While making any programs, we can follow 2 approaches

1. Procedural (Steps)

2. OOPS

How many entities are involved

Trainer > Teach classes, give test , check test

Student > They will come for enquiry, can take admission

Counsellor > Handle enquiry, new featues

OOPS

Concepts of OOPS

* Class > Blueprint based on which objects of a class could be created, user defined type
* Object > An instance of a class
* Encapsulation > Hiding the details that the user do not require
* Abstraction > Showing the details that the user requires
* Polymorphism > One name different forms

1. Compile Time (Function Overloading)
2. Run Time (Virtual Functions, Method Overriding)

* Data Hiding > Data is hidden outside the class
* Inheritance > Deriving features of a class into other class

1. Single A > B
2. Multilevel A > B > C
3. Hierarchial > A > B , C
4. Multiple A, B > C its not supported through classes because of Diamond Problem

Multiple Inheritance in C++ - GeeksforGeeks

The "diamond problem" (sometimes referred to as the "Deadly Diamond of Death") is an **ambiguity that arises when two classes B and C inherit from A**, and class D inherits from both B and C. ... It is called the "diamond problem" because of the shape of the class inheritance diagram in this situation.

How do we achieve Encapsulation and Abstraction > By using Access Specifiers

Private , public

Data Type

inbuilt / user defined

int

char

float

Data type indicates 2 things

what we can store / What functions we can perform

int >

1. What we can store

we can store only numbers /

1. Functions > + - \* /

Int x;

User defined types > class structure enum

Class Student

{ int rn;

String name;

Void get() {}

Void show() {}

}

Student student = new Student();

Polymorphism

int add(int x, int y)

{

return x+y;

}

int add2(int x, int y, int z)

{

return x+y;

}

int add3(int x, int y, float z)

{

return x+y;

}

Instead of this, we can only different functions with same name (Method Overloading)

int add(int x, int y)

{

return x+y;

}

int add(int x, int y, int z)

{

return x+y;

}

int add(int x, int y, float z)

{

return x+y;

}

**To initialize objects at compile time**

1. **Constructors have same name as class name**
2. **They are like methods but they are not called explicitly. They are called automatically**

**at time of object declaration**

1. **They don’t have return type**

Constructors > How many constructors

* Static (it is used to initialize only static variables) 1 , parameter less, shud not have any access specifier
* Private
* Copy
* Default (there is 1 def cons by default)
* Parameterized

Class Student

{ int rn;

String name;

Static string batch;

public Student() {}

// Para

public Student(int rn , string name) {

this.rn = rn;

this.name = name;

}

static Student()

{

Batch=”B001”;

}

void get() {}

void show() {}

}

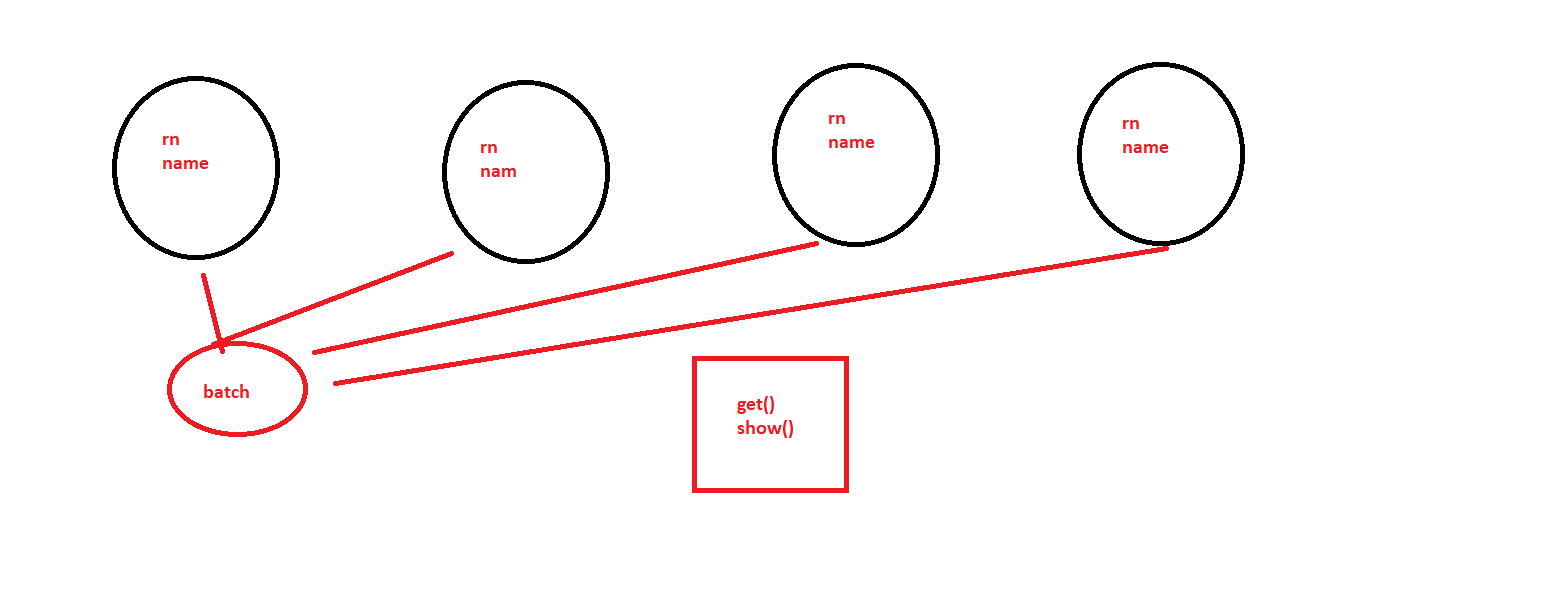
Main(){

Student s1 = new Student(); // It will call def constructor , but before that will invoke static contr, if it is there

Student s2 = new Student();

Student s3 = new Student();

Student s4 = new Student();



If within a class, we have multiple constructors

Def

Static

Para

**Which will be called first (STATIC)**

**Inheritance**

Class Student() {}

Class ParttimeStudent : Student

{

// the public members will be inherited

}

**Arrays Vs Collection**

Array > it is a structure in which all the elements share a common name & type & they are distinguished by their position

Int[] num = new int[10];

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 5 | 6 | 8 | 9 | 7 | 90 | 87 |  |  |

Limitations of Arrays

1. Static Memory Allocation, memory is allocated at compile time, memory is fixed
2. We cannot delete elements
3. Insertion / deletion are time consuming
4. Memory is wasted

Collections

A structure which allows you to store elements

1. Dynamic Memory Allocation
2. Memory to be used is decide at run time

Different collections

ArrayList

Queue

Stack

HashTable

**Using System.Collections**

ArrayList list = new ArrayList();

List.Add(1);

List.Add(2);

List.Add(3);

List.Remove(1);

// In Arraylist , insertion & deletion are done from anywhere

Stack stack = new Stack(); // LIFO

Stack.Push(10);

Stack.Push(20);

Stack.Push(30);

Stack.Pop(); // To delete element

Queue queue = new Queue(); // FIFO

Queue.Enque(1);

Queue.Enque(1);

Queue.Enque(1);

Queue.Deque(); // Delete

DIsplay elements

Foreach(var x in list)

Console.Write(x);

For(int i=1;i<list.Count;i++)

Console.Write(list[i]);

For loop is faster , it allows to retrieve & modify them

Foreach loop is slower , only for retrieval