DATA STRUCTURE —X—X—

Object oriented Programing (OOPS)

Objects
- Real WORLD ENTITIES ABOUT WHICH WE CODE.
- HAVE PROPERTIES AND They Perform Functions.

```
# package classes_and_objects;

Public class student {
    string name;
    int rollNumber;
}
```

within the same package u don't Have to write Import.

class is a template or a blue print and the objects are specific copies of it.

```
# package classes_and_objects;
Import java.util.Scanner;

Public class studentuse {
    Public static void main(String[] args) {
    Scanner s = new scanner(System.In);
    Student s1 = new student();
    s1.name = "Ankush";
    s1.rollNumber = 123; System.out.println(s1.name);
    Student s2 = new student(); s2.name = "manisha";
}   System.out.println(s2.name);
```

OUTPUT
Ankush
Manisha

```
# package temp;
  Import classes_and_objects.student;
Public class studentuse2 {
    Public static void main(String[] args) {
        student s = new student;
    }
}
```

Access modifiers ⟶ Sometimes u don't allow some other developer to use it.

Package classes_and_objects ;

Public class student {
  String name ;
  Int RollNumber ;
}

Default → ⟶ Accesible Inside same Package

→ INSIDE SAME PACKAGE

① default or friendly
② Private ⟶ Only within the same class.
③ Public ⟶ kahi pa bhi

- A public member of a class can be accessed in all the packages in the same project.
- A private member of a class can't be accessed outside the same class.

Constructors — SPECIAL METHOD that is called when an object is created. It is used to initialize an object.

— NAME IS SAME AS CLASS NAME.

— We can have more than one constructors within the same class. (i.e construction overloading)

- which constructor will be called will be decided on runtime depending on the type and no. of arguments specified while creating the object.

5 Feb, '20

CONSTRUCTORS

→ It is a function which basically lets u create new function.

```
Package classes_and_objects ;

Public class student {
   Public String name ;
   Private int rollNumber ;
Public void setRollnumber(int rn) {
   if(rn<=0) {
      return ;
   }
   rollNumber = rn ;
}
Public int getRollNumber() {
   return rollNumber ;
}
}
```

BY DEFAULT IT CONTAINS NULL

OUTPUT
null
0
Ankush
Manisha
0

#

```
Package classes_and_objects ;
Import java.util.Scanner ;
Public class studentUse {
Public static void main(String[] args) {
   Scanner s = new Scanner(System.in);
   Student s1 = new student();
   System.out.println(s1.name);
   System.out.println(s1.getRollNumber());
   s1.name = "Ankush" ;
   s1.setRollnumber(-123);
   Student s2 = new student();
   s2.name = "Manisha" ;
   s2.setRollnumber(121);
   System.out.println(s1.name);
   System.out.println(s2.name);
   System.out.println(s1.getRollNumber());
}
}
```

Package classes_and_objects ;

```
Public class student {
    Public string name ;
    Private int rollNumber ;

Public student(string n, int rn) {
    name = n ;
    rollNumber = rn ;
}
Public void setRollNumber(int rn) {
    if(rn<=0) {
        return ;
    }
    rollNumber = rn ;
}
Public int getRollNumber() {
    return rollNumber ;
}
Public void print() {
    System.out.println(name + " : " + rollNumber) ;
}
```

CONSTRUCTOR

```
import java.util.Scanner ;
Public class studentUse {
Public static void main() {
    Scanner s = new Scanner(System.in);

student s1 = new student("Ankush", 123);
s1.print() ;

Student s2 = new student("Manisha", 121);
s2.print() ;
```

Static —— That belongs to the class rather each specific object. So their seperate copies are not created.

Important keywords

Final

// Final datamembers can be Initialized as soon as they are declared

OR

INSIDE A CONSTRUCTOR

(this) keyword works as a reference to the current Objects whose method or construction is being invoked.

static keyword is bound to the class and not to an individual object, thus we can't make constructor static.

Fraction Class

```
Fraction f = new Fraction(2,3);
```

#

```
{
    Fraction f1 = new Fraction(20,30);
    f1.Print();
    // 2/3

    f1.setNumerator(12);
    // 4/1
    int d = f1.getDenominator();
    System.out.println(d);
    f1.Print();
    //

    f1.setNumerator(10);
    f1.setDenominator(30);
    // 1/3
    f1.print();
```

```
Fraction f2 = new Fraction(3,4);
f1.add(f2);
f1.print();
// f1 => 13/12
f2.print();
// f2 => 3/4

Fraction f3 = new Fraction(4,5);
f3.multiply(f2);
f3.print();
// f3 => 3/5
f2.print();
// f2 => 3/4

Fraction f4 = Fraction.add(f1, f3);
f1.print();   // 13/12
f3.print();   // 3/5
f4.print();   // 101/60
```

```
# Public class Fraction {
    private int numerator;
    private int denominator;

Public Fraction(int numerator, int denominator)
{
    this.numerator = numerator;
    if(denominator == 0)
    {
        // TODO ERROR OUT f3 = 4/5 x 3/4
    }
    this.denominator = denominator;
    simplify();
}

Private void simplify() {
    int gcd = 1;
    int smaller = Math.min(numerator, denominator);
    for(int i=2; i<=smaller; i++) {
        if(numerator % i == 0 && denominator % i == 0)
        {
            gcd = i;
        }
    }
    numerator = numerator/gcd;
    denominator = denominator/gcd;
}
```

```
Public int getDenominator() {
    return denominator;
}

Public int getNumerator() {
    return numerator;
}

Public void setNumerator(int n) {
    this.numerator = n;
    simplify();
}

Public void setDenominator(int d) {
    if(d==0) {
        // TODO error out
        return;
    }
    this.denominator = d;
    this.simplify();
}

Public void print() {
    if(denominator == 1) {
        System.out.println(numerator);
    }
    else {
        System.out.println(numerator + "/" + denominator);
    }
}
```

```
Public static Fraction add(Fraction f1, Fraction f2)
{
int newNum = f1.numerator * f2.denominator +
             f2.numerator * f1.denominator;
int newDen = f1.denominator * f2.denominator;
Fraction f = new Fraction(newNum, newDen);
return f;
}

Public void add(Fraction f2) {
this.numerator = this.numerator * f2.denom
                 +
                 this.denominator * f2.numerator;

this.denominator = this.denominator * f2.denom
simplify();
}

Public void multiply(Fraction f2) {
  this.numerator = this.numerator * f2.num
this.denominator = this.denominator * f2.denom
    simplify();
}
```

Complex Numbers

$C_1 = 4 + j5$ and $C_2 = 3 + j1$

C1. Plus(C2) results

$C_1 = 7 + j6$ and $C_2 = 3 + j1$

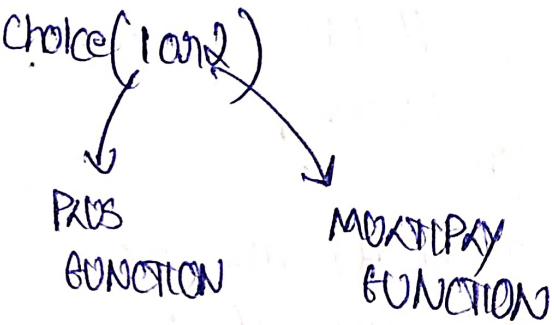
this.real = this.real +

$C_1 = 4 + j5$ and $C_2 = 1 + j2$

$(4 + j5)(1 + j2)$

$4 + j8 + j5 + (-1)10$

$-6 + 13j$



~~C. plus~~

plus( )

Dynamic Array class

→ When you don't know how many elements you put in an Array.

→ It Internally store data in Array.

```
Package classes_and_objects;

Public class DynamicArrayUse{

  Public static void main (String[] args)
{
   DynamicArray d = new DynamicArray();
   for(int i=0 ; i<100 ; i++)
    {
      d.add(i+10);
    }

   System.out.println(d.size());
   d.set(4,10);
   System.out.println(d.get(3));
   System.out.println(d.get(4));

   while(!d.isEmpty()) {
    System.out.println(d.removeLast());
    System.out.println("size = " + d.size());
   }
}

}
```

```
Package classes_and_objects;

Public class DynamicArray {
    Private int data[];
    Private int nextIndex;

Public DynamicArray() {
    data = new Int(5);
    nextIndex = 0;
  }

Public int size() {
    return nextIndex;
  }

Public void add(int element) {
    if (nextIndex == data.length) {
        restructure();
    }
    data[nextIndex] = element;
    nextIndex ++;
}

Private void restructure() {
    Int temp[]=data;
    data = new Int[data.length * 2];
    for(int i=0; i<temp.length; i++)
    { data[i] = temp[i];
    }
}

Public void set(int index, int element) {
    if (index > nextIndex) {
        return;
    }
    if (index < nextIndex) {
        data[index] = element;
    }
    else {
        add(element);
    }
}

Public int get(int index) {
    if (index >= nextIndex) {
        return -1;
    }
    return data[index];
}

Public boolean isEmpty() {
    if(size()==0) {
        return true;
    } else {
        return false;
    }
}

Public int[] toArray() {
```

```
Public int removeLast() {
    if(size()==0) {
      //error out
      return -1;
    }
  int value = data[nextIndex-1];
   data[nextIndex-1] = 0;
   nextIndex--;
   return value;
}
```

~~////POKEMON////~~

Component of OOPS

→ PUTTING DATA and function in same class

1. Encapsulation
2. Inheritance
3. Polymorphism

```
Package vehicle;

Public class vehicle {
    String color;
    int maxspeed;

    Public void print() {
        System.out.println("vehicle color :" + color);
        System.out.println("vehicle speed :" + maxspeed);
    }
}
```

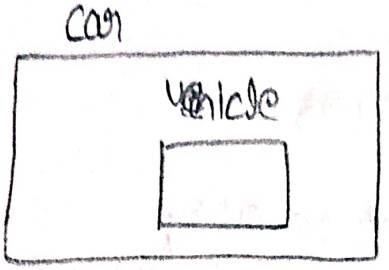
```
Package vehicle;
Public class vehicleuse {
    Public static void main(String[] args) {
        vehicle v = new vehicle();
        v.print();

        Car c = new car();
        c.numGears = 10;
        c.color = "Black";
        c.print();
    }
```

```
Package vehicle;
Public class Car extends Vehicle
{
    int numGears;
    boolean isConvertible;
}
```

Class Car is inheriting Properties of class vehicle.

Protected
→ friendly
→ outside package subclass

Public class car extends Vehicle
↓ child, subclass, derived
↑ Parent class, Super class, base class



friendly or default
→ Available only inside Package

super.print(); → Parent class ka obj ko Print krvata h
NOT GRAND PARENT

- Access modifiers in increasing order of their visibility
  Private, Default, Protected, Public

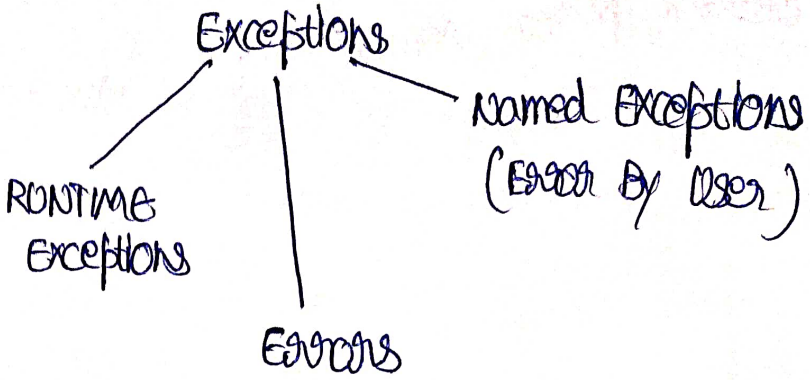
Inheritance and Constructors

Default constructor nahi Ha to super() ko call krna Pda ga Parameter da ka in derived class.

POLYMORPHISM

→ something taking multiple form

Exception Handling




```
Exception e = new Exception("Denominator cant be 0");
throw e;
```