STRING

In Java, a string is an object that represents a sequence of character.

Java.lang.String Class

Creating a String

```
String str1 = "Welcome";
// Using literal String
Str2 = new String("Edureka")
// using new keyword.
```

String is Immutable

We can not add or change string

charAt() Method

return character at specified index of a string.

```
String str1 = "Welcome";
char ch1 = str1.charAt(0);
                // W
        charAt(int Index);
```

compareTo()

Each character of both the string
If equal to ~~that~~ string = 0
Not equal = + or negative

```
String str1 = "Welcome";
String str2 = "Welcome";
int var1 = str1.compareTo(str2);
          // => 0
```

If length of string ~~of~~ check ~~for~~ empty string ~~&~~ ~~for~~ compare ~~for~~ !!!

```
String str1 = "Hello";
String str2 = "";
```

```
int var = str1.compareTo(str2);
          //=>5
        = str2.compareTo(str1);
          //=> -5
```

- case sensitive ~~है~~ !
- compareToIgnoreCase()
  => case ignore ~~or~~ ~~करता~~

contains()

~~ये~~ check करता है कि string में वो word है या नहीं।

```
String str1 = "Game of Throne";
S.O.T(str1.contains("Game")
        // True
```

concat()

~~ये~~ string जोड़ने का काम करता है।

```
String str1 = "Welcome"
String str2 = "Sonny";
String str3 = str1.concat("Versha")
String str4 = str1.concat(str2);
```

=> Welcome Versha
Welcome Sonny.

substring()

```
String str1 = "Welcome";
S.O.T(str1.substring(1,5);
                  start  End.
    // elcom
```

Why String is Immutable

=> String object are ~~all~~ cached in the String pool and it makes the string immutable. The cached String literals are accessed by multiple clients. So there is always a risk; where action perform by one client affect

~~value of~~

valueOf()

conversion का काम करता है।

```
int num = 23;
String str = String.valueOf(num);
    ⇒ 23 string
  = String.valueOf(boolean b);
    String.valueOf(char);
String.valueOf(int i);
String.valueOf(double d);
String.valueOf(float f);
String.valueOf(long l);
```

startsWith()

check करता है कि word से string start हो रहा है या नहीं

```
String S = "This is Just a string";
S.O.T(S.startWith("This"));
    // True
S.O.T(S.startsWith("Hi"));
    // False
```

किस index से word start हो रहा है या नहीं use भी check कर सकते हैं

```
S.O.T(S.startsWith("Just" 8);
    // True
```

equals()

2 string equal है या नहीं use check करना।

```
String str1 = "Welcome";
String str2 = "Welcome";
S.O.T(str1.equals(str2))
    // True
```

equalsIgnoreCase()

upper case or lower case को ignore कर देता।

format()

String के add करने का काम और किस format में लिखता है, वो भी काम करता है।

```
String str = "Sonny";
String s1 = String.format("My
        ("My name is %S", str
// My name is Sonny.
String.format("My string is %.6f"
        ,121-1121);
// My string 121.121100;
String.format("My string
%1$s %1$s and %2$s, str1, str2)
%05d", str);
```

⇒ Total 5 digit का string बनाना और digit के साथ।

```
%c = Character
%d = Integer
%s = String
%o = Octal
%x = Hexadecimal
%f = Floating Number
%h = hash code of value.
```

endsWith()

Same

Last से word को check करेगा

indexOf()

first occurance of particular character का index बताता

```
String str = "Sonny";
S.O.T("str.indexOf(n));
    //⇒2
    str.indexOf(n, 2));
        ↓
    //3
```

2 के बाद n के index को check करेगा

String

lastIndexOf()

backward direction से check करता है।

```
string str = "beginnersbook is
              for beginnners";
char ch = 'b';
char ch1 = 's';
int n1 = str.lastIndexOf(ch);
        // => 21
                         (ch1);
        // => 29
```

length()

length of string देता है।

```
string s1 = "sonny";
s.o.T( s1.length())
      //5
```

replace()

⇒ जहा जहा वो character होगा उसे replace कर देगा

```
string str = "sonny nyna";
s.o.t (str.replace('n', 'k')
```

old को replace ↓ new

```
// sukky kyka
```

replaceFirst()

First occurrence of character या word को replace कर देगा

replaceAll()

सब character या word को replace कर देगा

split(string regex, limit)

```
string str = "sonny@kumar";
string str1 = str.split("@")
=> sonny, kumar.
```

split(regex 2)

```
s.split("[,;#$]");
```

सब के basis पे split कर देगा

2 → 2 substring में devide कर देगा

-5 → unlimited substring में devide कर देगा

trim()

start और end of string के white space को remove कर देता But बीच में से नहीं)

```
string str = "_ Sonny _";
s.o.t (str.trim());
```

hashCode()

string को hashcode में convert कर देता है।

isEmpty()

string null है या नहीं use check करता है।

regionMatches()

string के किस position index से किस index तक match करवाने का काम करता है।

```
string str1 = "Hello How are you";
string str2 = "How";
S.O.T (str1.regionMatches(7, str2, 0,3)
```

string1 के 7 से index को ↙ ↓ string2 के 0,3 के साथ Match करेगा

toCharArray()

string को character array में convert कर देता है।

```
string str = "My Sunny";
char[] array = str.toCharArray();
```

getBytes()

String को Bytecode में convert करता है।

join()

character के बीच में कोई pattern या किसी special character के साथ join करता है।

```
String str = String.join("^", "you",
"are", "Awesome");
S.O.T(str)
you^are^awesome
```

copyValueOf()

Array के data को copy करता है। But string को value को set करता है।

```
char[] data = {"a","b","c","d"}
String str1 = "Text";
str1 = str1.copyValueOf(data);
S.O.T(str1)
// ⇒ abcd
```

String to Integer :

```
Int num = Integer.parseInt(string);
int num = Integer.valueOf(string);
int var = 11;
String str = Integer.toString(var);
           = Double.parseDouble(string);
long.parseLong(string);
```

Why string are Immutable

⇒ The string object are cached in the String pool and it make the string immutable.

The cached string literals are accessed by multiple clients So there is always a risk, where action performs by one client affect the other client. For eg If one client performs an action and change the string value from Pressure to PRESSURE all remaining clients will also read that value.

So to remove the risk we ~~have~~ string is Immutable.

String Pool ⇒ is a pool of String stored in Java heap memory.

String Buffer.

create an Empty String buffer with the initial capacity 16

String Buffer & String Builder is Mutable.

append()

To concatenate the given argument with the String

```
StringBuffer sb = new StringBuffer("Hello");
sb.append("Java");
S.O.T(sb) ⇒ Hello Java
```

insert()

insert string at given position

```
StringBuffer sb = new StringBuffer("Hello");
sb.insert(1,"Java")
SO.T(Sb) // HJavaello
```

replace()

replace the specified string at begin Index & End Index.

```
Sb.replace(1,3,"Java")
          star end
```

HJava|o 1–3 तक हटा कर replace कर दिया

delete()

delete the string from beginIndex & EndIndex.

```
Sb.delete(1,3)
// ⇒ Hlo
```

reverse()

```
Sb.reverse();
// OLLeH
```

capacity()

current capacity of buffer

default capacity of buffer = 16

(oldcapacity * 2) + 2 = New.

(16 * 2) + 2 = 34

```
StringBuffer sb = new StringBuffer();
S.O.T(sb.capacity())
// ⇒ 16
```

charAt()

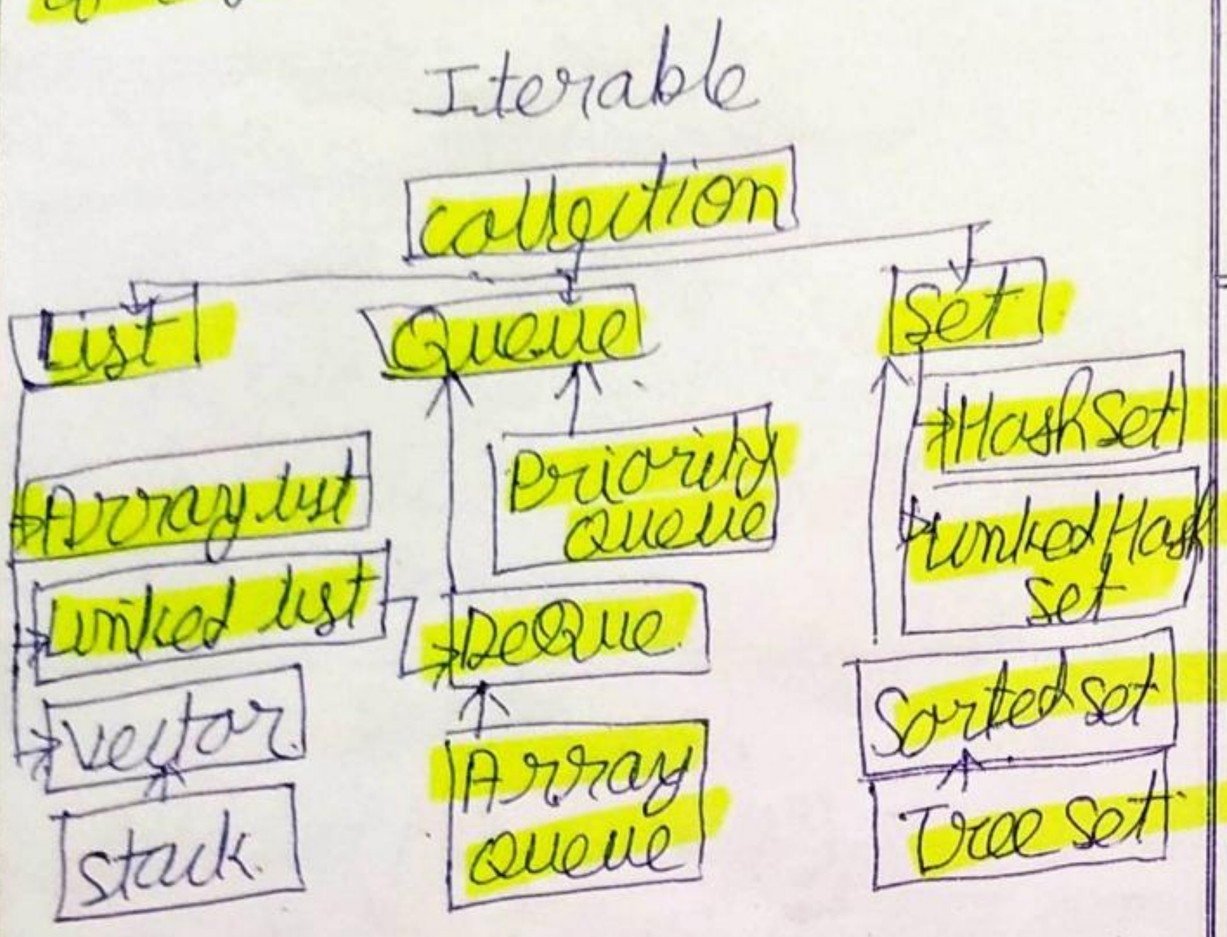
length()

substring()

String	String Buffer.
String class is immutable	String Buffer class is mutable.
String is slow and consume more memory when you concat too many string because every time it creates new instance	String Buffer is fast and consume less memory when you concat strings.
String class override the equals() method of object class So you can compare the content of two string by equals() method.	String Buffer class doesn't override the equals() method of object class.

Collection

Collection in Java is framework that provide an architecture to store and manipulate the group of objects.



size.() ⇒ return size of list
get() ⇒ get the element of list
add()⇒To add element in list

sort()

```
collection.sort(list1);
// ascending order में list sort हो जाएगी
// reverse order में sort
collections.sort(list1, Collection.reverseOrder());
```

List Interface

```
List<data-type> list1 = new ArrayList<>();
                = new LinkedList();
                = new Vector();
                = new Stack<>();
```

Searching in collection.

```
Collection.binarySearch(list1, "Fruits")
```

addAll()

```
Import java.util.ArrayList
     ''     ''.Collection;
            .List;
List<String> list1 = new ArrayList<>();
list.add("Shoes")
list.add("Toys")          // Add all m
Collection.addAll(list1, "Fruits" "Bats")
for(int i=0; i<list1.size(); i++){
    S.O.T(list1.get(i)+" ")
```

get method

• size method.

// Shoes Toys Fruits Bats.

copy()

copy element at the same index

```
Collection.copy(destination_list, source_list)
```

remove()

```
remove(2,)"Sonny
remove("Sonny")
```

• indexOf()

return indexOf element

nCopies()

```
Collections.nCopies(10, 5);
```

nTimes element

```
ArrayList<String> list1 = new ArrayList<String>();
```

List

Araylist <Integer> list1 = new ArrayList<Integer>();

Same all the property of Collection.

ArrayList ~~Linked list~~

list1.add("Sunny");
List1.add(3, "Sunny");
// This will add Sunny at the 4th position.

set()

list1.set(0, "Lucy")
0th position पे Lucy को डाल देगा और 1st को remove कर देगा

clear() remove() removeAll()

list1.remove("Sunny");
list1.remove(2);

size()

return size of arraylist

Collections.sort(list1)
For sorting Arraylist

sublist()

जहाँ से जहाँ तक का value चाहें उसे ले सकते हैं
list1.sublist(1, 4)
start → end.

contains()

list1.contains("Sunny")
// True

swap()

Collections.swap(1, 4)
1st element को 4th पे डाल देंगे और 4 को 1st पे।

clone()

जो original copy होता है वही clone return करता है। remove नहीं कर पाता
list1.clone()
हर हाल में original list को return करता है।

isEmpty()

check list empty or not

trimToSize()

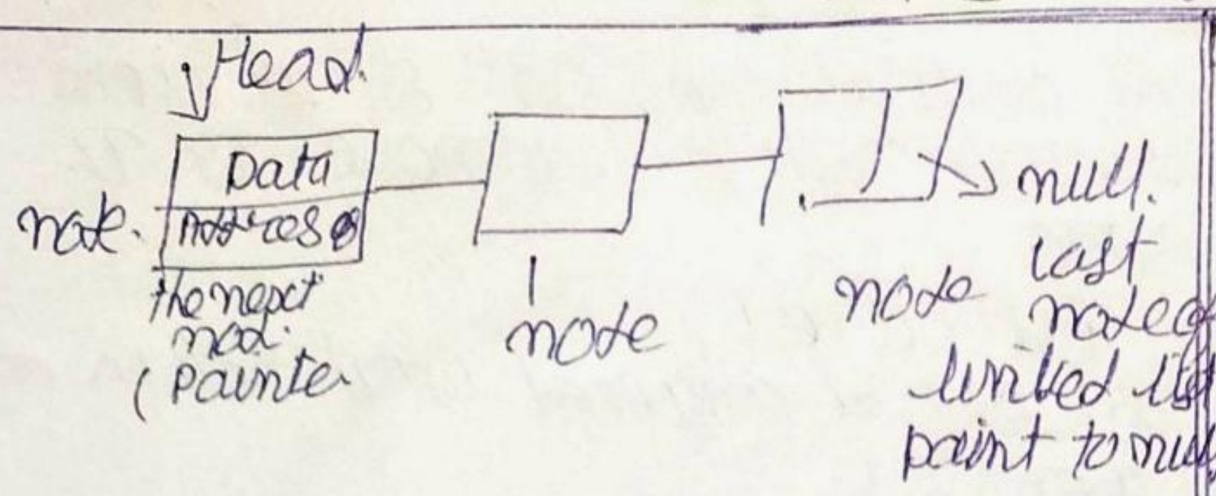
If capacity of Arraylist is 15 & we have given only 5 element after applying trimToSize it will change the capacity of arraylist 15 → 5.

ensureCapacity()

जितने capacity का arralist चाहिए उतने capacity का बना सकते
list1.ensureCapacity(5)

Arralist has

Linked List



add() ⇒ last of element add
addFirst() First — — —
addLast() last
remove() = last of element
removeFirst() First —
removeLast()
.add(4, "New element")
.offerFirst() // add the elem to the front of list
removeAll()
clear()
get()
getFirst()
getLast()
.indexOf()
.lastIndexOf()

Iterator list2 = list1.descendingIterator();
↖ reverse order of list
print out

Set(int index, Element)
contain()
clone()

push() ⇒ Insert the element to front of list
pop() ⇒ remove first element of list

pall() remove head of the element (1st element)
PallFirst()
// remove 1st element of list
pallLast() // remove last element of the list.

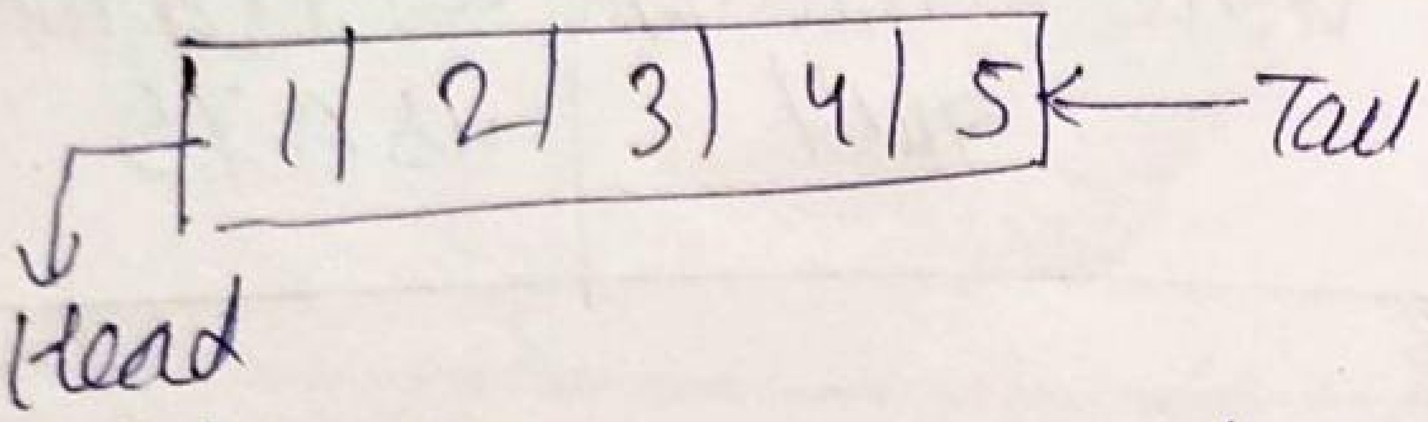
Peek() | Retrieves, but does not remove the head
PeekFirst() | ⇒ 1st element
PeekLast() | ⇒ last element

Queue

FIFO (First In First Out)

	Linked list	Priority Que
Insertion Order	✓	✓
Duplicate	✓	✓
Hetrogenius	✓	X

Insertion Order (जिस order में डालते हैं उसी order में मिलता है।)



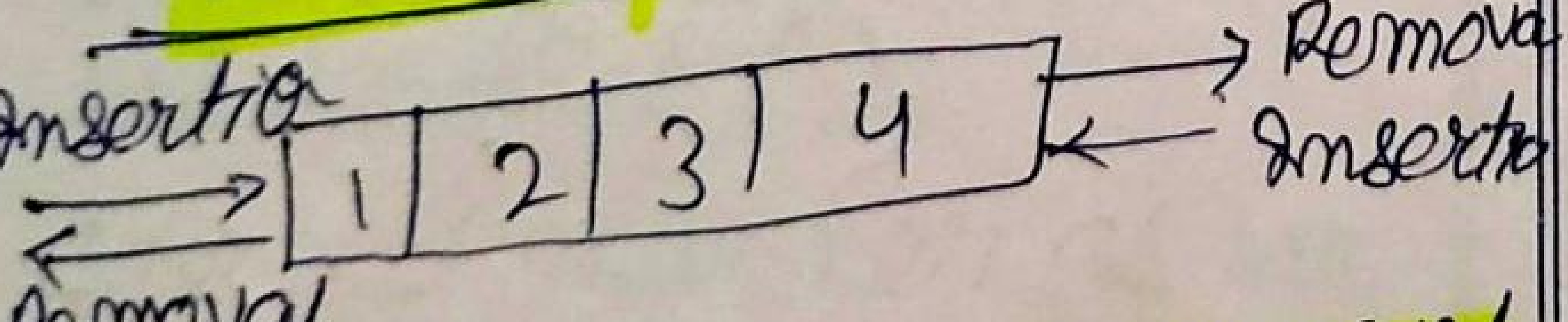
add() } add element in
offer() } que at the last.
(offer() → T/F)

add() → successful → True; Not successful → Exception return करता।

element() — T / Exception
peek() — T/F
return head element

remove() — Exception
poll() — F
return or remove head element

Deque



- In deque Insertion & removal take place at both end.
  Not Follow FIFO
- Input Restricted Deque ⇒ Input is restricted at one end, but deletion ⇒ both end.
  Output restricted deque ⇒ Output one end ⇒ Input both end.

insertFront()
insertLast()
deleteFront()
deleteLast()
getFront()
getRear()
isEmpty()
isFull()

Deque<String> deque = new LinkedList<String>();

offerFirst()
offerLast()
same all the method of queue

Map & HashMap

Elements are store in (key & value pair

{1, Sonny} {2, Varsha}

(1) Insertion order not maintain
(2) Duplicate keys are not allowed
(3) Duplicate values are allowed
(4) Null key allow once.
(5) Null values multiple times

Method of HashMap.

list.put (key, value)
list.putAll (key, value) ← multiple times
list.get (key) → return value
list.remove (key) ⇒ remove pair
list.contain (key)
list.contain (value)
list.isEmpty ( )
list.size ( )
list.clear ( ) ← remove all
list.keySet ( ) ← combination of all key will return
list.values ( ) ← all values of hashMap
list.entrySet ( ) ← return all pair of key, values.
list.entrygetKey ( )
list.entrygetValues ( )
list.entrySetValues (object)

HashMap	HashTable
NonSynchronized multiple thread allow	Synchronized thread allow at a time.
Not thread safe.	Thread Safe
performance good.	Performance be
Null accept keys - once Values - multiple null	Null can not accept
	capacity of HashTable is 11 and Loade is 0.75

Set

duplicate value cannot be allow.

Set
↓
Hash Set
↓
Linked Hash Set
↓
Sorted Set
↓
Navigable Set
Tree set

Set
↓
Sorted set
↓
Navigation Set
↑
Tree Set ← Implem

Sorted set — Navigable Set
↑
Treeset.

- Store the element in sorted manner.
default ascending order.
descending order.
TreeSet.descendingIterator();

Hash Set()

HashSet<String> list1 = new HashSet<String>();

- Duplicate element not allow
- Not in same order as we provided.
- Null element हो सकता है.
All method of collection frame work.

Linked HashSet()

LinkedHashSet<String> list1 = new LinkedHashSet();

- ordered manner as we provided.
- Duplicate element is not allowed.
- All method of collection frame work.

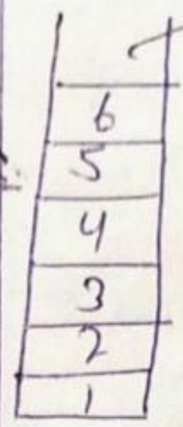
Vector

Vector <String> list1 = new Vector <String> ();
Default capacity of Vector = 10

Sublist()
Vector.setSize() ⇒ set the size of vector.
All property of ~~the~~ collection

STACK

LIFO
(last In First Out)



Stack <String> list1 = new Stack <String> ();
- Push (4)
- push ("AL")

peek() ⇒ Tap of the elemon
- pop() ⇒ remove tap of element

Oops Concept

Classes ⇒ A class is a blueprint from which individual object creat

```
Public class Animal {

}
```

Local variable = Variable define inside method, constructor or block are called local variable

Instance variable = Variable with in a class but outside any method. Variable initialized when class is instantiated

Class Veriable = A variable declare with in a class, outside any method, with the static keyword.

Types of Constructor

Default | Noargs | Parameterized.

Default Constructor.

Default Constructor compiler द्वारा create कर दिया है। Start में object create करे 3रा time.

```
Public Myclass {
    P.S.V.M(String[] args)
    {
    Myclass obj = new Myclass();
    }
    ---
}
```

⇒

```
Public Myclass {
    Myclass() {
    }
    P.S.V.M {
    My class obj = new My class();
    }
```

- Constructor can not be abstract, final & static & Synchronized
  Method ।। be ।। ।। ।।

No-arg Constructor.

Parameter नहीं होता है

```
Class Demo{)
    Public Demo(){
    }
```

constructor should have same name of class.

Parameterized Constructor.

```
Class Demo {
    Public Demo(int Id, String name){
    }
}
```

- constructor do not have return type & Method have.

```
Public static Void Main(String[] args){
Demo obj = new Demo(100, Sonny
```

Method.

```
Public int DisplayDetail(){
    return(.) S.O.P = Sonny
Demo obj = new Demo();
obj.DisplayDetails()
// Sonny -
```

If we create constructor of parametrized & we can't giving their value then ⇒ Compilation error.

- The purpose of constructor is to initialize the object class.
- Metho is to perform a task by executing Java Code.

super()

```
class MyParentClass {
    MyParentClass() {
        S.O.T("Sonny")
```

```
class MyChild extends MyParentClass {
    public MyChild() {
        S.O.T("MyChild constr.")

P.S.V.M
new MyChildClass();
```

Super class automatic Insert हो जाता है अगर parent class के feature को Inherit करता है। child class में ही child class में constructor में पहले super() keyword automatic Insert हो जाता है।

- refer to object of Immediate parent class.

- static constructor का कोई concept static keyword parent class में child class में extend करने पर पाता उसकी जगह static block use कर सकता है।

```
static {
    S.O.T("static block of child class")
}
```

Static Method.

⇒ object create किए बिना हुम static method को use कर सकता है।

```
class ... {
    static void method() {

    }
    P.S.V.M(
        Mymethod();
```

→ return करता है।

Static Block

Static variable को call करने के Static Block को use करते है।

```
static int num;
static String name;

static {
    num = 97;
    name = "Sonny";
}

P.S.V.M.
S.O.T(num)
S.O.T(name)
```

- A class can have multiple static blocks.

1st static block execute होने के बाद 2nd static block execute होता।

Static Variable.

Only a single copy of ~~instance~~ static variable is created and shared among all instance of the class.

- If we want to have a common value of all the instance like global variable.
- save memory.
- cannot be change static variable.

Global Variable = A variable which declare outside function or block is known as global variable.

Any statement in the entire program can access variables.

Method Overriding

same method को parent class और child class में call करे with same name तो उसे Method overriding कहते है।

Aggregation

HAS-A relationship

student HAS-A address

But address HAS nata student. full code on notes.

To maintain code reusability

- runtime happen / Dynamic binding

Method overloading

Same method in same class with different parameters (argument or types)

~~pubt~~

```
class Display overloading {
  public vaid display (char c)
      S.O.C (c)
  public vaid display (int num, String)
  p.s.v.m (Str ar)
  Display overloading obj = new
         Display overloading();
  obj.display('A')
  obj.display(12, sunny)
```

- happen at compile time
- static binding
- Static polymorphis

No of argument

Type of argument

Access Modifier

	class	Package	subclass	subclass	Outside
		same Package		Different Package	class
Public	✓	✓	✓	✓	✓
Protected	✓	✓	✓	✓	X
default	✓	✓	✓	X	X
Private	✓	X	X	X	X

Polymorphism

allow us to perform a single action in different ways.

```
public class Animal {
   Public vaid sound() {
     S.O.T (Animal sound
   }
}
public class Horse extend Animal {
   @Override
   Public vaid sound () {
      S.O.T ( Horse sound)
}
public class catt extend Animal {
   public vaid soun() {
     S.O.T ( catt )
   }
P.S.V.m (      ) {
Animal obj = new catt(); // cat sc
  obj.sound();
Animal obj = new Horse();
  obj sound; // Horse sc
```

Types of polymorphism

→ Static Polymorphis — compile time — overloading

→ dynamic polymorphis — Runtime. — overriding

Inheritance

- The process by which one class acquire the properties of another class is called Inheritance
- reusability of code.

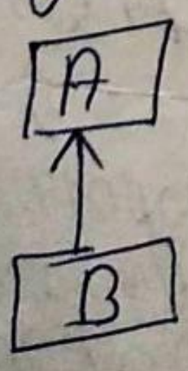
```
class Teacher{
    String designation = "Teacher";
    String collegeName = "BBSTC";
void does(){
    S.O.T("Teaching")
}
}
Public physicsTeacher extends Teacher{
    String mainsubject = "Physics";
    P.S.V.M(S1 ){
    PhysicsTeacher obj = new PhysicsTeacher();
SOT(obj.collegeNam);
    obj. designatic
    obj. mainsubje
    obj.does();
```

1. S.B.S.S.T
   Teacher
   Physics
   Teaching

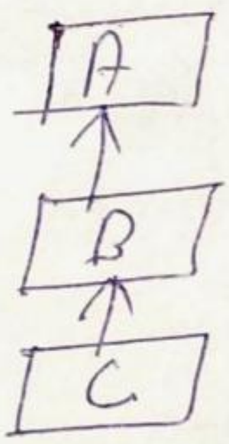
~~P Teacher~~ PhysicsTeacher IS-A relationship with Teacher.
IS-A relationship b/w child & parent class.

Types of Inheritance

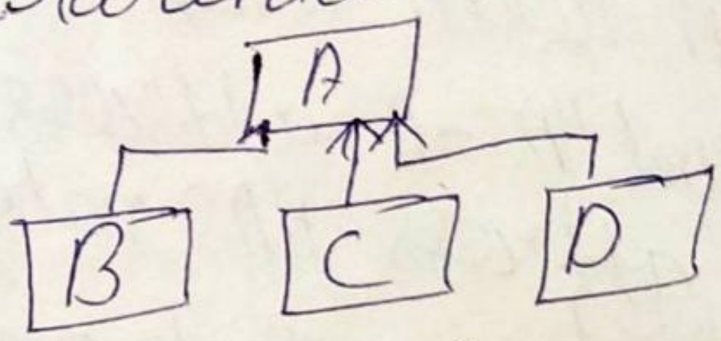
Single Inheritance



Multilevel Inheritance

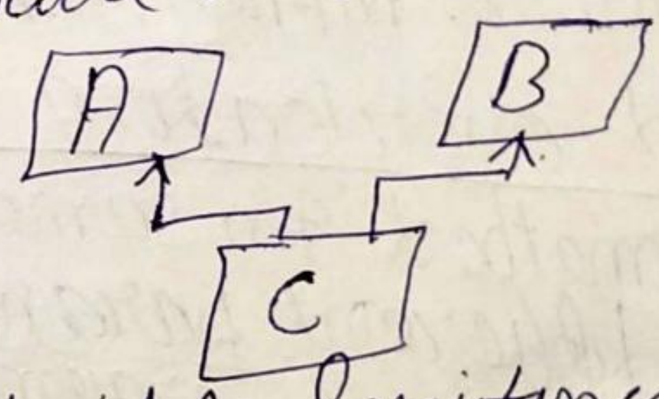


Hierarchical Inheritance

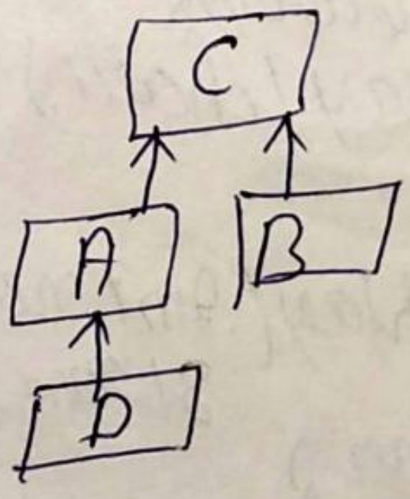


Multiple Inheritance

(Java does not support)



Hybrid Inheritance



a Abstraction.

displaying only the essential information and hiding the details. background.

Abstract class cannot be instantiated ~~to~~ means you can not create object of it.

It may or may not contain abstract method.

To use abstract class you have to inherit it from subclass.

abstract method does not have body:

an abstract method have abstract class.

```
Abstract class vehicle {
    int no of tyres;      यहां पे कुछ भी details नहीं है, बस method और variable create करते हैं
abstract void start();
}

class car extends vehicle {
    void start() {
    S.O.P ("starts with key")
}
class ~~car~~ bike extends vehicle {
    void start() {
    S.O.P ("starts with kick")
    }
}
p s v m (string[] args) {
car c = new car();
c.start();
Scooter s = new scooter();
s.start();
```

yourajson

• If a regular class extends an abstract class, then the class must have to implements all the abstract method of parent class, or it has to be declare abstract as well

Interface

Interface are Similar to abstract class but having all the methods of abstract type.

1) Use to achieve full abstraction
2) It support multiple Inheritance
3) Use to achieve loose coupling
   (एक के अन्दर change करेंगे तो दूसरे पर change नहीं होगा)

~~Inte~~
Method जो होगा Interface के abstract public method();
variable होगा तो
Public static final int a=10;

8th version.

→ default concrete method() we can create

→ static methods ↓

```
default void display() {
}
static void run() { }
```

9th version.

private method() we can also create

Syntax

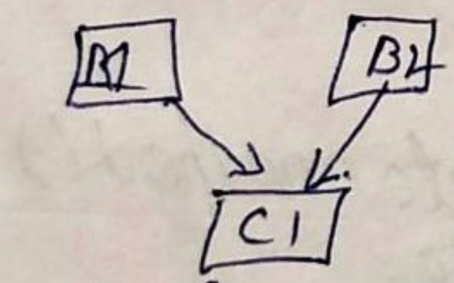
```
⇒ Interface Demo {

}
```

We can not create object of Interface.

Encapsulation.
wrapping up of data and information under a single unit.
We use private variable, getter & setter method.
. Data hiding where as abstraction है get detail hiding है।
extends या implement तो use करने child class के parent class के object create करने है।

Interface I1 {
public void show();
}
class test implement I1{
public void show(){
S.O.P("1")
P.S.V.m(String[]ar)
Test T = new Test();
T.show();
// 1
We can achieve multiple.
Inheritance - by Interface.



Interface I3 {
}
Interface I2 {
}
class Test Implement I1, I2{
}

This()
1. This() keyword use to refer current class instance varieb
2. current class method
3. current class constructor.
4. This can be used to pass an argument in the method call.
5. pass argument in constructor.
6. return current class instan
ce from the method.

Super()
. Super() keyword refer imedia
class - parent class instance variable.
Stub - parent class method
parent class constructor.

...nal keyword.

If we create final variable it become constant, we cannot change the value of final variable.

2). If we cc final method we can not override it.

3) final class we can't extends it or inherit it.

Static keyword ()

Only use for class veriable not for local variable.

static method ()

static block ()

static veriable belong to class not object.

simple variable belong to object.

create standard.

Abstract class	Interface
Abstract class can have non final non static variable.	variable declare with in an interface are always static and final.
Abstract class can have abstract method as well as normal methods	Interfaces can have only method declaration (abstract method). You can not define a normal method.
Abstract class is extends by another class using "extends" keywork.	An Interface implemented by a Java class using implements keywords.
An abstract class can extends another class and it can implement or or more interfaces.	An interface can extends one or more interfaces but can not be extends a class. It can not implement an interfaces.
An abstract class can have constructor define within it.	You can't define constructor within an interface.
An abstract class can't be instantiated using new keyword. Abstract class can not have object	An interface can not be instantiated.

Arraylist	Linked List
Arraylist internally uses dynamic array to store the elements.	LL internally uses a doubly linked list to store the elements.
Manipulation with Arraylist is slow because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory.	Manipulation with Linked list is faster than Arraylist because it uses a doubly linked list. So no bit shifting is required in memory
3. An arraylist class can act as a list only because it implements List only.	Linked list class can act as a list and queue both because it implement List and Deque interface.
4. Arraylist is better for storing and accessing data.	Linked list is better for manipulating data.
5. The memory location for the elements of an Arraylist is contiguous.	The location for the elements of a LL is not contagious.
6. Default capacity of 10 is assigned to the Arraylist.	There is no case of default capacity in a Linked list. In LL an empty list is created when a LL is initialized.
An arraylist is a resizable array	LL implements the doubly LL of the list interface.

Array & ArrayList

Similarities ⇒
- Array & Arradist both are used for storing elements.
- Array & Arraylist both can store null values.
- They can have duplicate value.
- They do not preserve order of element.

Object ⇒ Object is an actual entity to which a user interacts whereas class is just the blueprint for the object.

Array	Arraylist
An array is a dynamic-created object. It serves as container that holds the constant number of value of the same type. It has a contagious memory allocation.	The arraylist is a class of Java collection framework. It contains popular class like Vector, Hash Table,
Array is static in size	Arraylist is Dynamic in size
An array is fixed-length data structure	Arraylist is variable-length data structure. It can resize itself
It performs fast in comparison to Arraylist. because of fixed size.	Arraylist is internally backed by the array in Java. the resize operation in Arraylist slow down the performance.
Array can store both object and primitives type.	We can not store primitive type in Arraylist. It automatically converts primitive type to object.
Array can be multi dimensional	Arraylist is always single-dimensional.

What is MVC Architecture in Java.

⇒ MVC architecture separate the application logic from the user interface when designing software.

Model ⇒ Represents the business layer of the application.

View ⇒ Define the presentation of the application.

Controller ⇒ Manages the flow of the application.

Advantage of MVC Architecture

⇒ Multiple developer can work with three layers.

⇒ Offer improved scalability

- As component have a low dependency on each other, they are easy to maintain
- A model can be reused by multiple view which provides reusability of code.
- adoption of MVC makes an application more expressive and easy to understand.
- Extending and testing of the application become easy.

Enums

An Enums is special class that represents a group of constants.

enum level { low, Medium, High }

level myVar = level.Medium

advantage of Enums

- Type Safety :- Enum provide compile time type safety and prevent from comparing constants in different enums.
- group things in a set.
- Enums are iterable.

Global variable	Local variable
Global variable are declare outside all the function block.	Local variable are declared with in a function block.
The scope remains throughout the program.	The scope is limited and remain within the function only in which they are declared.
Any change in global variable affects the whole program wherever it is being used.	Any change in the global variable does not affect other function of the program.
A global variable exists in the program for the entire time the program is executed.	A local variable is created when the function is executed and once the execution is finished the variable is destroyed.
It can be accessed throughout the program by all the function present in the program.	It can only be accessed by the function statements in which it is declared and not by the other function.
If the global variable is not initialized it take 0 value by default.	If the local variable is not initialized. It takes the garbage value by default.

EXCEPTION IN JAVA

Exception handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained

Types of JAVA Exception.

1. checked Exception
2. Unchecked Exception.

checked Exception ⇒ The class that directly inherit the Throwable class except Runtime Exception and Error is known as checked Exception.

Unchecked Exception ⇒ The class that inherit the Runtime Exception is known as Unchecked Exception.

Try ⇒ A block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.

Catch ⇒ The catch block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. handle the exception by declaring the type of exception within the parameter.

Throw ⇒ throw is used to throw an exception.

Throws ⇒ Is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used in method Signature.

Finally block ⇒ used to execute important code such as closing the connection.

- Finally block is always execute whether an exception is handle or not.
- important statements to be printed can be placed in the finally block.

Custom Exception ⇒ To catch and provide specific treatment to a subset of existing Java exception. Business logic exception. These are exception related to business logic and workflow. It is useful for the the application user

- each catch block must contain a different exception handler.

Throw	Throws
Java throw keyword is used to throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code
Types of exception using throw keyword, we can only propagate unchecked exception i.e the checked exception can not be propagate using throw only.	Using throws keyword, we can declare both checked and unchecked exception. the throws keyword can be used to propagate checked exception only.
The throw keyword is followed by an instance of exception to thrown.	The throws keyword is followed by class names of exception to be thrown.
throw is used within the method.	throws is used with the method signature.
We are allowed to throw only one exception at a time i.e we cannot throw multiple exception	We can declare multiple exception using throws keywords that can be thrown by the method For example. IOException, SQL Exception

Final	Finally	Finalize
Final is the keyword and access modifier which is used to apply restriction on class, method or variable.	Finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.	Finalize is the method in Java which is used to perform clean up processing just before object is garbage collected.
Final is used with the classes, methods and variable.	Finally block is always related to the try and catch block in exception handling	Finalize() method is used with the objects.
(1) Once declared, final variable becomes constant and can not be modified. (2) final method can not be overridden by sub class. final class can not be inheritance.	(1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block.	Finalize method perform the cleaning activities with respect to the object before its destruction.