OOPS (Object Oriented Programming)

Lecture - 70

OOPS

- It is an approach or a programming pattern where the programs are structured around object rather than function and logic.

- Earlier, if I have 10 student and we want to store the information of student like name, reg-no, and branch, then we have to make an array for which we want to store.

```
String name [10];
int reg-no [10];
String Branch [10];
```

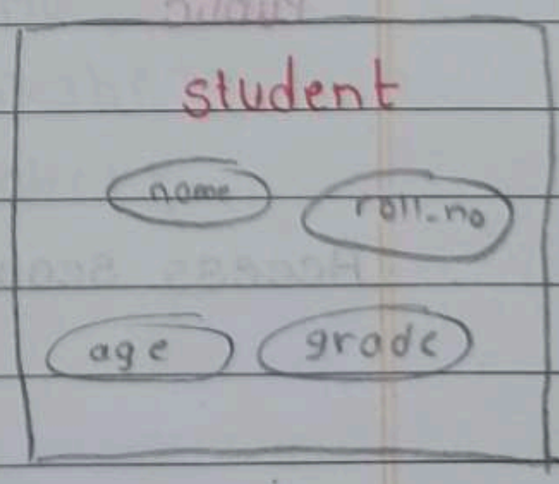
- We have to make 3 array for storing. Accessing the data of 1 student can be difficult that we have to access all the array.

- If we want to add one more data then we have to add a different array.

⇒ Can we make it easier to store the data?

Using Class & Object

```
class Student {
    string name;
    int age, roll-no;
    String grade;
};  → important
```



- Class - student is user defined datatype and whenever we need to access : name, age, roll-no and grade we will directly write : Student S1; S1 is object associated

with class student, Now to access student's data we use it object s1.

```
S1.name = "Sakshi";
S1.age = 20;
S1.roll-no = 2205111093;
S1.grade = "A+"
```

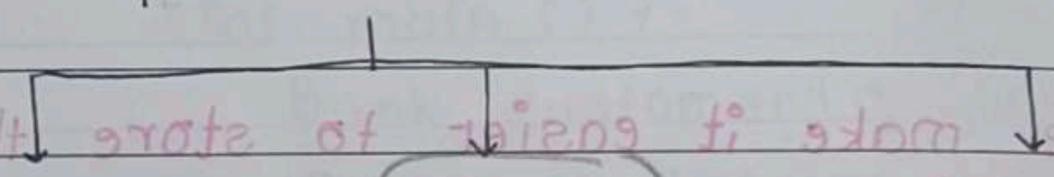
} Modifying values using S1 object

- By default class is private so, while printing st.roll-no it gives error. We have to make the class a public

```
class Student {
    public:
        String name;
        int age;
        int roll-no;
        String grade;
```

Now the code works

Access Specifier / Modifier



by default

Access scope	External Code main	Within class	Derived class
Public	✓	✓	✓
Private	×	✓	×
Protected	×	✓	✓

PAGE NO.
DATE:

Class → Behaviour : click, setting
Class → State / Property : Memory, Color, Company

⊙ We can also make function inside class
↳ to show behaviour

Example: class Bank {
public:
int balance;
int name;
void Balance-check () {
cout << balance << "";
};
void withdraw () {
balance -= 100;
cout << "Rs 100 withdrawn";
cout << "New balance" << balance;
}};

int main () {
Bank customer1;
Customer1.balance = 500;
Customer2.Name = "sakshi";
customer.Balance-check();
customer.withdraw();
} return 0;
}

In this program there is a problem: User can self modify the balance, if user have access then if is not safe. We can change the data by self

PAGE NO.
DATE:

So, what we do :
Make balance as private :
And make public to those which is required :

Ex : class Bank {
int Balance, } → private
string name ;
public :
void setValue (int amount, string names) {
Balance = amount ;
name = names ;
}
void checkBalance() {
cout << Balance ;
}
void withdraw() {
Balance -= 100 ;
cout << " Rs 100 withdrawn" ;
} cout << Balance ;

int main() {
Bank Customer1 ;
Customer1.setValue (10000, "Paridhi") ;
Customer1.check-Balance() ;
Customer1.Withdrawn() ;
}

⊙ This can also create problem. Because user or anyone can update the Balance easily

⊙ So, we have to make this like, whenever we want to update balance, we cannot update it directly

We can update it with the permission of bank employee and they can verify it - by using some password or pin

OR We can use get and set function

- Functions inside Class

Getters

```
void getage() {
    cout << age;
}
```

```
int getroll-no() {
    return roll-no;
}
```

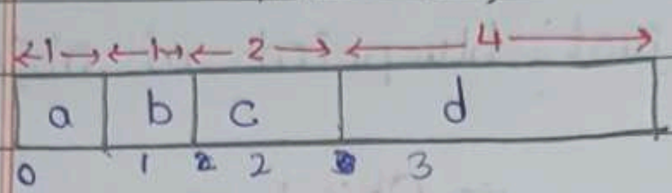
↳ Get are use to display, get access or say return values inside class to main

↳ However we can directly print these values, but with help of these we can easily apply conditions too.

↳ Similarly, there exist set function which allow us to set or say declare values to variables inside class.
As set function is written in public but variables are declared in private So how can we access set function but not in main function?
⇒ because we declare set function in same class

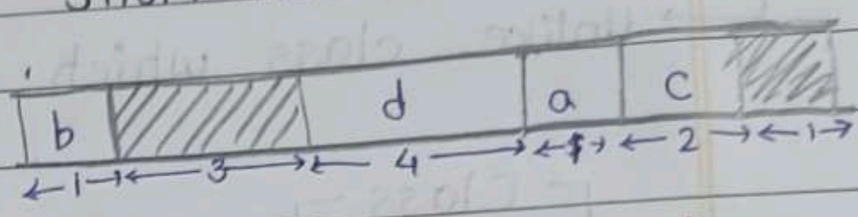
→ Conditions like if (s.size() == 0) cout << "Invalid Name" or giving information only on getting a pin are few example

Ex: (1) char a;
char b;
short-int c;
int d;



Ans: 8 byte

(2) char b;
int d;
char a;
short-int c;



Ans: 12 byte

→ One thing which we learnt from padding is agar size of variable equal yaa kam hai compared to remaining size of 4 bytes toh hum padding nhi lagayega Ex: (1)

→ Aur agar size badi ho toh padding lag jati hai and agar akhri mein size bach jata hai from yaane next 4 bytes mein toh padding ~~increase~~ add multiple hoti hai Ex: (2)
of 4.

→ So to save memory: we use Greedy Algorithm Which states that first allocate variable having datatype of larger size & later keep decreasing

int d;
short-int c;
char a;
char b;

} This is correct order of allocation

PAGE NO.
DATE:

Static VS Dynamic Memory Allocation

• Dynamic Memory Allocation of variable :

```
int *p = new int
*p = 10;
```

Similarly :

Dynamic Memory Allocation of CSS :

```
Student *s = new Student;
(*s) : name = "Rohit";
```

OR

```
S→ name = "Rohit";
```

PAGE NO.
DATE:

Lecture : 72

Constructor

- It is special function that is invoked automatically at the time of object creation
- Name of constructor should be same as class name
- They doesn't have any return ~~true~~ type
- It is used to initialise the value

Ex: class Customer {
    string name;
    int account-number;

This is default constructor which is automatically called once class was called

```
Customer () {
    cout << "Constructor is called";
}
```

This is paramedrized ~~customer~~ constructor which sets value to variables

```
Customer (string a, int b, int c) {
    name = a;
    account-no = b;
    balance = c;
}
```

Note !!

- By default a constructor is always present in class but it is not visible to us its code is written in compiler

- We can create constructor too... Agar humne khudse constructor create kiya hai tab compiler koi bhi constructor create nahi karega

- We can create multiple constructor in single class but there should be difference like maybe in no. of parameters or datatype of parameters - ~~Co~~ Constructor (Functions) Overloading

- Constructor is used to initialise the value.... so as to save default values or null values getting place in our object which has no sense or not needed

Why 'this' keyword

```
class Customer {
    String name;  yaa ismein
    int account-number;
    int balance;

    Customer() {
      cout << "Constructor is called";
    }
    ismein
    Customer(String name, int accoun-number, int balance)
    {
(this → name) = name ;
(this → account-number) = account-number;
(this → balance) = balance;
};
```

Ab yaha confusion honga na ki konse name variable mein dalna hai and acc to compiler jab variable pass mein honga usmein jayega

And because of which woh galat variable mein jaa raha and to remove this confusion - we will use "this" which will point to actual variable

- In short,
  The nearest value is allocated to variable if multiple declarations are done for same variable in class

- Constructors are differentiated from each other by no. of arguments they are asking for

- So if in main, we call a constructor & give 5 argument as input, it will check in class from top to bottom for such constructor

- Also note that, Constructor in same class with different no. of arguments are called Overloading Constructor

Inline Constructor

```
inline Customer (String a, int b, inet c): name(a),
 account-number(b), balance(c) {
 }
```

⊙ If constructor with same no. of argument are made, it show error

Copy Constructor

If we want to copy properties of one in another one, we simply need to write -

```
Customer A2 (&A1) {
    name = B.name;
    account-number = B.account-number;
    balance = B.balance;
}
```

Reference of class A1 to copy details easily

Another way to copy constructor is

```
A2 = A1
cout << A2.balance;
```

Another way to initialize values of constructor is -

```
Customer A1 ("Rohit", 1000);
```

Note !!

- Constructor is also used for initialising resources like opening file, etc.
- Copy Constructor is also present in our class by default but once we create it then compiler do not create it again. Same goes for default or parametrized constructor
- Agar hum copy constructor ka use kare and usse declare yaa code nahi likha toh bhi copy constructor

Lecture - 72

Static Data Members

- They are attribute of classes or class member

- Declared using static keyword

- Only one copy of that member is created for the entire class & is shared by all the objects

- It is initialized before any object of this class is created

In Total, Agar hum chahte hai ki har object 1 data variable ko share kare Jaise ki : Bank - Total - Balance, Nation - Population, etc. then we create static data member

* This is declared inside class static int total - customer and initialized outside class & before main int customer :: total - customer = 0

- let's see following code which is using static to fix the total-balance for all customers (objects)

aram se kaam karega due to its already presence in our codebase

- Value can be added from one object to other object using assignment operator also

Destructor

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

- It is last function that is going to be called before an object is destroyed

```
~Customer () {
    delete balance;
}
```

- Both constructor and Destructor are written under public access modifier

- Default constructor, copy constructor & Destructor are already made & hidden but once created manually need to take care of, Also they need to be created when any constructor is written in class

- Dynamically Allocated storage are released by destructor if multiple objects are created.

Copy Constructor

If we want to copy properties of one in another one, we simply need to write -

```
Customer A2 (&A1) {
    name = B.name;
    account-number = B.account-number;
    balance = B.balance;
}
```

Reference of class A1 to copy details easily

Another way to copy constructor is

```
A2 = A1
cout << A2.balance;
```

Another way to initialize values of constructor is -

```
Customer A1 ("Rohit", 1000);
```

Note !!

- Constructor is also used for initialising resources like opening file, etc.
- Copy Constructor is also present in our class by default but once we create it then compiler do not create it again. Same goes for default or parametrized constructor
- Agar hum copy constructor ka use kare and usse declare yaa code nahi likha toh bhi copy constructor

```
static int total-balance;

void deposit (int amount) {
    if (amount > 0) {
        balance += amount;
        total-balance += amount;
    }
}

void withdraw (int amount) {
    if (amount <= balance && amount > 0) {
        balance -= amount;
        total-balance -= amount;
    }
}

int bank :: total-balance = 0;
```

Suppose, deposit (2500)
withdraw (1500)
deposit (4000)
} were called in main

Then At last total-memory will reflect ⇒ 5000

Homework

What is const keyword?

Static keyword use a variable value for all functions const do the same but unlike static where values can be changed, value here remain constant

Encapsulation

- Wrapping up of data and information in a single unit, while controlling access to them

- This is done without giving direct access to user for variables. This is done to avoid changing in database by user by mistake and called Data Hiding

Ex: Class Customer {

~~public:~~ Private: To prevent by mistake modification of data by user using deposit(-ve) instead we make a constructor public & use it in function to insert (correct) data by applying conditions

string name;
int balance;
int age;

Public:
Customer(string a, int b, int c) {
name = a;
balance = b;
age = c;
}

* Doing this will show error on incorrect inputs

void deposit(int amount) {
if (amount > 0)
balance += amount;
else
cout << "Invalid amount";

Bursting Myth !!
Hum data encapsulate or hide hacker se bachne ke liye nhi krte balki by mistake agar hum galat value daal de toh usse bachne ke liye karte & function helps us to achieve this by if-else conditi

jisse hum wrong valuess feed na kr sake
Ex : Negative valve

Abstraction

→ Displaying only essential informations & hiding the details

Ex: Class Customer {
    string name;
    int balance;
    Public :
    Customer(string a, int b) {
        name = a;
        balance = b;
    }
    void deposit (int amount) {
        if (amount > 0)
            balance += amount;
    }

* Suppose in bank while depositing amount the information to display is

Amount Deposited
Total balance
Time of Transaction
ID Reference No

So we will show this only and not other things which are personal, crucial or avoidable

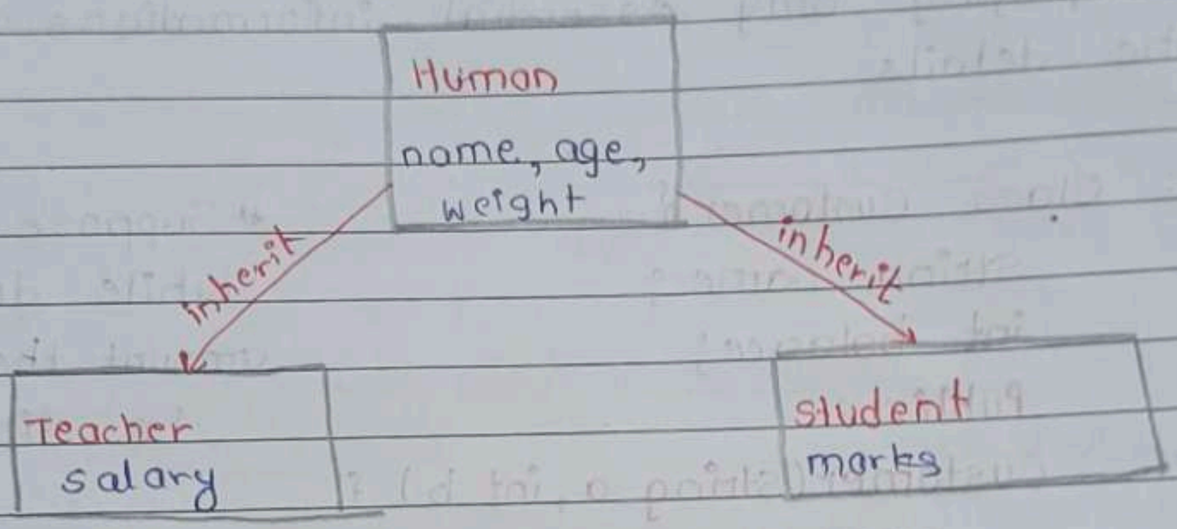
Other Ex:
POW(2,4) : function will only show final result & not steps done to achieve it

sort(arr.begin(), arr.end()) : will also show sorted array and not procedure or line of codes written in function

Lecture - 73

Inheritance

→ The capability of a class to derive property & characterstic from another class.



→ So A class can inherit properties from other class but also have their own unique properties & this is automatic so can't be controlled, selective inheritance can't be done.

Types of Inherited Class

If Base class is	Private	Public	protected
Private	Hidden Nt accessible	Hidden Nt accessible	Hidden Nt accessible
Public	Private	Public	Protected
Protected	Private	Protected	Protected

* see another table

This is due to the Power order of these access modifiers

Private > Protected > Public

Ex:

```
Class student {
        public :
            int roll-no , fees ;
        }
Class Human : Protected students {
        Protected :
            fees = 11000 ;
        }
```

// Super Method enables derived class to inherit all properties of base class

Protected will dominate & error will be shown

→ Unlike Private, protected can be inherited to Derived class ~~Really?~~

and also helps/saves Encapsulation

Code:

```
class student {
  public :
    student ( string name, int age, int weight, int roll-no,
              int fees ) {
      this → name = name ;
      this → age = age ;
      this → weight = weight ;
    }
```

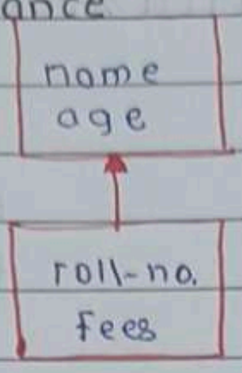
Q. What is object base? obj oriented? Pure object oriented? init ?n ?

this → refers to current instance of a class & allows you to access its properties & methods within its own scope, also is a pointer & known as self in some language

Lecture -74

Types of Inheritance

1) Single Inheritance



We can see clearly, student class need name & age so it undergo single level inheritance & added roll-no. and fees as extra's

Code :

```
Class Human {
  protected :
    String name;
    int age;
  Public :
    void work() {
      cout << "I am working";
    }
};
Class Student : Public Human {
  int roll-no., fees;
  Public :
  student (String name, int age, int roll-no, int fees) {
    this → name = name;
    this → age = age;
    this → roll-no = roll-no;
  } this → fees = fees;

int main () {
  student A1;
  ....
}
```

→ Here is sample code for the same, where student is child's class

→ If both parent class & child class both have constructor & deconstructor then : Constructor will be implemented firstly of parent class but Deconstructor will be implemented firstly of child class

* Function Calling

In function call however, if the function with sufficient argument exist in child class. It will call this only otherwise search in parent class

PAGE NO.
DATE:

2) Multilevel Inheritance


```
Class Person {
  Protected:
    string name;
  Public:
    void introduce() {
    } cout << "Hello my name is: " << name << endl;
};
Class Employee : Public Person {
  Protected:
    int salary;
  Public:
    void monthly-salary() {
      cout << "My monthly salary : " << salary << endl;
  };}

Class Manager : Public Employee {
  Public:
    string department;
    Manager(string name) {
      this → name = name;
    }
    void work() {
      cout << " I am leading department " <<
                                    department;
    }
}
```

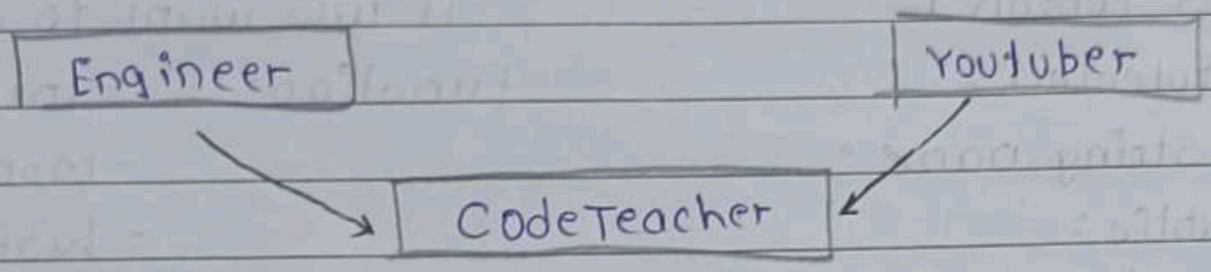
↳ If we want to access functions : - Introduce
- Monthly-sal
- Work
by help of manager class -
- we can do it as its parent are public (& it itself too)

PAGE NO.
DATE:

3) Multiple Inheritance

→ In this, one class inherits multiple classes is properties in it. This is done by -

Class CodeTeacher : public Engineer, public Youtuber



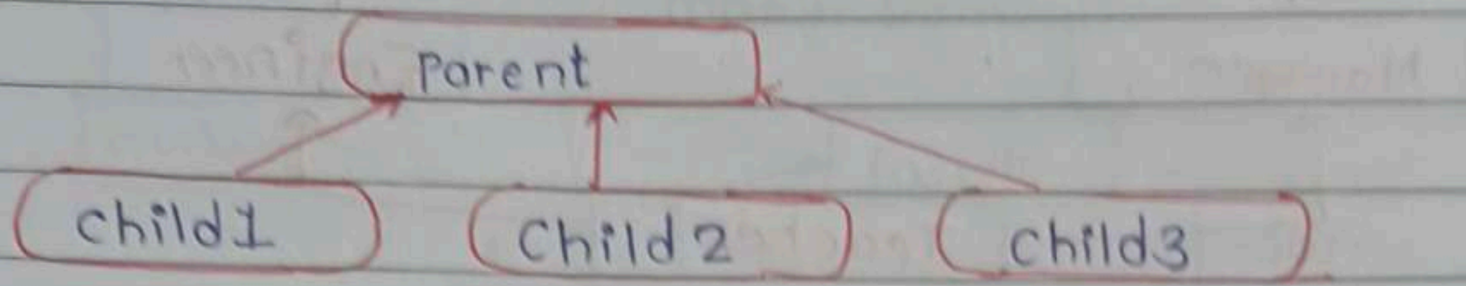
→ Supposing that Engineer and Youtuber are its independent parent classes similar to previous 2, function belonging to any parent or child class itself, can be called via child class &

→ If we create constructor & deconstructor of both class (parent) then the one first inherited, here Engineer will be called in constructor while in Destructor it will reverse

(code in VS)

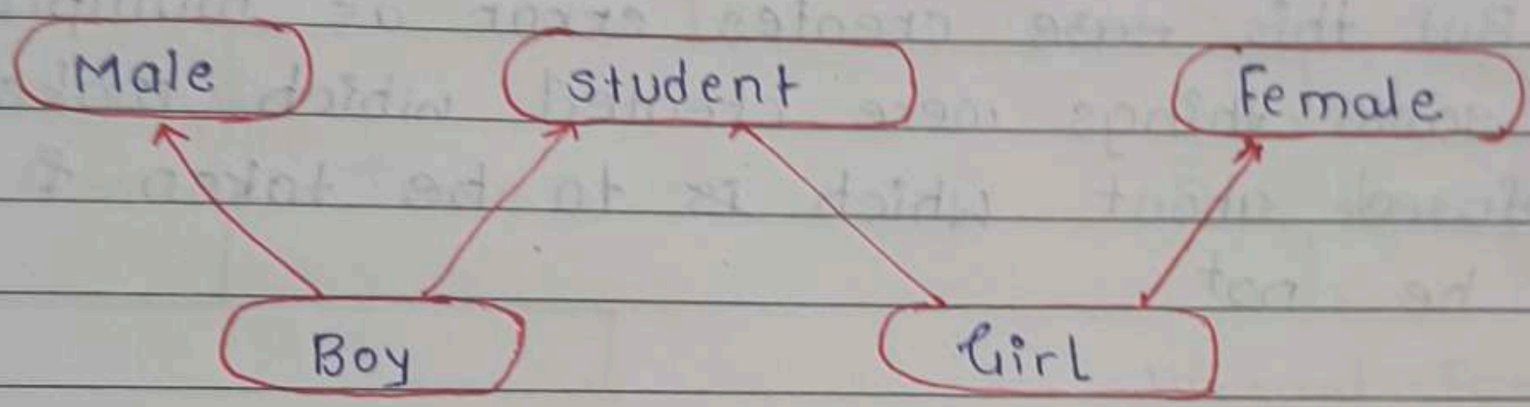
4) Hierarchial Inheritance

Unlike previous one here properties of one are inherited into multiple child class



5) Hybrid Inheritance

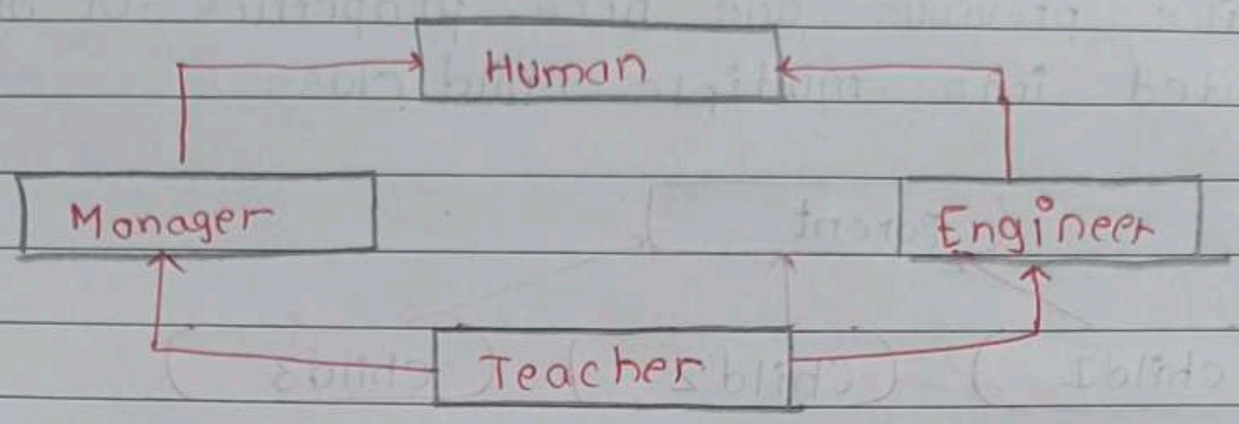
If more than 1 type of inheritance is present that it become Hybrid



Ex: Suppose Parent is Parent class of child 1,2,3,4 & Grand child is child class which inherit properties from child 1,2,3,4

VS code for code

6) Multipath Inheritance



→ ~~Suppose~~ * Same things come back to parent class along with unique characterics of other classes

→ But this ~~case~~ creates error as multiple copies of same things were created which make system confused about which ~~is~~ to be taken & which to be not.

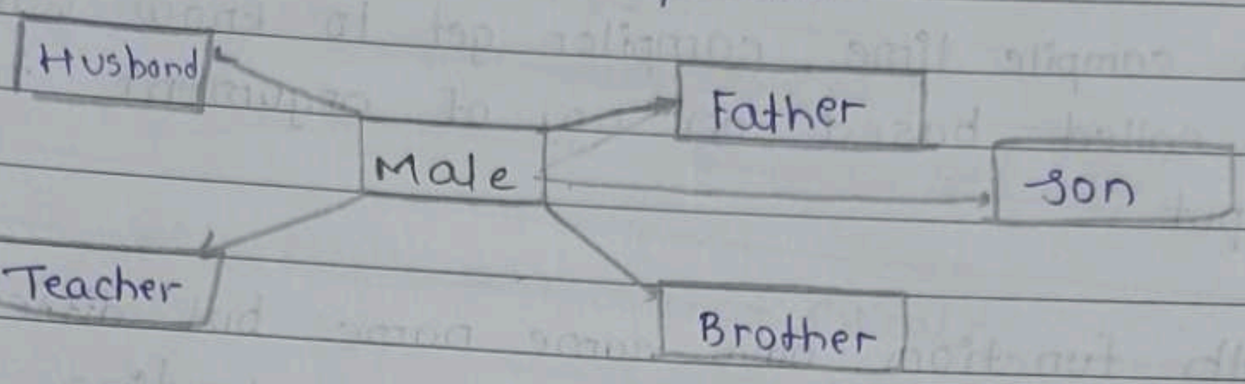
→ So here Virtual keyword plays important role by telling that no multiple copies are required as it is same thing.

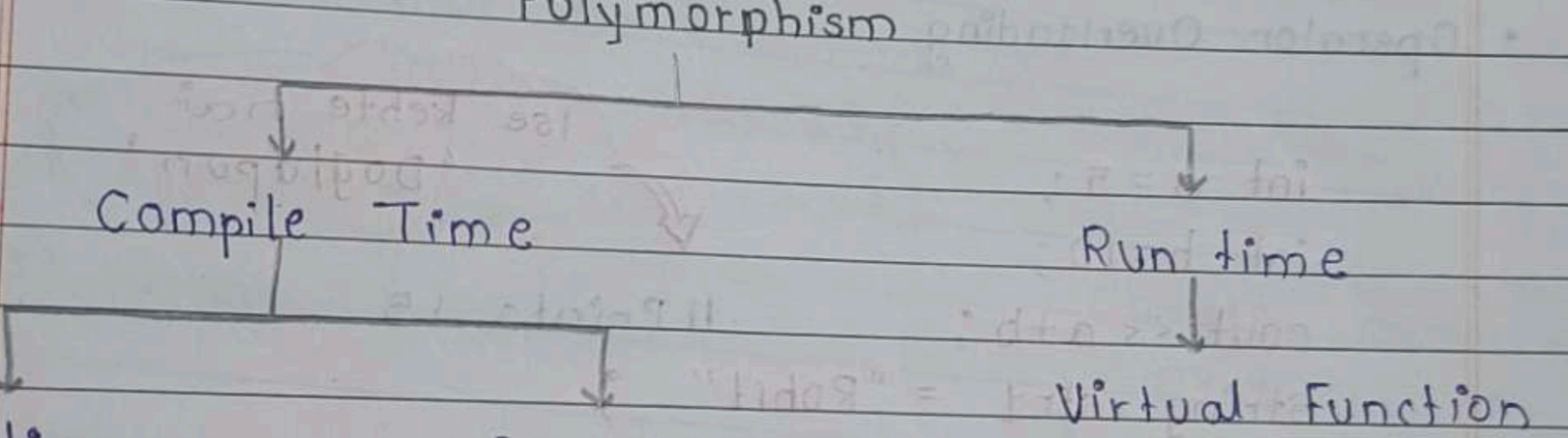
Lecture - 75

Polymorphism

Poly : many morphism = form

" many-form "





Compile Time

This finds out syntax error & error like 2 constructor in same class with same no. of arguments

- Function Overloading

```
class Area {
public:
int calculateArea (int r) {
    return 3.14 * r * r;
}
```

```
p → speak();
```

```
int calcubtearea (int l, int b) {
    return l*b;
}
```

* On compile time, compiler get to know which function is called based on no. of argument taken as input

* Both function have same name but different no. of argument leading to function overloading

- Operator Overloading

```
int a = 5;
int b = 10;
cout << a+b;          // Prints 15
String str1 = "Rohit";
String str2 = "Negi";
cout << str1 + str2;  // Prints RohitNegi
```

Ise kehte hai 'Doglapon'

- This is operator overloading as same sign '+' performed differently as per input data type

do overcome confusion we can use function

```
Class Complex {
    int real, img;
    Public:
    Complex (int real, int img) {
        this → real = real;
        this → img = img;
    };
    void display () {
        cout << real << "+i" << img;
    }
}
```

```
Complex Operator + (complex & c) {
    Complex ans ;
    complex Operator+ (complex c) {
    ans                          ——→ Default construction
    }
    ans.real = ans real + c.real ;
    ans.img = img + c.img ;
    return ans ;
}
int main () {
    complex c1(3,2) ;
    Complex c2 (4,6) ;
    c1 + c2 ;
}
```

Although private but due to some class ← we can access it

We can't do it this way

Complex c3 = c1+c2
c3.display() ;

Virtual Function

```
class Animal {
  Public :
virtual void speak() {
    } { cout << "Hu Hu" ;
}
Class Dog : Public Animal {
  Public :
  void speak () {
} } cout << "Bark" ;
int main () {
  Animal *p ;
  p = new Dog () ;
} p → speak () ;
```

• Now in main function when we dynamically allocate P variable which is of animal class type later on, we assign Dog class to it Then on calling speak function it should print Bark, But on compile time it see ki animal class kha hai & print Hu Hu as it see speak function in Animal class

→ Virtual keyword : tell computer not to decide this in compile time but decide it in run time, then on doing this - we get bark as output

→ However, with this we can only access those functions which are present in both parent & derived class

→ This will help when there are many derived class which need to be executed & linked to a parent class

↳ So if animal is parent class & dog, cat, Buffalo & Lion are derived class then,

```
for (int i=0 ; i<animal.size ; i++) {
    p → animal[i];
    p → speak;
}
```

will write speak function for ~~even~~ each of the given animals

Pure Virtual Function

```
virtual void speak() = 0 ;
```

- This is abstract class
- Overridden of function from derived class is necessary else error will be shown.
- We do this when we want a parent class for pointer but don't ~~worry~~ want anything to be called/print from parent class, like we want Cat, Dog, Lion to speak, but not from animal.

- use of const is to keep msg fixed

PAGE NO.
DATE:

Homework

- Explain other cases where we neglet Primary class
- We can overload almost every operator except a few : one -

① Conditional Operator ?:
② Size of operator sizeof
③ Scope resolution Operator ::
④ Class Member Selector .
⑤ Member Pointer selector .*
⑥ Object type typeid

Lecture - 76

Exception Handling

An exception is an unexpected problem that arises during the execution of a program & our program terminates suddenly with some errors / issues, handling these runtime error is Exception Handling

- **TRY** : It represents a block of code that may throw a exception placed inside the try block

- **Catch** : It represents a block of code that is executed when a particular exception is thrown from the try block.

- **Throw** : An exception in c++ can be thrown using the throw keyword

↳ If we want to ~~know~~ write a code for deposit and withdraw then we have to apply condition of -

```
if (amount > 0 && Total-balance > amount)
    withdraw amount;
```

Now this ensure that exception of low balance is handled but, as this is exception we must handle & highlight it separately

Throw prints the error message ← **throw** "amount should be greater than 0";

← **throw** "Your balance is low"

TRY Block : This block actually just execute the cases which gives exceptions by putting these at 1 place.

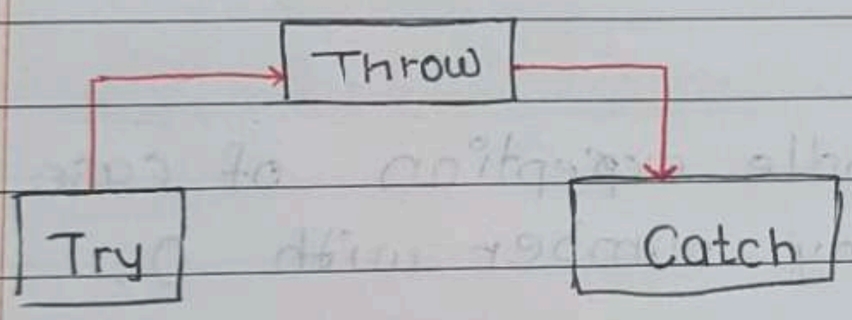
```
try {
    c1.deposit(100);
    c1.withdraw(6000);
}
```

Catch Block : Catch exist after try, this contain the throw ~~elemen~~ statement with additional add on

```
Catch(const char *e){
    cout << "Exception Occurred :" <<e<<endl;
}
```

It prints,

Error Occurred : Balance is low



→ If throw statement exist all below written statements will not execute

```
} runtime-error (const string &msg) :exception(msg);
```

↳ The use of const is to keep msg fixed

PAGE NO.
DATE:

Example for Try, Catch and Throw

1] WAP to divide 2 number

```
int main() {
    int a, b;
    cin >> a >> b;
    try {
    if (b==0)
        throw "Divide by 0 is not possible";
    int c = a/b
        cout << c <<endl;
}
catch (const char *e) {
} } cout << "Exception Occurred" <<e <<endl;
```

→ And by this we handle exception of case when we were dividing any number with 0.

2] Allocating a large memory

```
int main() {
try {
    int *p = new int[10000000000];
    cout << "Memory allocation is successfull \n");
    delete []p;
}
catch (const exception &e) {
    cout << "Exception occurred due to line 9:" <<e.what()
                                                <<endl;
} }
```

↳ The use of const is to keep [illegible]

TRY Block : This block actually just execute the cases which gives exceptions by putting these at 1 place.

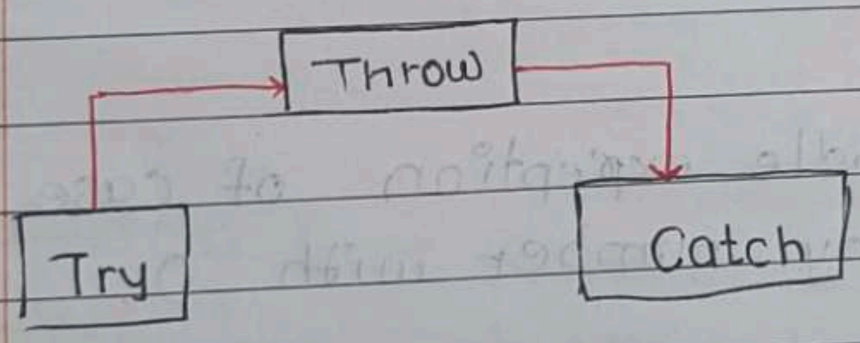
```
try {
    c1.deposit(100);
    c1.withdraw(6000);
}
```

Catch Block : Catch exist after try, this contain the throw ~~elemen~~ statement with additional add on

```
Catch(const char *e){
    cout << "Exception Occurred :" <<e<<endl;
}
```

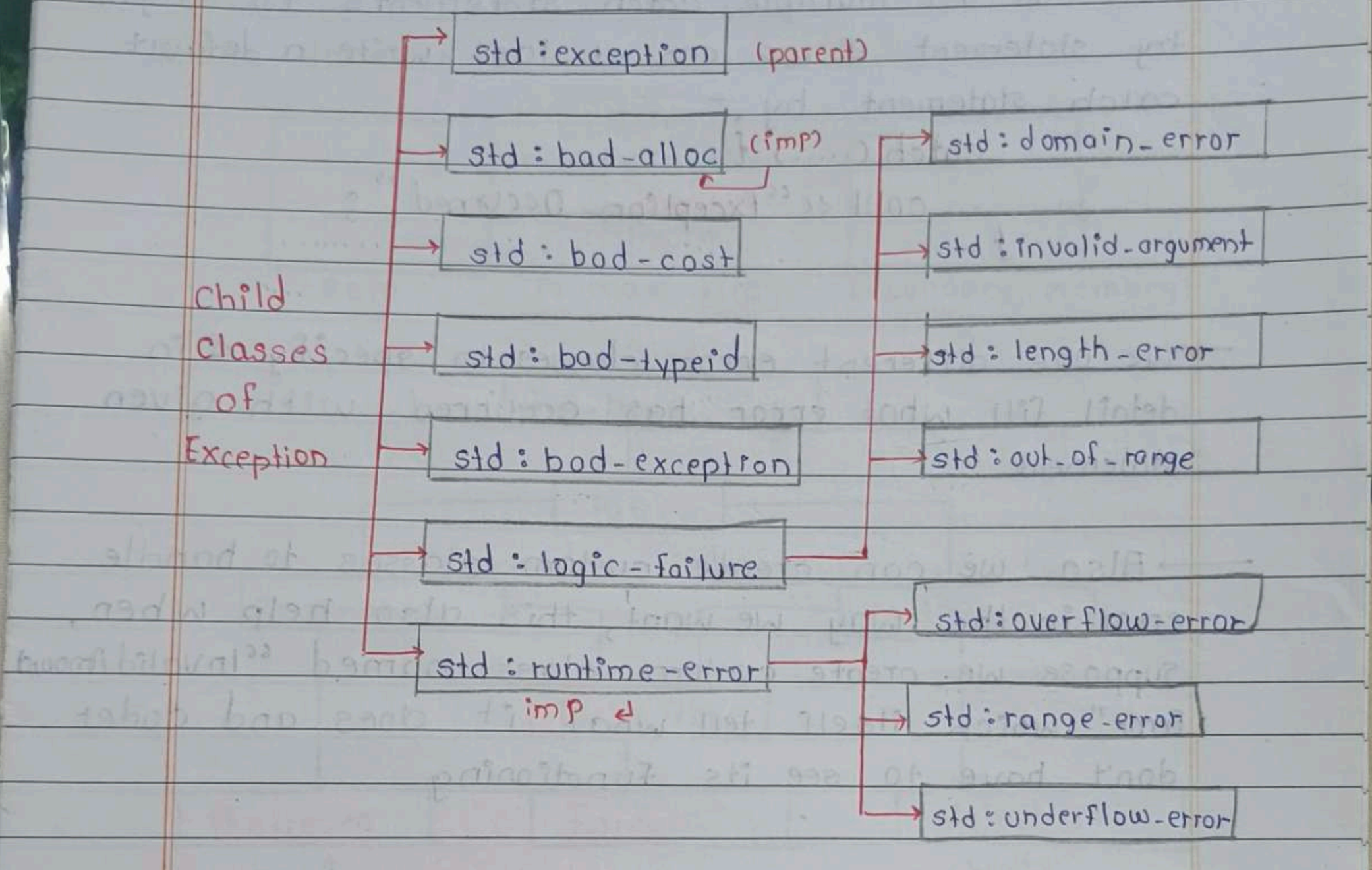
It prints,

Error Occurred : Balance is low



→ If throw statement exist all below written statements will not execute

```
}
(
}
```



Create Custom Error Class

```
class Exception {
  Protected :
    String msg;
  Public :
  exception (String msg) {
     this → msg = msg;
  }
  String what() {
     return msg;
  } }
Class runtime-error : public exception {
  Public :
} runtime-error (const string &mgg) : exception (msg);
```

↳ Now by this code, in example ② under catch when we call exception this will work if we create a what function too

↳ Also under runtime-error class we can add more functionality to it

single level inheritance

↳ The use of const is to keep msg fixed

PAGE NO.
DATE:

→ We can use multiple catch statements for single try statement & we can at last write a default catch statement by,-

```
catch (....) {
    cout << "Exception Occurred";
}
```

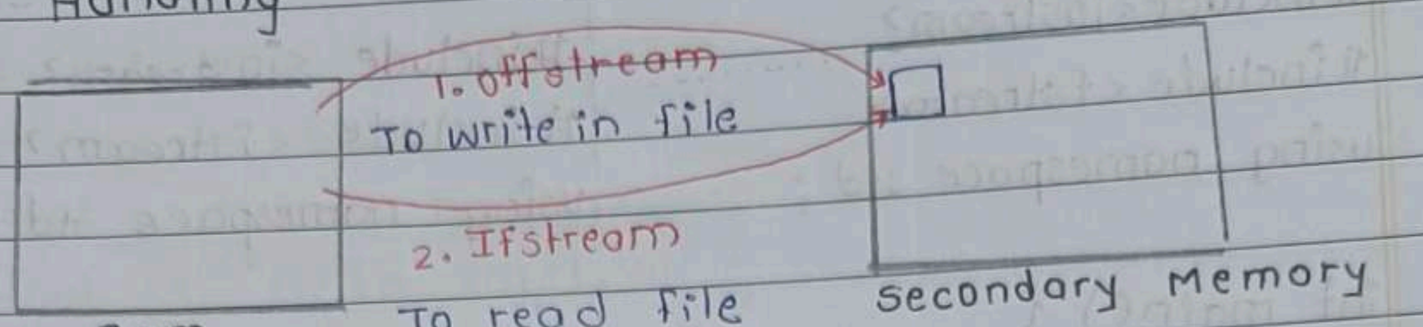
→ We use different error classes to specify & in detail fill what error has occurred with given statement

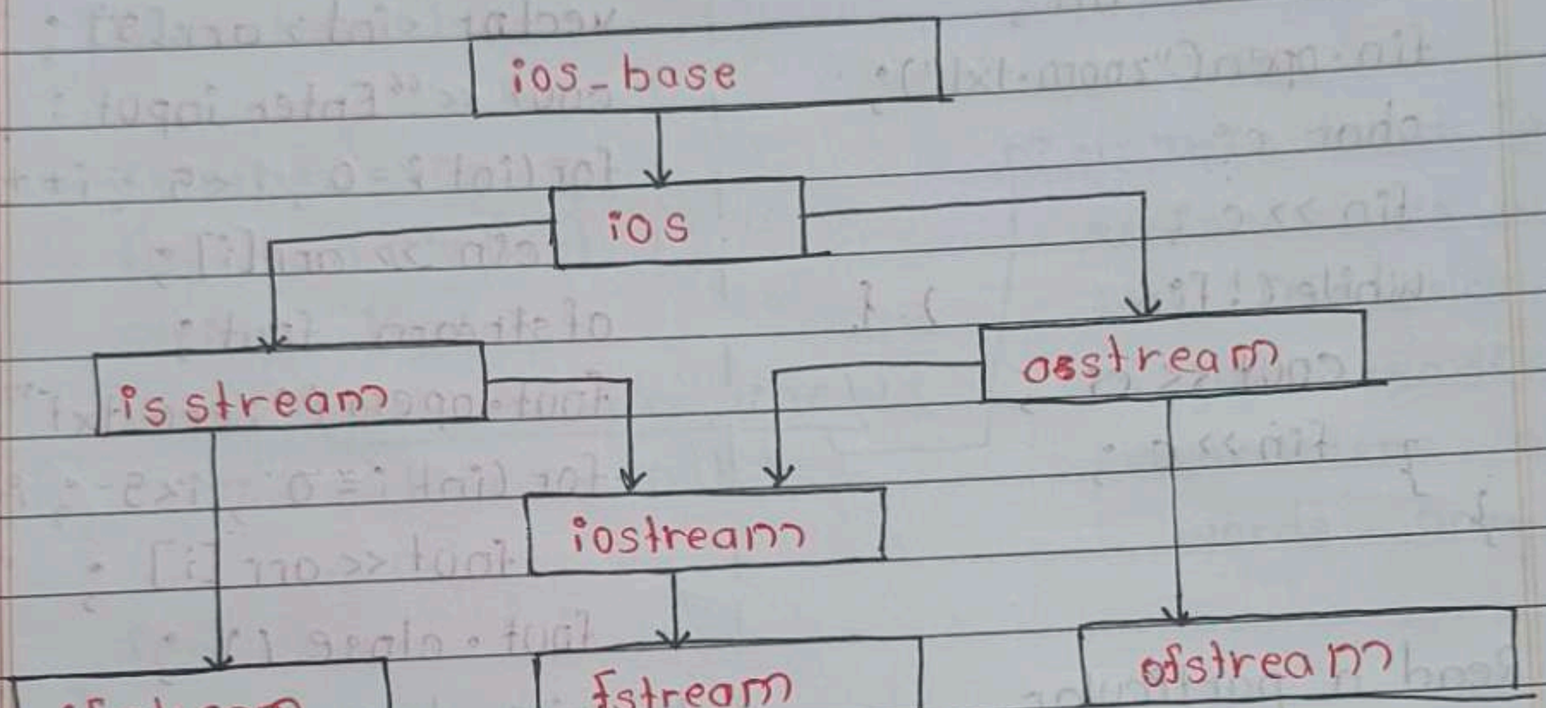
→ Also we can create custom classes to handle error in the way we want, this also help when, suppose we create custom class named "Invalid Amount Error" which itself tell what it does and coder don't have to see its functioning

PAGE NO.
DATE:

Lecture - 77

File Handling





- Code - 1

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
ofstream fout;
fout.open("zoom.txt");
fout << "Hello India";
fout.close();
}
```

↳ ofstream fout created object named fout which helps to write in file

↳ fout.open helps to open a mentioned file or create a file if not present

↳ fout << "Hello India" write text in zoom

↳ fout.close helps to close opened file and help in releasing occupying resources (in use)

Code : 2

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

int main() {
  ifstream fin;
  fin.open("zoom.txt");
  char c;
  fin >> c;
  while (!fin      ) {
     cout << c;
     fin >> c;
  }
}
```

Read a particular character in file

Code : 3

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

int main() {
  vector <int> arr[5];
  cout << "Enter input : ";
  for(int i=0; i<5 ; i++)
    cin >> arr[i];
  ofstream fout;
  fout.open("zero.txt");
  for(int i=0; i<5 ; i++)
    fout << arr[i];
  fout.close();
// Write data in vector to a
    file

  fout <<"\n Sorted data\n";
  sort(arr.begin(), arr.end());
  for(int i=0; i<5 ; i++) {
    fout << arr[i] << " ";
  };
  fout.close();
}
// sorts the input data &
  then input data into
    file
```

PAGE NO.
DATE:

Code : 4

```
int main() {
  ofstream fout;
  fout << "Hello India \n";
  fout << "Hello Rohit \n";
  fout << "Hello Bhai \n";
  fout.close();

  ifstream fin;
  fin.open("z1.txt");
  string line;
  while (getline( fin, line)) {
     cout << line << endl;
  }

  fin.close();
}
```

This Reads entire sentence as whole till file ends instead of just reading characters or words only