## * Object Orientated Programming (OOPs):

↓

Programming technique in which only interact with only objects. (revolves around objects)

## * Object:

- It gives descriptive information of an entity.
- It consists of properties & methods/function.
- It is an instance of a class.

## * why we need:

(1) Real life Application. (2) Readable (3) Re-useable.

## * Classes:

- It is a user-defined datatype / Data structure.

ex: custom datatye → solution → int, bool, string.

An user can create its own datatype of class type.

- It is a blue-print of an object.
- class Animal { → Syntax. };
- It consists of Data Member & Member functions.

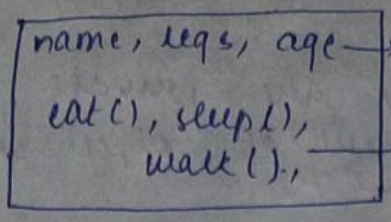
Example: Dog → 4 legs, 2 eyes → state/properties.

Dog → eat, sleep → Behaviours/Methods.

## * why do we need class?

We have pre-defined datatypes such as int, bool, char etc which have their own functionalities. There might be some situation where we need to create our own datatype which perform some task according to us and classes are the best way to define our own datatypes.

Example:



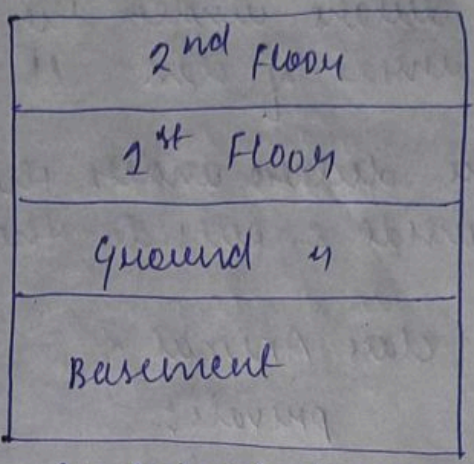

class Animal

* To understand in just simple way:

An architect creates the blue-print of a building. i.e


Blue-print of a house.

It is said to be class. i.e blue-print of an object




An Actual House which is said to be an object (Instance of class) → properties → Behaviours

* Does empty class consists of some size:

→ create a class named Animal

```
class Animal {
}
```

→ an empty class.

* It consists of size of 1 byte but it is empty. because as we have created a class so in the memory class Animal is being provided with minimum size. i.e Minimum possible size is allocated to empty class so that it could be recognized i.e its existence.

That's the reason the empty class is allocated with size of 1 byte (Minimum possible size)

## Access modifiers:

Access modifiers of C++ are private, protected & public are important component of object oriented programming.

→ By default all the members of class are under private section. Here member means → Data Member / Properties
→ Member Functions / Behaviours.

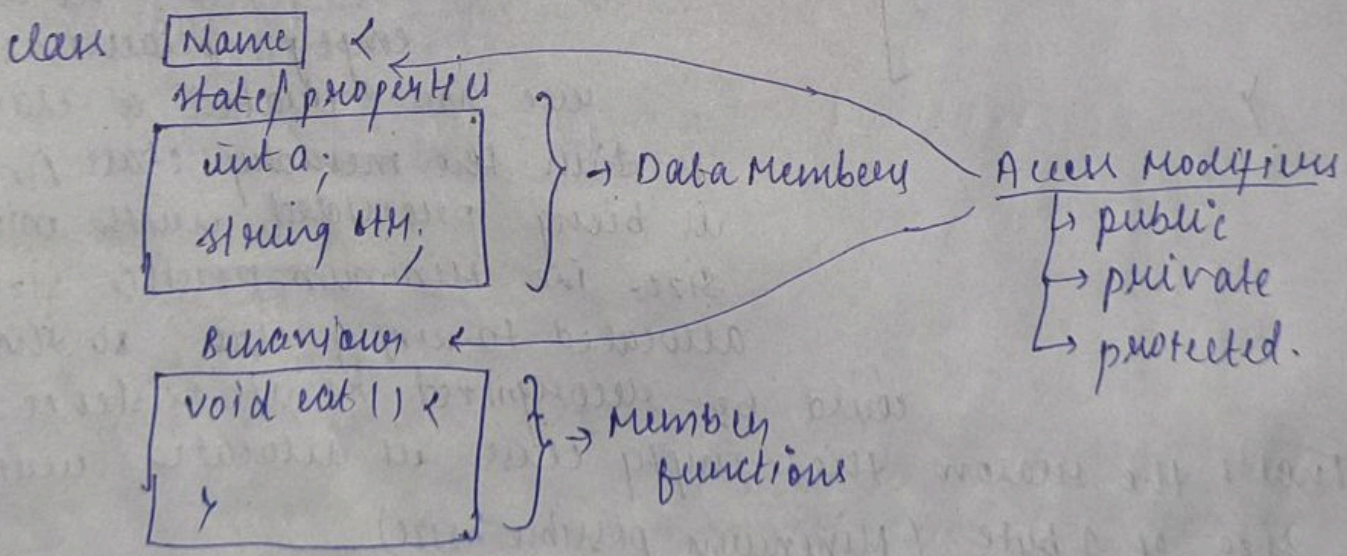
→ member declare under the private section can only be access inside of class. It can't be access outside of class.

→ members declare under the public section can be accessed both inside & outside the class.

ex:
```
class Animal {
    private:
        int age;
};

Animal a;
a.age ] → accessing member under privat section outside of class can't be accessed error.
```

→ Representation




};

* How we can access the members declare under the private outside the class?

→ we can do so with getters and setters. These are functions

fetch the property — set the property

```
class Animal {
    private:
        int weight;
    public:
    void getWeight() {
        return weight;
    }
    void setWeight(int w) {
        weight = w;
    }
};
int main() {
    Animal lion;
    lion.setWeight(400);
    cout << lion.getWeight();
}
```

getWeight → it is used to retrieve the property. i.e get back the ~~raw~~ property

setWeight → it is used to set the value.

output:

400 → you can see weight was under private section but we were unable to access with getters & setters

* How to create an object an access the members with the help of object:

→ we can create object in two types first for static memory allocation second for dynamic memory allocation.

① static

```
class Animal {
    int age;
    string name;
    void eat() {
    }
    void sleep() {
    }
};
int main() {
    Animal a;
```

↳ Here a is object

```
    cout << a.age;
    cout << a.name;
}
```

↳ this will give us an error bcz by default all members are in private section. private section can't be access outside of class.

→ If you want to access the properties / behaviours of an object it can be done with the help of dot operator. i.e a.age;
a.name;

ex: class Animal {

public:
int age;
string name;

void eat() {
cout << "Eating" << endl;
}
void sleep() {
cout << "sleeping";
}

→ Now all members are marked under the public section.
∴ It can be accessed both inside & outside of a class.

Now if we write / create an object and want to access D.M & M.F of a class it will happen bcz all are under public section. And will return garbage value.

}

int main() {
Animal a;
cout << a.age << endl;
cout << a.name;

output
2 ] → Garbage value will print.

a.age = 12
a.name = "lion";
cout << a.age << endl;
cout << a.name;

output: 12
lion.

int main() {
Animal a; → object
a.eat()
a.sleep() ] → accessing member functions.

output:
Eating
sleeping.

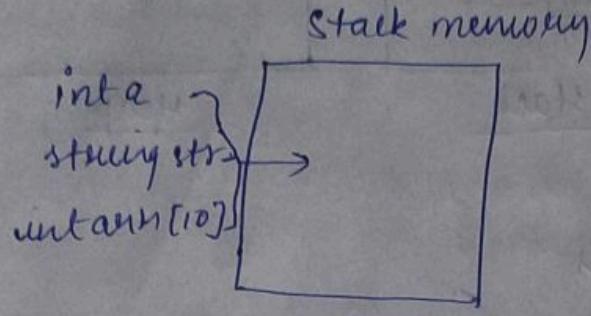
* Note: In most of cases the members of a class are marked as private this is due to security purpose bcz anyone can access it., to access the private section we can use getters &

sizes.

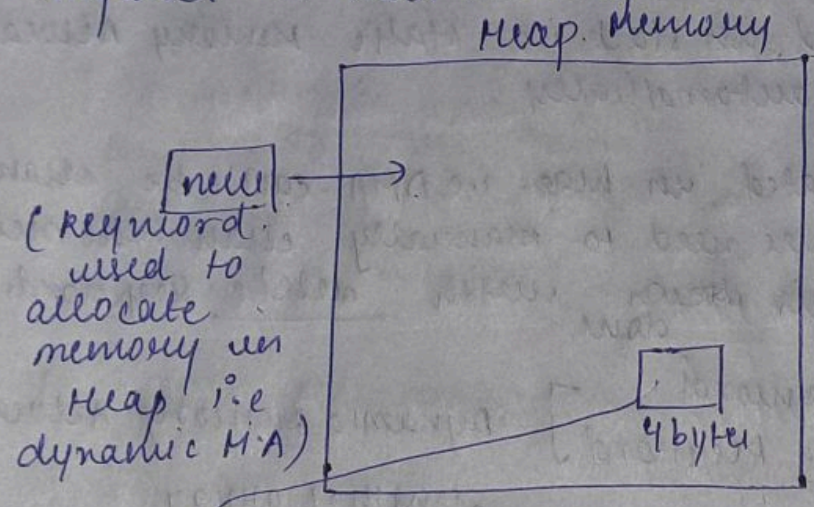
## * Dynamic Memory Allocation

The Now we know about static memory which is stored in the form of stack and ~~non~~ static memory is also limited. i.e stack memory is limited.

i.e




We also have dynamic memory allocation which is stored in a form of Heap and has large memory compared to stack memory.




Ex: new int [new it create heap memory of 4 byte as new keyword allocate the heap memory]

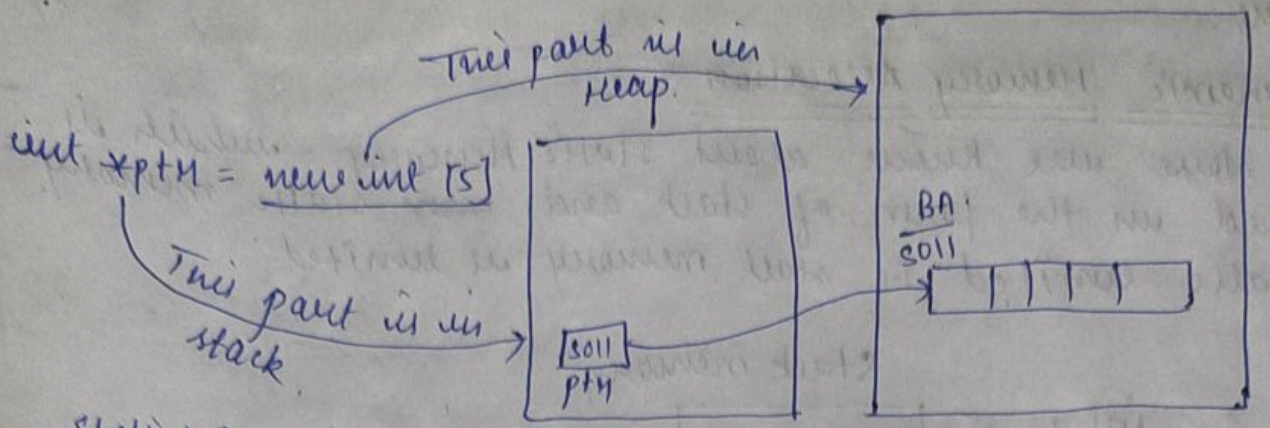
new int will hold the address i.e B.A of stored in Heap Memory.

int *a = new int;

↳ pointer, as pointer hold the address

int * arr = new int [len];

↳ create heap memory for an array of allocate length len which return B.A of an an array.




Static MA · Dynamic MA · Stack · Heap

int *a = new int

this part is stored in stack memory holding B.A

this part is stored in heap memory which returns the B.A/address

* The memory allocated in stack i.e static memory Allocation. can be cleared automatically.

* The memory allocated in heap. i.e DMA can't be cleared automatically. we need to manually clean the memory allocated & this is done with delete keyword.

Allocation → new keyword
Deallocation → delete keyword
] → Dynamic memory Allocation.

→ new int, new char, new char[7]. (7 = length of array; char = array of char type)

→ delete a,

→ Now how to create create object in Dynamic memory allocation.

→ In static

```
class Animal {
public:
    int age;
    string str;
    void eat();
};
```

```
int main() {
    Animal a;
    cout << a.age;
    cout << a.str;
    a.eat();
}
```

Above ex ni of creating object in static memory Allocation

→ In Dynamic

ex:
```
class Animal {
    public:
        int age;
        string name;
        void eat() {
        }
        void sleep() {
        }
};
int main() {
    //Dynamic
    Animal *b = new Animal;
    b.age = 15;
    b.weight = 104;
    b.name = "tiger";
}
```
→ it will throw back error bez Animal *b → holds address not actual value.

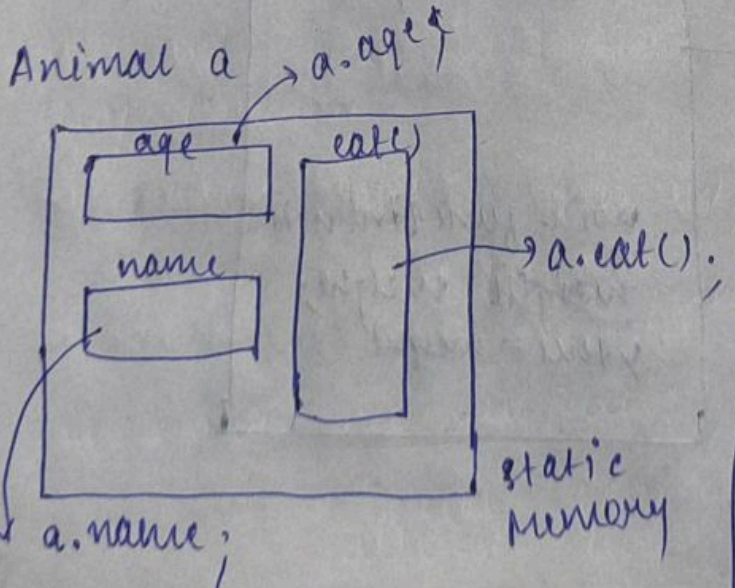
```
int main() {
    //Dynamic
    Animal *b = new Animal;
    (*b).age = 15;
    (*b).name = "tiger";
}
```
→ *b will not hold address it will be actual object. The code will run.

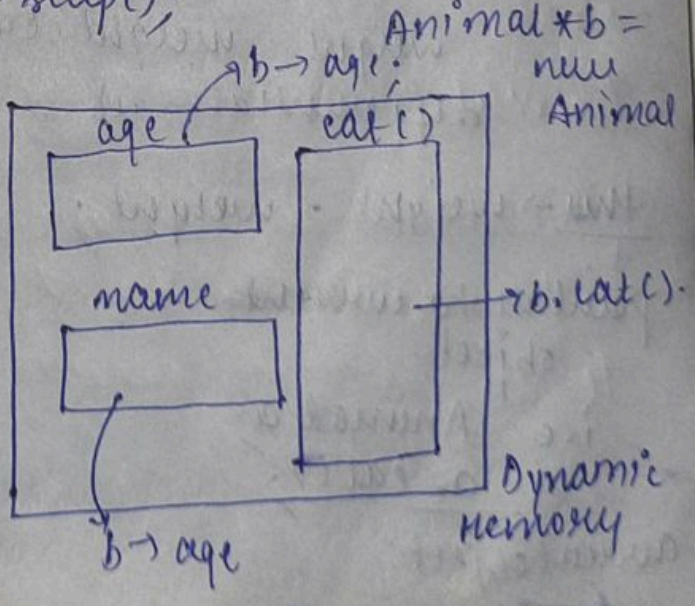
Alternative way

① `(*b).age = 15;`
`(*b).name = "tiger";`

② `b→age = 17;`
`b→name = "snake";`

`b→eat();`
`b→sleep();`



```
class Animal {
private:
    weight;
public:
    int getweight() {
        return weight;
    }
    int setweight(int weight){
        weight = weight;
    }
};
```

Now we can see that in setweight weight = weight which is object & which value need to be taken.

How to differentiate both weight.

→ we can differentiate it with ~~this~~ this keyword

this: → it is pointer to current object.

this → weight = weight;

- this → pointer
- weight → class weight declare under private section.
- weight → weight which we take input from user.

```
class Animal {
private:
    int weight;

    void func(int weight)
    {
        weight = weight;
        this → weight
    }
}
```

weight = weight can't differentiate it.

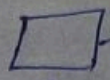
this → weight = weight;

this → pointer to current object.

i.e Animal a;
a.eat();

a → current object.

a.eat()
↓
current object.

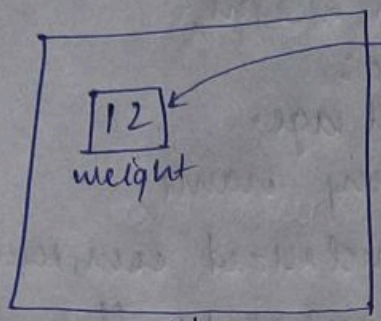


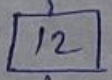

→ this pointer (which point out to the current object)

## Two ways to use this pointer:

this → weight (dynamic Memory Allocation)

(*this).weight (static Memory Allocation)




we want to store this weight inside object weight to diff variable we can use this pointer

this→weight = weight;

↓

current object ki weight wali property

## *Object creation:

**constructor:** • special member function which initilize the values to the data member of a class.
also said object initializer

• Same name as class name.
• No return type.

① Default constructor:

```
class Animal {
    private:
        int weight;
    public:
        int age;
        string name;

    //default constructor
    Animal() {
        this->weight = 0;
        this->age = 0;
        this->name = "";
        cout << "Constructor called";
    }
};
```

*By default constructor call itself whether static or dynamic. In this we are manually creating a constructor. That means it overwrites the public constructor.

```
int main {
    Animal a;
    Animal *b = new Animal;
}
```

② Parameterized constructor: Passing Parameter

```
class Animal {
  private:
    int weight;
  public:
    int age;
    string name;
    //parameterized constructor
    Animal (int age) { // single parameter
        this->age = age;
    }
    Animal (int age, int weight) { // double parameter
        this->age = age;
        this->weight = weight;
    }
    Animal (int age, int weight, string name) {
        this->age = age;
        this->weight = weight;
        this->name = name;
    }
};
```

```
int main () {
    Animal a;
    Animal *b = new Animal;
}
```

default

```
Animal a(1.0);
Animal *b = new Animal(10,100);
```

parameterized constructor

```
Animal *b = new Animal(10,100,"Subhat");
```

* Parameterized constructor is used to pass parameter to constructor named as class name.
It is used to provide values to the data members of a class.

## * Copy constructor:

Creating a new object which is copy of another object said to be copy constructor.

ex:

```
class Animal {
    private:
        int weight;
    public:
        int age;
        string name;
    //copy constructor
    Animal(Animal obj){
        this->age = obj.age;
        this->age = obj.weight;
        this->name = obj.name;
    }
}
```

```
int main(){
    Animal b = a;
    Animal b(a);
}
```

→ two ways to create copy constructor.

pointer to current object i.e this

current object

This code will give error, because

Animal(Animal obj) { } → PBV

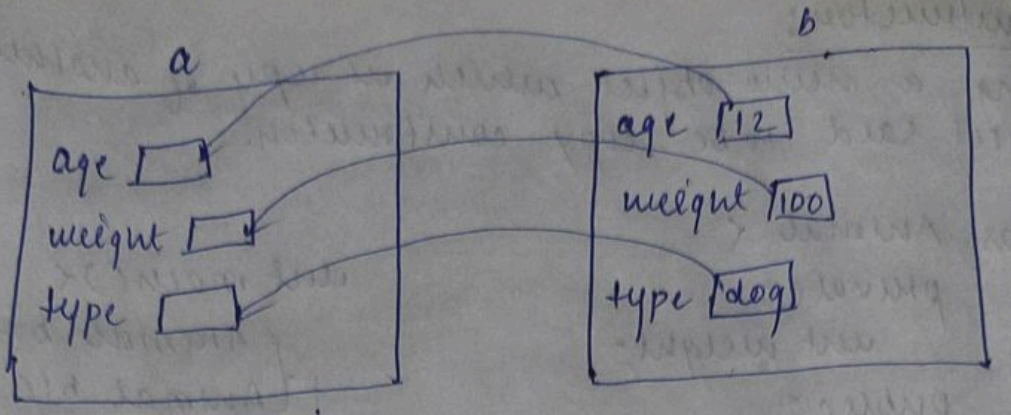
Animal a = b;

↓

Then it is copy constructor and now call the copy constructor.

Here we can see that Animal obj is pass by value ∴ It will copy of an object and then again it is pass by value & it will create copy of an object goes on.... we see we are stuck inside a infinite loop.

To get rid of it we used to pass by reference which will not create the copy of object over again & again.

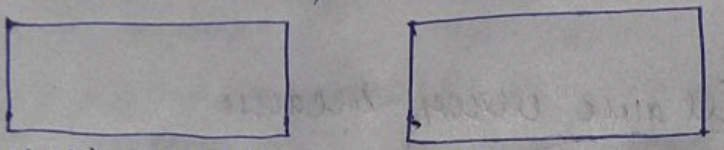



Animal obj
copy

Animal b(a) || Animal b = a

* Destructor : Denoted by (~)

- member func same as class name.
- comes in practice when we want to empty the space allocated by constructor.



static memory Allocation

Destructor calls automatically in this case

Dynamic

We need to manually call the destructor to free the space. This is done with delete keyword.

- No returntype - No input parameter.

ex:

```
class Animal {
    private:
        int weight;
    public:
        int age;
        int size;
~Animal {
    cout << "I'm inside Destructor";
}
};
```

```
int main() {
    Animal a;
    a.age = 5;
Animal *b = new Animal();
b->age = 12;
delete b;
}
```

static i.e destructor called automatically