Date: / /
Page:

Day - 100

OOPs - 2

* **Constructor:**

=> It is a special fun. that is invoked automatically at the time of object creation.

=> Name of the constructor should be same as class name.

=> It doesn't have any return type.

=> It is used to initialise the value.

Ex:

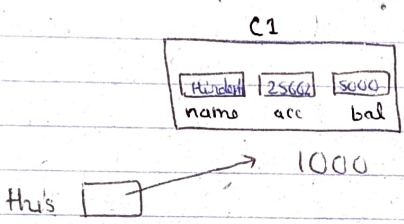
=>
```
Class Customer{
    String name;
    int acc_no;
    int balance;
    Customer(){                          → Default constructor
        cout << " constructor is called";
    }
    Customer( string a, int b, int c){   → Parameterised constructor
        name = a;
        acc_no = b;
        balance = c;
    }
    Customer (string name, int acc_no, int balance){
        name = name;
        acc_no = acc_no;
        balance = balance;
    }
```

This will not work because compiler will select the nearest name, acc_no & balance.

```
int main(){
    Customer C1("Hirdesh", 25662, 50000);
    Customer C2(C1);
}
```

→ Copy Constructor

⇒ To solve above problem, we use 'this' keyword.



⇒ 'this' will work as a pointer that points to the object.

⇒ So,

```
Customer (string name, int acc_no, int balance){
    this → name = name;
    this → acc_no = acc_no;
    this → balance = balance;
}
```

⇒ Inline Constructor

```
inline Customer (string a, int b, int c){: name(a),
    acc_no(b), balance(c){ }
```

⇒ Copy Constructor

```
Customer (Customer B){
    name = B.name;
    acc_no = B.acc_no;
    balance = B.balance;
}
```

(Instead of B, use &B) → This will not work because we have created our own copy constru. but this is also not correct.

⇒ But if we take reference of the customer that is coming to B then our problem will be solved.

Date: / /
Page:

* **Destructor:**

⇒ It is an instance member function that is invoked automatically whenever an object is going to be destroyed.

⇒ It is the last function that to be called before an object destroyed.

Ex:

```
Class Customer{
    string name;
    int * balance;

    Customer(string name, int bal){
        this → name = name;
        balance = new int;
        *balance = bal;
    }
    ~Customer(){          → Destructor
        cout<< Destructor called <<endl;
    }
```

⇒ Destructor releases the heap memory not destroy the object.

⇒ Like, it will releases the memory taken by balance in the above example.

⇒ Destructor called in reverse order.