OOPS in Java

Major use → Program Refactorization (ie resizing Program into shorter one and easy access)

* Object

Each var, data is called as objects

* class

~~data d d~~ collection of data & data members function is called class, By default in Java all members are defined in Public

* Inheritance :

linking of two classes

* Polymorphism

function overloading or overwriting

* abstraction :

Packages & interface

(data hiding)

Single inheritance

One Parent & one child class

For inheritance of one class from another we use extends

```
class circle
  {
    void area(int r)
      {
        Statements;
      }
  }
```

Parent class

```
class Perimeter extends circle
(// class circle variables is inherited in class
  Perimeter)
  {
    void Peri(int r)
      {
        Statements;
      }
  }
```

child class

```
Public class example
  {
  Public Static void main (String args[])
    {
    Scanner Sc = new Scanner(System.in)
     Statements;
      :
      :
      :
  Perimeter Pe = new Perimeter();
    Pe.area(r);
    Pe.Peri(r);
  }
  }
  }
```

main class

note:
We can have create objects for child class during inheritance, if we create object for Parent class alone then we can't able to access child class If we create object for child class then both Parent & child class can be accessed bcoz Parent class is inherited in child class

multilevel inheritance

~~It's process of when~~
It's chain process of inheritance

class A
↓
class B
↓
class C
↓
class main

class A inherited into class B
class B inherited into class C
[so class C can access A & B variables]
using objects class C [child class] is called into main class.
[main class can now access A, B, C variables.

Eg
~~Public~~ class A
{
void name ()
{
statements;
}
}
class B extends A
{
void age ()

Parent class
child class for A
Parent class for C

```
{
  Statements;
}
}
class C extends B
{
void voter_id ()
{
  Statements;
}
}
Public class Main
{
Public static void main (String args[])
{
Scanner S= new Scanner(System.in);
Statements;
.
.
.
C obj = new C();
obj.name(str);
obj.age(a);
}
}
```

Child class for B, A
Parent class

main class

Object In multiple must be created last child class then only u can access previous class elements

hierarchical inheritance

one Parent class, many child class
e.g. The elements of Parent class A will be inherited to class B, class C & class D)

Eg

```
class A
{
    void methodA()
    {
    }
}
class B extends A
{
    void methodB()
    {
    }
}
class C extends A
{
    void methodC()
    {
    }
}
class D extends A
{
    void methodD()
    {
    }
}
class JavaEg
{
```

Parent class

Child class-1

Child class-2

Child class-3

```
Public void static name (String args[])
{
    B obj1 = new B();
    C obj2 = new B();
    d obj3 = new D();
    Statements;
    :
    :
}
}
```

main class

note:

Always ~~remember~~ keep in mind object should be created to child class in inheritance

The main reason if we create obj for child class we can access child class and Parent class also

In Java multiple inheritance can't be implemented, instead of that we are using abstraction

abstraction :

abstraction has two method

abstract & interface

- abstract → 0-100% accuracy
- interface → 100%

In abstract class we can create abstract ~~class~~ method as well as normal method, If we abstract method in class, we have to define it in next class

```
abstract class abstract
{
    abstract void add();
    void sub()
    {
    }
}
class abstract2 extends abstract
{
    void add()   { defination of abstract method done here }
    {
    }
}
```

Interface

In interface we can only create abstract function, normal function can't be created. But defination of abstract function must be done in class. we can define normal method in interface either by default function name() or static function name()

Eg

```
interface Inter
{
    abstract void add();
    default void mul();
    {
        statements;
    }
}
class Inter2 implements Inter
{
    Public void add()
    {
```

```
    }
}
Public class Inheritance
{
    P.V.S.M (String args[])
    {
    }
}
```

aggregation

(For simple understanding)

It is accessing one class from another through objects

method - I

```
class Demo
{
int a,b;
}
class Demo1
{
  void display()
  {
    Demo d = new Demo();
    Scanner sc = new Scanner(System.in);
    d.a = sc.nextInt();
    d.b = sc.nextInt();
  System.out.println(d.a+d.b);
  }
}
Public class class_Ex
{
  Public Static void main
  {
    Demo d = new Demo();
    Demo1 d1 = new Demo1();
    d1.display();
```

[accessing a & b of Demo in Demo1 class]

```
}
}
```

method II

```
Class Demo
{
int a, b
}
class Demo2
    void display1 (Demo d)
    {
    Scanner sc = new Scanner (System.in)
        d.a = sc.nextInt();
        d.b = sc.nextInt();
System.out.println (d.a - d.b);
    }
}
Public class class_Ex
{
    Public Static void main (String args[])
    {
        Demo d = new Demo();
        Demo2 d2 = new Demo2();
    d2.display1 (d);
    }
}
```

directly passing class through a variable in the function

note:

If we are using method I, we have to just the object for Parent class in child class and we pass the element in second class then into main class But if we are using method II we have create object for Parent & child class in main class, since Parent class passed into child by a variable

Polymorphism

method overloading
Same method name but different arguments in a single class

method overriding
Same method name, but different arguments in two separate classes

overloading

```
class Poly
{
void display()
{
SOP ("overload1");
}
void display(int a)
{
System.out.Punth ("overload1"+a);
}
}
```

overriding

```
class Poly1
{
void display ()
{
SOP ("overriding");
}
void display (int a)
{
SOP ("overriding");
}
}
```

```
Public class Polym
{
  PSVM (String [] args)
  {
    Poly P = new Poly ();
     P.disPlay ();
     P.disPlay(5);
   Poly1 P1 = new Poly1 ();
    P1.disPlay ();
    P1.disPlay (7);
   }
}
```

Encapsulation

It's process of wrapping up the data and variable, (For better understanding what variables we create in Java code, can be called in another separate Java code by setter and getter method.

Eg

```
Public class Student {

Private String name;   [Private data members]

Public String getName ()
  {
   return name;        [getter method]
  }
Public void setName (String name)
  {
  this.name = name
  }
}
```

```
class Test{
Public static void main (String [] args)
{
Student S = new Student ();
// Setting value in the name member
S.setName ("Vijay");
// getting value in the name
System.out.println (S.getName());
}
}
```

This function

This function specifies the value of the variable in current class

For example ~~if we specify~~ a

```
class Test
{
    void ~~dis~~ ab()
    {
    int a, b, c;
    }
    void inPut()
    {
    ~~this.a this.a()~~.
      this.a = a;
      this.b = b;
      this.c = c;
    }
  Public static void main (String[] ~~args~~)
  {
  ~~main ab~~ inPut obj = new inPut();
  System.out.println ("value
      of x=" + obj.x);
```

}
}

If we have variable a, b, c and if we have use over and over in order to avoid confusion this() is specified.

String methods

Creating a String

String str1 = "welcome";

str2 = new String ("welcome");

methods of Strings

* str1 == str2 // compares address
* String newStr = str1.equals(str2);
  // compares the values ~~ignoring the~~
* newStr = str1.length()
  Calculate the length
* String newStr = str1.equalsIgnoreCase()
  // compares the values ignoring the case
* newStr = str1.charAt(i)
  // extract ith character
* newStr = str1.toUpperCase()
  returns the str in caps
* newStr = str1.toLowerCase()

Returns string in lowercase

* newstr = str1.replace(oldval, newval)
  search and replace

* newstr = str1.trim()
  trims surrounding whitespace

* newstr = str1.contains("value");
  check for the values

* newstr = str1.toCharArray();
  convert string to character type

* array newstr = str1.isEmpty();
  Check for empty string

* newstr = str1.endswith();
  checks if string ends with the given suffix

* concatination
  combines two strings

```
class stringmutable
{
  Public static void main (String args[])
  {
    String S = "Javastrings";
    S.concat("cheatsheet")
    System.out.Println(s)
  }
}
```

output:

JavaStrings cheatsheet

* String to int conversion

```
String str = "123";
int inum1 = 100;
int inum2 = Integer.ParseInt(str);
```

capital

* int to String conversion

```
int var = 111;
String str = String.valueOf(var);
```

* String to Double

```
String str = "100.222";
double dnum = Double.ParseDouble(str);
```

capital

* Double to String

```
double dnum = 88.9999;
String str = String.valueOf(dnum);
    // conversion using valueOf() method
```

String Buffer

It's changeable & edutable.

String Buffer class doesn't override the equals() method of object class

String Builder

It's more efficient than String Buffer as it's not synchronized

Recursion

It is process in which method is called itself continously

Eg

```
int factorial (int n)
  {
    int k=1;
    if (n==0)
    {
    return k;
    }
    else
    return n* factorial (n-1);
```

association

It defines the connection between two classes that are set up through their objects

one to one

one to many

many to many

Eg

```
Public class Method
  {
  Static int fact (int n)
      {
        int fact=1
        for (int i=1 ; i<=n; i++)
        {
        fact = fact * i;
        }
        return fact;
        }
```

String Buffer methods

~~append()~~ —

* append
  newStr = str1.append()
  Joins old string with new one, not only string, it can join int, char, Boolean

* newStr = str1.Insert(int offset, String)
  // combines the string at Particulas index

* newStr = str1.Replace(Start index, end index, String)
  // It replaces a string to old string with Particular indexes which we want

* newStr = str1.Delete(start index, end index)
  // It deletes a string between starting value & end value)

* newStr = str1.Reverse()
  // It reverse the string

* newStr = str1.capacity()
  // It returns current capacity of string

* newStr = str1.EnsureCapacity(min capacity)
  It ensures string contains minimum specified capacity

* newStr = str1.charAt(int)
  It returns the character at the Particular specified index
  (returns means prints)

* newstr = Str1.length()
  It finds the length of the string

* newstr = Str1.Substring(begin index, end index)
  // (It returns the specified substring from the original string using begin value & end value)