type
3) Has ~ sign infront of it.

* Static allocation ⇒ Automatically called /calls the destructor
* Dyanamic allocation ⇒ Manually we need to call the destructor by delete keyword

Syntax

```
class Animal {
        public:
            ~Animal() {
                cout << "Destructor called";
            }
};
In main function,
  Animal suresh; // Destructor automatically called.
```

→ Manual deletion required

```
Animal * suresh = new Animal;
delete ramesh; // Destructor called
```

Note → When we move out of scope, then destructor in case of static allocation will be called.