OOPS

OOPS-1

1. Intro to OOPS

a) OOPS is a popular programming paradigm
b) One of the best way to manage the code
c) Usually used to write good quality code in almost all the sector whether it be
→ App (Android) Development
→ iOS development
→ Web development
} OOPS is the default choice everywhere

d) The concepts in OOPS are very useful to us

If we want to turn/write our communication in form of a code, then the OOPS concepts comes handy.

e) We are surrounded by Objects.
Our communication also includes objects
Table, Chair, Lamp, Pen, (even us human) are also Objects. Using OOPS, we want to replicate all our comm. in form of code

f. In Object Oriented Programming,

1. first of all, we identify what are the important objects that our system will revolve around & then we stay writing code stimulating to the problem that we are trying to solve.

eg1) If we trying to solve a problem of E-Learning System

What are the key identities that the platform revolve around?

* User
* Batches
* Courses
* Videos
* Lecture
* Problems

} Objects/Keyentities

If I say, In this batch give admission to this user, we can translate it to code

Batch1.giveAdmission(user1)

property

Object

Object

→ Give all the batches in which user1 is enrolled to chck how much discount user is elegible for

user1.getAllBatches()

→ Check how much discount is the user elegible for?

user1.isEligible();

eg: If we talk about College Management System

Key Entities	Properties
Teacher	Class, Emp id, Specialization, Qualification
Student	Year, Course, Roll No, Ph. No

g. OOP is a faster and easier way to execute code

h. OOPS basically provide a clear structure for the program

i. OOPS helps to keep the code DRY (Do not repeat yourself) and make the code easier to maintain, modify, debug

j. We basically want to replicate all our communication in the form of code using OOPS

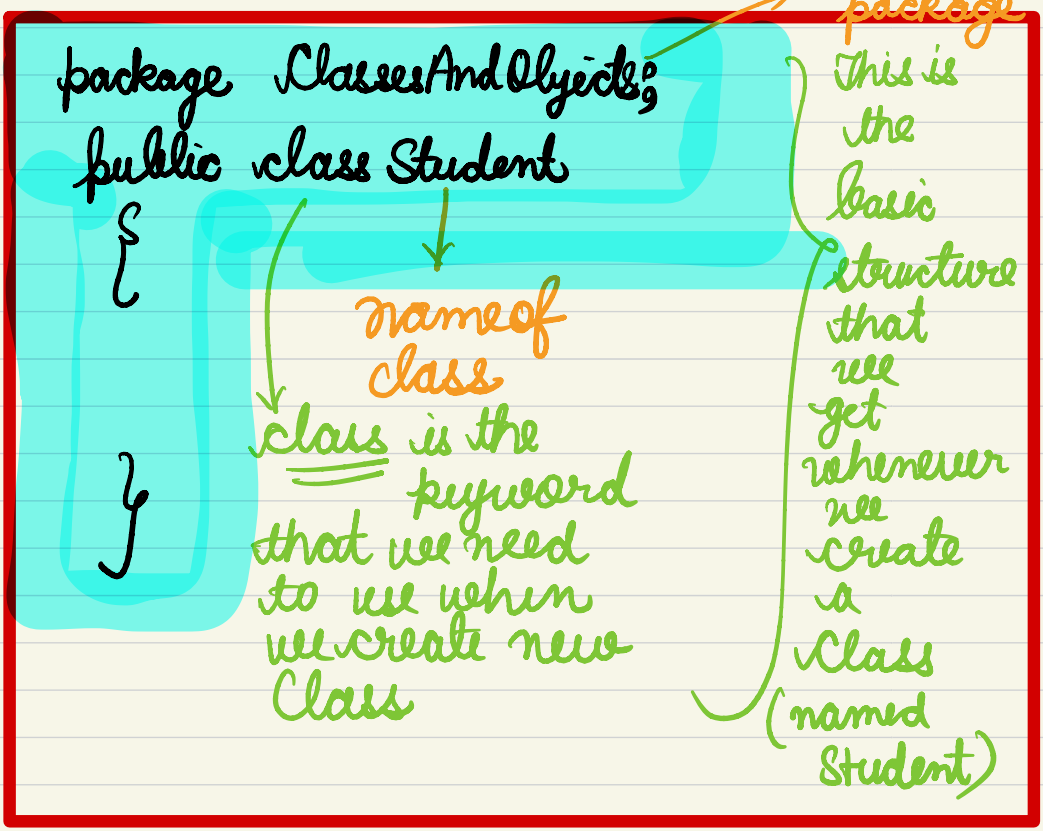
2. OBJECTS AND CLASSES

1. Entire idea of OOPS revolve around 2-major terms
   - ↳ OBJECTS
   - ↳ CLASSES

Classes basically specify the general template of how an object belonging to that class should look like and Objects individually are instance of Class.

Every object will be evolving from some class. (only then we can create them) and that Class will indicate the properties & behaviour of an object.

When we want to create a new file in Java IDE, we need to write a new class in (some project in some package)



Student Class

```
package ClassesAndObjects;
public class Student
{
    String name;
    int rollNo;
}
```

We can also use main function here to create the object of this class

If I want that Student that I create should have name, roll no → properties of Student

We generally make separate classes for objects when using for development purpose

```
package ClassesAndObjects;
import java.util.Scanner;

public class StudentUse {
    public static void main(String[] args)
    {
        Scanner is = new Scanner(System.in);
        Student student = new Student();
        student.name = 'Yugal';
        student.rollNo = '48';
    }
}
```

Object of student class

Scanner is a class

we create objects of a class using the new keyword

need not to pass any parameter here

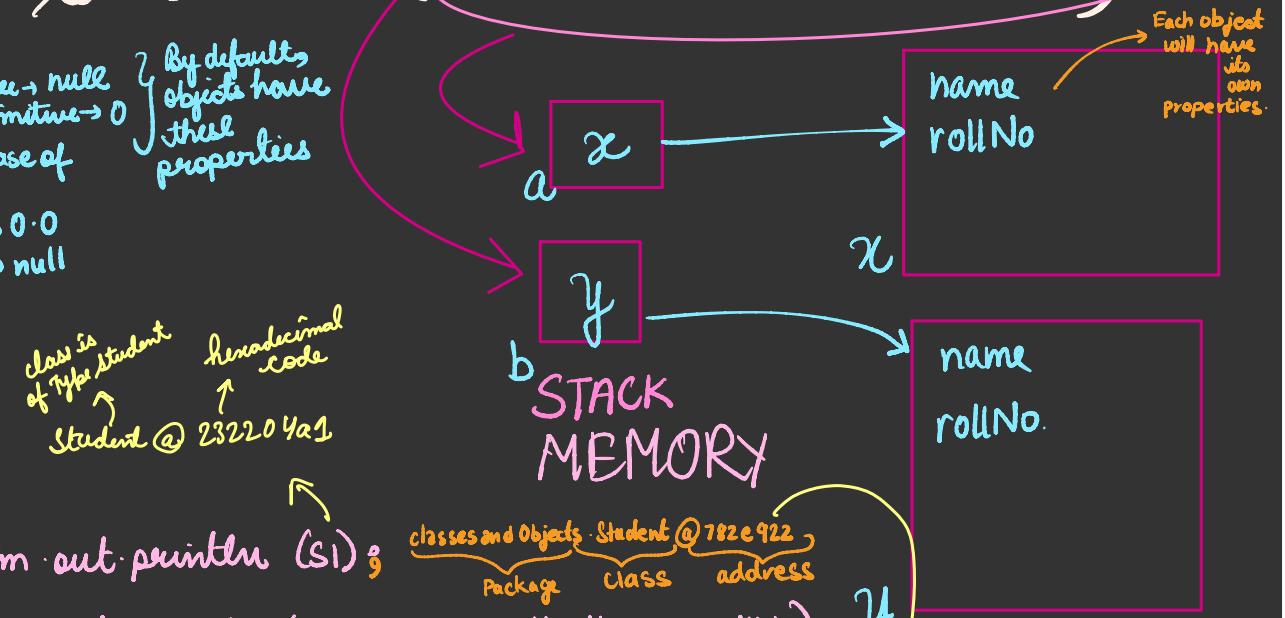
or
Student S1;
S1 = new Student();
↓
new keyword create an object.

```
publicClassesAndObject;
public class StudentUse
{
  public static void main (String[] args)
  {
    Student S1 = new Student ();
    Student S2 = new Student ();
```

package

Primitives → null
Non-primitive → 0
i.e in case of
int → 0
double → 0.0
String → null

} By default, objects have these properties



class is of type Student

hexadecimal code

Student @ 23220 4a1

```
* System.out.println (S1);
* System.out.println( S1.name +" "+ S1.RollNO);
```

classes and Objects . Student @ 782e922
Package | class | address

null 0

```
S1.name = "Tamanna";
S1.rollNo = "7";
S2.name = "Shivani";
S2.rollNo = "4";
System.out.println ( S1.name + " "+ S1.rollNo);
System.out.println(S2.name + " "+ S2.rollNo);
}
```

Binary form is not address form because its way too long

hexadecimal code bcz it can be easily converted to Binary (bcoz that is what computer need

Hexadecimal code cant have i, j, etc

Colors are also represented in #FFAA54
Hexadecimal form

{ 0123456789ABCDEF

* Defining a class in Java won't occupy any memory. It'll just be saved as a file in your Hard drive

* Program → Saved in Hard Drive

  Process → Works in RAM

* All Stack and heap memory is a part of the RAM

1. All RAM memory require a constant current to hold its state i.e as soon as you turn the power off all your RAM memory vanishes i.e RAM is volatile.

2. Stack is a specific area of RAM memory used to store temporary variables during the program execution. Each function call 'pushes' data to the stack. The data is 'popped' when the function returns.

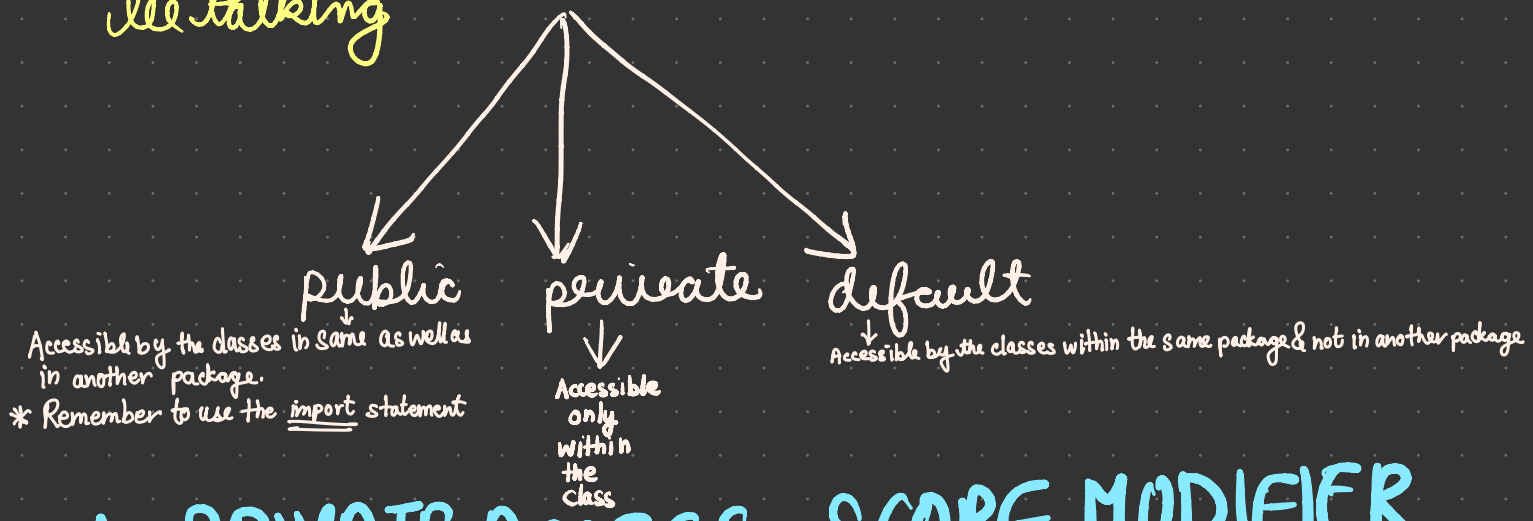
3. Because Stack too is stored in RAM, it too is volatile.

4. The RAM is the physical memory of your computer. Heap memory is the (logical) memory reserved for the Heap. So only a part of RAM is used as Heap memory and Heap memory doesn't have to be fully loaded into RAM (eg → part of it may be swapped to disc by OS.

	STACK	HEAP
Basic	Memory is allocated in a Contiguous block	Memory is allocated in any random order
Allocation & Deallocation	Automatic by compiler Instruction	Manual by the programmer
Cost	Less	More
Implementation	Easy	Hard
Access Time	Faster	Slower
Main Issue	Shortage of memory	Memory fragmentation
Locality of Reference	Excellent	Adequate
Safety	Thread safe, data stored can only be accessed by the owner	Not thread safe, data stored visible to all threads
Flexibility	Fixed Size	Resizing is possible
Data Type Structure	Linear	Hierarchical
Preffered	Static memory allocation is preferred in array	Heap memory allocation is preferred in the linked list
Size	Smaller than Heap memory	Larger than Stack memory

3. Access Modifier

There are 3 access modifiers about whom we will be talking



1. PRIVATE ACCESS SCOPE MODIFIER

Eg-1. If I have created Student class & Person B created Student Use Class and I dont want roll no to be set lesser than 0, so, we can restrict the control to within the class by writing private before that particular variable or function of the class that we dont want to be accessed

PRIVATE ACCESS/ SCOPE MODIFIER

```
package classes and Objects
public class Student{
    String name;
    private int rollno;
}
```

```
package classesAndObjects;
public class StudentUse
{ public static void main(String[]args)
  { Student S1;
    S1= new Student();
    Student S2 = new Student();
    S1.name= "Tamanna";
    //we cant access rollno property
      of student now
    S2.name = "Yugal";
    System.out.println(s1.name+" "+
                       S2.name);
  }
```

2. DEFAULT ACCESS MODIFIER

→ By default the properties of a class can be accessed by classes within the same package but not outside the package

```
package classesandObjects
public class Student{
  String name;
  int rollno;
}
```

Class1

```
package classesAndObjects;
public class StudentUse
{ public static void main(String[]args)
  { Student S1;
    S1= new Student();
    Student S2 = new Student();
    S1.name= "Tamanna";
    S2.name = "Gugal";
    System.out.println(S1.name+" "+
                       S2.name);
}
```

Class2

Package = ClassesAndObjects

Different Package

We need to import the class from another package like we import Scanner

```
package Temp;
import classesAndObject.Student;
public class StudentUse2
{ public static void main(String[]args)
  { Student S1;
    S1= new Student();
    Student S2 = new Student();
    S1.name= "Tamanna";
    S2.name = "Gugal";
    System.out.println(S1.name+" "+
                       S2.name);
}
```

We cannot directly use S1.name or S1.rollno here if we have the properties with default access

Package = Temp

3. PUBLIC ACCESS MODIFIER

```
package classesandObjects
public class Student{
  public string name;
  public int rollno;
}
```

Class1

```
package classesAndObjects;
public class StudentUse
{ public static void main(String[]args)
  { Student s1;
    s1= new Student();
    Student s2 = new Student();
    s1.name= "Tamanna";
    s1.rollno="20";
    System.out.println(s1.name+" "+
                       s1.rollno);
  }
```

Class2

Package = ClassesAndObjects

Different Package

We need to import the class from another package like we import Scanner

```
package Temp;
import classesAndObject.Student;
public class StudentUse2
{ public static void main(String[]args)
  { Student s1;
    s1= new Student();
    Student s2 = new Student();
    s1.name= "Tamanna";
    s1.rollno = 20;
    System.out.println(s1.name+" "+
                       s1.rollno);
  }
```

properties of class from another package is accessible using public access modifier

Package = Temp

	SAME CLASS	ANOTHER CLASS IN SAME PACKAGE	ANOTHER CLASS IN DIFFERENT PACKAGE
PRIVATE	✓	✗	✗
DEFAULT	✓	✓	✗
PUBLIC	✓	✓	✓

4. Getters And Setters

```
package classesAndObjects;
public class Student
{ public String name;
  private int rollno;

  public int getRollNo()
  { return rollno;
  }

public void setRollno(int num)
  { if (num<=0)
    { return;
    }
  rollno = num;
}
```

getter

setter

}

We need to set the rollno iff the rollno provided is greater than 0. otherwise let it be default i.e 0

So, that's why we don't keep the access as public but private & using getters & setter,

Using getter, Setter is a good practise

```
package classesAndObjects;
public class StudentUse
{ public static void main(String[] args)
{ Scanner scan= new Scanner(System.in);
  Student S1=new Student();
  S1.name="Tamanna";
  S1.setrollno(-100);
  System.out.println(S1.name+
                     S1.getrollno);
```

Tamanna 0

```
  S2.name="Yugal";

  S2.setrollno(50);
  System.out.println(S2.name
                     +S2.getrollno);
```

Yugal 50

}
}

5. This Keyword

```
package classesAndObjects;
public class Student
{ public String name;
  private int rollno;

  public int getRollNo()
  { return rollno;
  }

public int setRollno(int rollno)
{ if (rollno<=0)
  {return;}

  rollno = rollno ;
  }
}
```

getter

setter

the function will consider both these as the parameter given

we have changed the parameter name to rollno only

This is a logical error & not a syntax error

```
package classesAndObjects;
public class StudentUse
{ public static void main(String[] args)
{ Scanner scan= new Scanner(System.in);
  Student S1=new Student();
  S1.name="Tamanna";
  S1.setrollno(-100);
  System.out.println(S1.name+
      S1.getrollno);
  S2.name="Yugal";
  S2.setrollno(50);
  System.out.println(S2.name
      +S2.getrollno);
}
}
```

Tamanna

Yugal 0

Because we did not made the change in var rollno of the object S1 & S2 we just said parameter rollno= parameter rollno.

```
package classesAndObjects;
public class Student
{ public String name;
  private int rollno;

  public int getRollNo()
  { return rollno;
  }

public int setRollno(int rollno)
{ if (rollno<=0)
  {return;}
this.rollno = rollno;
  }
SOP(this);
}
```

getter

setter

we have changed the parameter name to rollno only

this here refer to the current object which called the function

```
package classesAndObjects;
public class StudentUse
{ public static void main(String[] args)
{ Scanner scan= new Scanner(System.in);
  Student S1=new Student();
  S1.name="Tamanna";
  S1.setrollno(-100);
  System.out.println(S1.name+
                     S1.getrollno);
  S2.name="Yugal";
  S2.setrollno(50);
  System.out.println(S2.name
                     +S2.getrollno);
}
}
```

Tamanna 0

Yugal 50

This is a reference to current object.

1. The "this" keyword refers to the current object in a method or constructor.

2. The most common use of this keyword is to eliminate the confusion b/w class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

* This can also be used for →

1. Invoke current class constructor.
2. Invoke current class method.
3. Return current class object.
4. Pass an argument in method call.
5. Pass an argument in current call

6. Final keyword

Sometimes, we want that a variable should be given a value only once and after that no one should be allowed to change its value so, we can mark that variable as final

We can either assign the value of final variable either while declaring it or in constructor for an object because constructor for an object is called only for once

```
package classesAndObject;
public class Student
{ public String name;
  final private int rollno;
  final double conversionfactor;
 peletic Student ( Stringnamn, int rollno)
  { this.name= name;
    this.rollno=rollno;
  }
```

conversion factor

We can't change its value now

```
package classesAndObjects;
public class StudentUse
{ public static void main(String[]args)
  { Student S1;
    S1= new Student();
    Student S2= new Student();
    S1.name= "Tamanna";
    S1.rollno= 20 ;
    System.out.println(s1.name+" "+
                       S1.rollno);
    S1.rollno= 45;
  }
```

It will give error

7. Static Keyword

→ Static properties are the properties which
a) belong to the class
b) shared by all the objects.

eg → We want to have no. of Students

```
package classesAndObjects;
Public class Students
{
public String name;
final private int rollno;
double cgpa;
final static double CONVERSIONFACTOR = 0.9;
Static int numStudents;

public Student ( String name)
{ this.name= name;
  numstudents ++;
  this.rollno = numStudents;
}
```

If we don't make it static, every object will have its own CF, but it shouldn't be like that. CF should be 1 for class ∴ static

final means it can be assigned a value only once, after that its fixed.

static means it belong to the class and not to the objects

Generally, when we are writing final static variables, the convention that we generally use is, keep all the letters in CAPITAL

Each time we will create a object, its value will increase because it belong to the Class & not the object

* Static func can't use non-static variables
* Static func can't use non-static function
* this or super keyword can't be used inside static func.

8. Static Functions

numStudents can be changed from outside easily.
So we should make it private & we can make a getter

```
package classesAndObjects;
Public class Students
{
public String name;
final private int rollno;
double cgpa;
final static double CONVERSIONFACTOR = 0.9;
Static int numStudents;

public Student ( String name)
{ this.name= name;
  numStudents ++;
  this.rollno =numStudents;
}

public static int getNumStudent( ){
   return numStudents;
}
```

foreg→ if we write SOP(rollno)
Rollno is non-static ∴ it belong to some obj. compiler will be confused about which object's rollno are we talking

This is a static function; we can only access static values in static func & not non-static values

```
package classesAndObjects;
import java.util.Scanner;
public class StudentUse
{ public static void main (String args[])
  { System.out.println (Student.
                        getNumstudent();
                       )
}}
```

main is public because we want java runtime system to access it we would not want it to be limited to only package / class

static because we would not want the object of student use class & we can directly access main via class bcz main is the first thing that run

we don't first need to create object of studentUse & then access main

Fraction Class 1] code 1
Fraction Class 2

Complex Number] code 2

Dynamic array class-1] code 3
Dynamic array class-2

Polynomial Class- Hint] code 4
Polynomial Class Problem

→ All the codes are shared in github.