Core-Java

Core java - OOP

① java was developed by "Alan kay" → small talk
② reusability → inheritance.
③ Ada → supports → classes but does not
④ static member function cannot be used for polymorphism b/c it is not the property of object

* **object based**
supports classes but not polymorphism and inheritance.

⑤ Encapsulatn:- code and data together; can be achieved using access specifiers
⑥ size of class :- object size defines size of class, classes do not have sizes.
⑦ encapsulation → it is compulsory that member function operate on class data members only.
⑧ object :- can be passed as value/reference/copy but not as function.
⑨ platform independance :- is not feature of oop its feature of languag.
⑩ abstraction :- converting real world objects in term of class & object or interface
⑪ associatn :- if an object has its own life cycle & there is no owner it is association ↳ has-a relatn.
⑫, ⑬ composition :- where child is killed when parent gets killed
↳ part-of relatn

⑭ loose coupeling :- is achieved by using interface

53** String handling (it is class) string class

26-10-20

⑮ java.lang ~~object~~ → java.lang.string.
⑯ string → immutable
string builder } mutable
string buffer }
⑰ String Constructor → → 13 constructors
string (char[])
string (int[], i, 3)
⑱ getchars(6, 15, ch, 0) (parray) always 15 - 1 = 14
"0123456789012345678" → 6789012345
i/p o/p

⑫ String function

bool matches(String regex) → tell weather string contains Reg Ex

regionMatches( ) → tests if two string regions are equal.

⑧ startsWith( )

④ compareTo → return an int
+Ve → बाद में आ
-Ve → String object precedes argument string (~~बाद में~~ पहले)
0 →

"abc".compareTo("bcd") →
a b पहले → -ve ✓ checked
बाद → +ve

a. compareto(b) → -Ve
c. compareTo(a) → +Ve

⑧ S_1 = "hello", S_2 = "hello" → both are literals and both will be stored at same locath

$S_1 == S_2$ → True
S_1.equals S_2 → True

String builder

SB S_1 = new SB("hello")
SB S_2 = ~~new SB(~~ S_1.append("123")
$S_1 == S_2$ → true, S_1.equals S_2 = true

⑧⑧ do not override equal method from object

→ trim( ) → is used to remove white spaces from lead and trail
→ replace( ) → replace(old, new) → all occurances are replaced

⑧ To print "Hello" → "\"Hello\""

b → \ → "\\" → \
' ' → "\" " → 'hello'

} print " " (' ' on Console /

→ sub string(start2, end2)
inclusive, exclusive

0 1 2 3 4 5 6 7 8 9
2 3 ← (2, 4)
= o/p

⑧⑧ delete(startInd, endIndex)
inclusive, exclusive → always up end2 - 1

replace(start, end, string)
→ exclusive.

String Buffer

Valid constructor → SB(), SB(int capacity), SB(obj), SB(String)

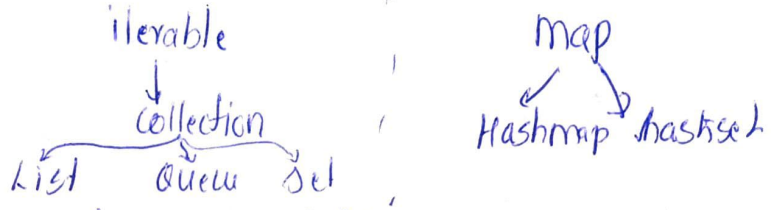
length → no of chars

IndexOf() → returns '-1' when not found in string.

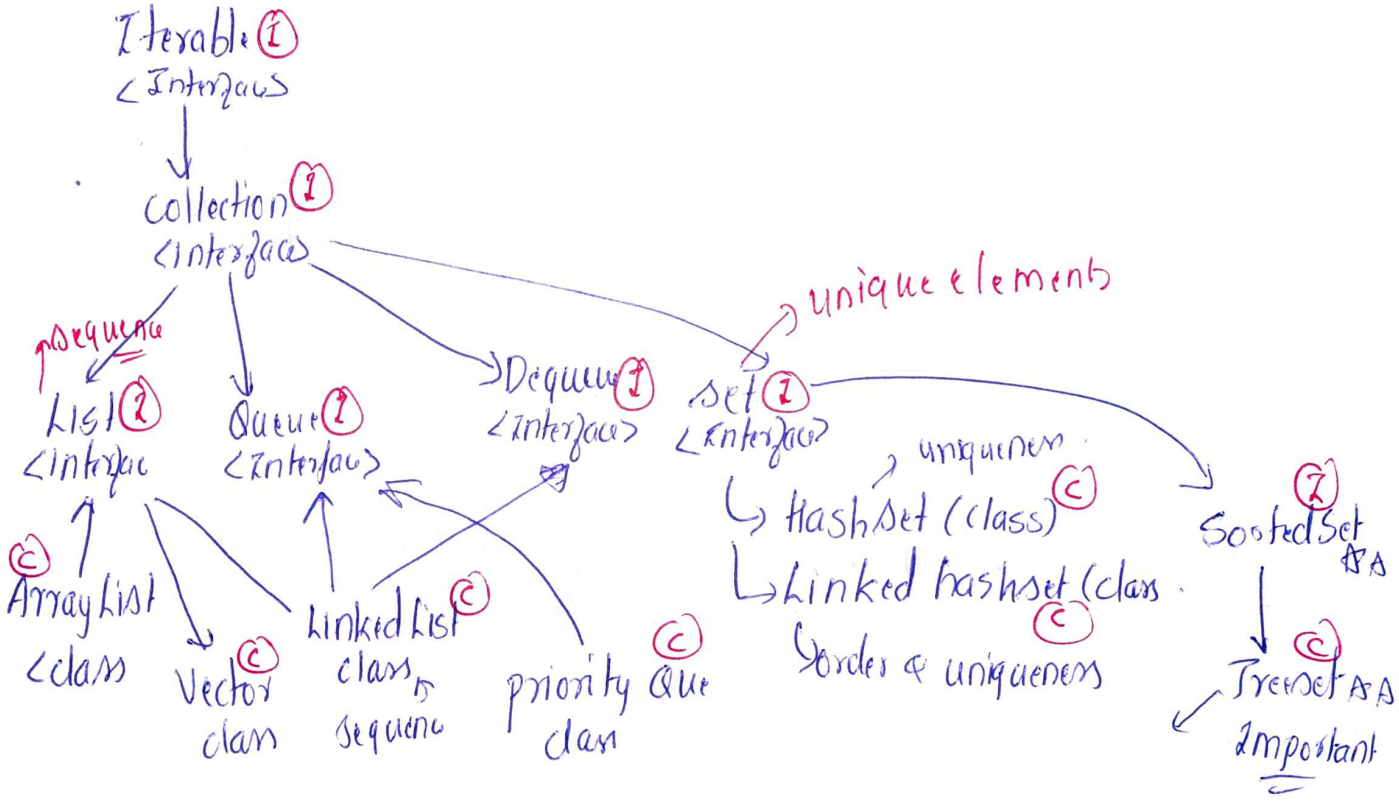
⊗ String buffer & String cannot be compared

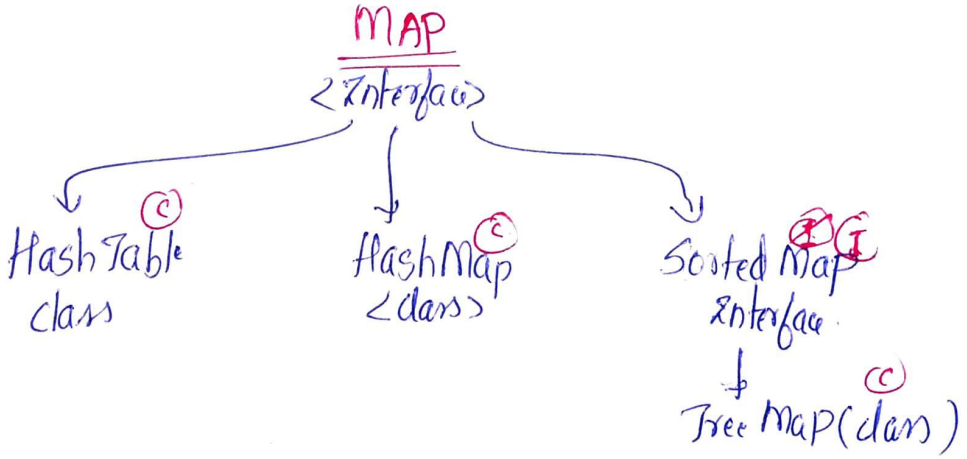
Collections.

java.util →



⊗ collection and map both are different, map is not collection





(4)

Methods

Collection
Void clear() → removes all elements from this collection

Arrays.fill(array, start, end, value)
inclusive ↓ exclusive ↓

Iterator → ArrayList AL = new AL()
iterator<Integer> itr = AL.iterator();
↳ type of AL

List iterator → can be used with only List
⇒ method used to get previous index = previousIndex.
→ Exception thrown by remove() → IllegalStateExcepⁿ

③ Iterator → ~~hasNext~~ Boolean hasNext()
next element next() → used to access next element (has next goes to next)
remove →
initially before element → complete loop → after last
↳ NoSuchElement found Exceptn.

117
List iterator
previousIndex() → returns index of previous element, if no element is present return '-1'

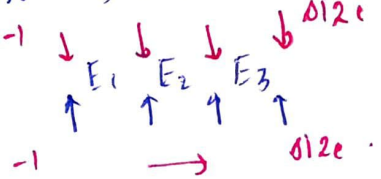
④ both are fail-fast → throws Concurrent Modificatn Exceptn
Concurrent Modificatn Exceptn.

Limitatn of remove() :- it can be called only one per next().
↳ illegal state exception.

16

List-iterator → both direction travel.

117



as next is call has next also called

List interface → ordered collection (sequence)
SET interface → no duplicate elements.

146 Map

- does not inherit collection interface
- permits null keys and values
- unsorted, unordered
- if key which already exist is put than earlier content is over written
- clear() → To remove all entries. - get()

→ emps.put(e) → k.hashcode → equals
- True → duplicate
- false → unique

- The key is hashed twice first → hashcode of object, second by internal hashing method of hasmap class.
- thread unsafe. throws Concurrent modificatn Exception

Treemap → sorted as per natural sorting ascending

Array List (throws Concurrent Modificatn Excepn)

1. dynamic Datastructure
2. public void ensureCapacity(int minCapacity)
   → to ensure min capacity
3. public int size() → used to calculate number of elements in list
   - it does not give size.
   - it tells how many elements are present
4. public <T> T[] toArray(T[] a) → used to convert to static array.
- public void trimToSize(); → used to trim to no of elements. remove extra space
- Void add(int index, Element);
  → pos
- boolean remove(int i) boolean remove(Object)

List

- To remove duplicate elements → convert to hashset.

6

- Arrays.AsasList( ); → fixed length List, does not allow add or remove,
- ArrayList Implements → "Random Access" interface
- AL is alway pass by reference

Hash Set (ConcurrentModificatn Exception)

Initial capacity & load factor
"(16)" 0.75

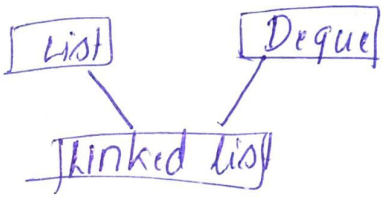
- hashSET internally implements hashMAP
- ensures no duplicacy
- boolean contain(Object o);

Set
↳ linked hashset
↳ hashset
Sorted Set Interface
↳ Treeset
ascending order

Linked Hashset → guarantees uniqueness & insertion order
↳ add(e)

Linked List

→ addfirst → add at start
→ removeLast( ); not delete Last
→ set( ) → to change the object.



Hashmap

① keySet( ) → returns a set containing all the keys used in map / set view
② Values( ) → "collection view"
③ set( ) not ~~insert~~ update,
clear( ) not removeAll
put( ) → insert

- max(collectn, comparator) → collections method
- fill( )

Singleton List / Map (unsupported Operation Exception)

- we to convert single object to list / map
  ↳ immutable list, cannot add or remove
  ↳ always only one element.

unmodifiableCollection() is only for List and set.

⊗ (EMPTY_LIST, EMPTY_SET, EMPY_MAP.) → static variable of collections.

⊗ fail-fast → list, map iterator → ConcurrentModificatnException.
fail-safe → CopyonWriteAL(), concurrent HashMap, ~~vector~~.
↳ works on copy ∫ slow (not set)
works on original list.

⊗ Accuracy and efficiency of hashmap → equals and hashcode

⊗

Comparable → compareTo( ) → natural sorting
Comparator → compare( ) → Custom sorting.

Vector → standard size is 10 capacity ↑ increament ∈ 100%.
→ 10 —— 〃 —— 50%.
ArrayList → Random Access
→ cloneable

List Interface :- allows storage of null values.
Vector → it extends Abstract List
↳ it is "fail-~~safe~~ fast" → iterator returned is fail-fast.
TreeSet → does not allow null values
preference for multithreaded env → CopyOnWriteArrayList → List
Natural order of key → ConcurrentSkipListMap → sorted
⊗ Set → ConcurrentSkipListSet → sorted & synchronised
map → ConcurrentHashMap

Which to prefer

performance → HashMap
frequent add & remove → LinkedList
more searching → ArrayList

fail-fast → ArrayList, LinkedList, Hashmap, LinkedHashmap, TreeMap,
(moving memory)
fail-safe → CopyOnWriteArrayList, ConcurrentHashMap,
↳ work on done ConcurrentSkipListMap.

Collection	Ordered	Random access	Key value	Duplicate elements	Null	Thread safety	Initial capa	Sorted	Add/get
1) ArrayList (L)	✓	✓	✗	✓	✓	✗	size-0 Cap-10	✗	O(1)
2) LinkedList (L)	✓	✗	✗	✓	✓	✗	✗ Cap-10	✗	O(n)
3) HashSet (S)	✗	✗	✗	✗	✓	✗	cap-16	✗	
4) TreeSet (S)	✗	✗	✗	✗	✗	✗	✗	✓	
5) Hashmap	✗	✓	✓	✗	✓	✗	load-0.75 Cap-16	✗	
6) Tree map	✓	✓	✓	✗	✗	✗		✓	
Collection summary: Vector (L)	✓	✓	✗	✓	✓	✓	s=0 Cap-10	✗	
Hashtable	✗	✓	✓	✗	✗	✓		✗	
properties	✗	✓	✓	✗	✗	✓		✗	
stack	✓	✗	✗	✓	✓	✓		✗	
CopyOnWriteArrayList	✓	✓	✗	✓	✓	✓		✗	
Concurrent hashmap	✗	✓	✓	✗	✗	✓		✗	
CopyOnWriteArraySet	✗	✗	✗	✗	✓	✓		✗	
Linked Hashset	✓	✗	✗	✓	✓	✗	16	✗	
Linked hashmap	✓	✗	✓	✓	only one ✓	✗	16	✗	

Treeset :- internally uses Treemap to store elements

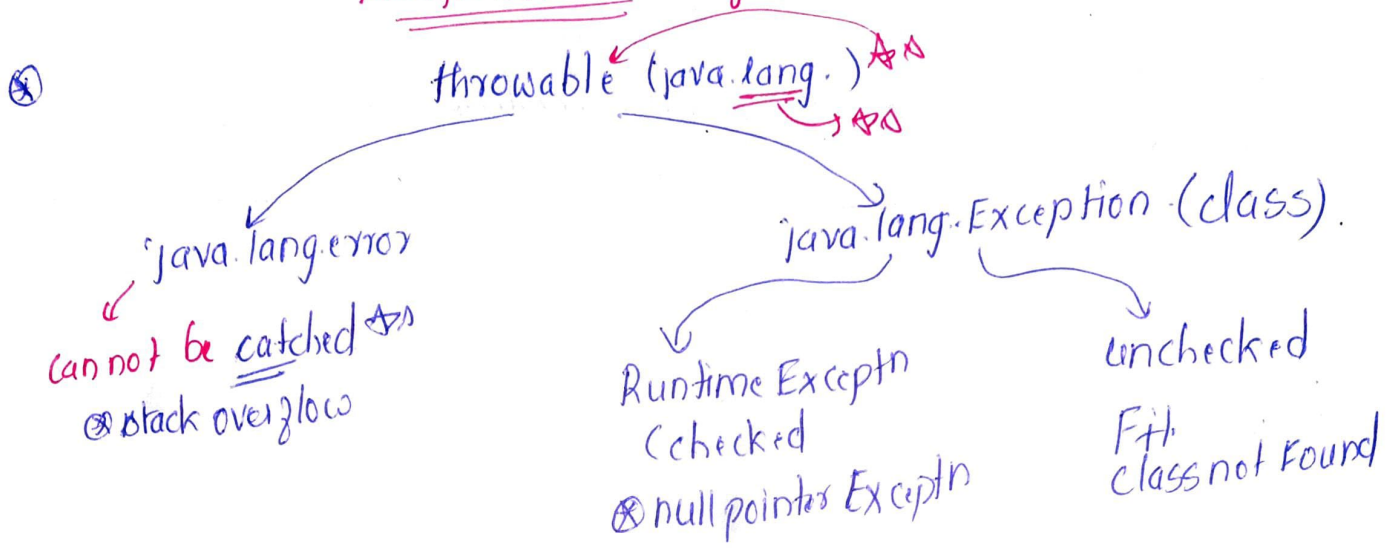
In case where sorting is used → no null values are allowed

Null key and value — Set → unique elements

LinkedHashmap → one null key & many null values.

Hashtable → No null is allowed

Exception Handling



→ File not Found Exception → checked, found in I/O package,

IO Exception → " " → .

```
try(){
    %0
  }
catch(){
    syso(A)
  }
finally{
    syso(B)
  }
```

⇒ AB

In Absence of try-catch default handler comes in picture

Variables declared inside try block are not accessible outside

parseint Exception → Number Format Exception .

Try must be compulsory follow catch

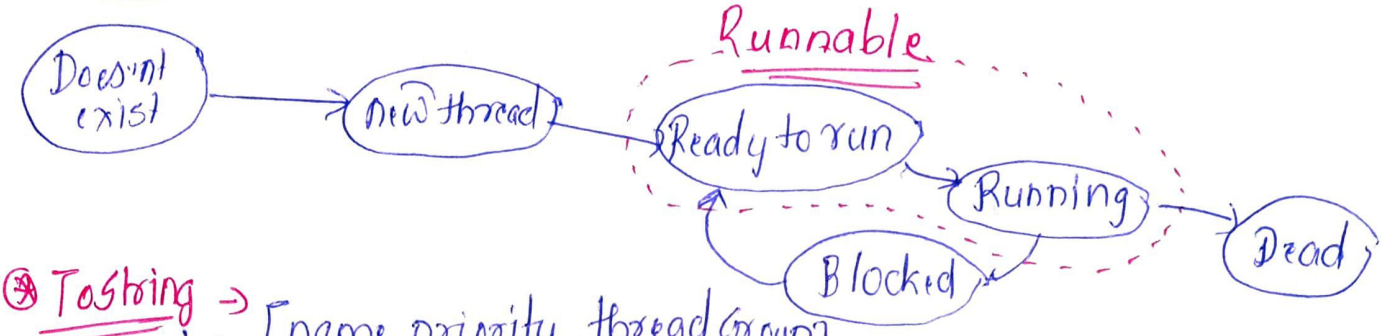
10

Exceptions In Java

Name	caused when
① IllegalThreadException	Try to restart dead thread, or runing thread. restarted -
② java.lang.InterruptedException	To unblock sleep, join, wait
③ illegalmoniterstate Excepin	If the thread is not owner of object monitor
④ illegal state Excepin	itr.remove twice after next()
⑤ illegal modificatn Excep	Itr modifies structusally while executing
⑥ illegal State Exception	bifor sending request printwriter flush...

⊗ priorities → are integers (min_priority
(Norm_)
(max_)

⊗ Thread states

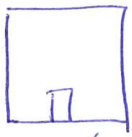


⊗ Tostring → [name, priority, thread Group].
Thread[one, 5, main]

---

keyword for synchonisatn → synchronised (Block level, method level)

notify()
it wakes up this thread waiting outside monitor. (single)



→ thread waiting out side monitor

both notify & wait compulsory with synchronised ✱✱.

To suspend monitor lock and wait

wait() ⇒ causes the current thread (on which this method is called) until it is awakened, by notified or interrupted (by other thread by (interrupt signal).

after notify compulsory running state ✱✱

Implement vs extend

extends	Implements
my class extends Thread { }	Runnable my class implements ~~thread~~ { }
⊗ a my class is a thread ⊗ Const thread(name) ⊗ all thread api allowed	my class is Runnable not thread thread(Runnable target, string name). not allowed

---

JVM termination :- only after all user threads are over, does not care about daemon thread.

thread priority :- it always decided by by thread scheduler & OS

thread Constructor Thread(name), Thread(Runable, name)
(ThreadGroup, —,, — .)

12

* If we dont give name to the thread automatically group name is assigned as name (e.g main).

Thread Api

getThread priority → getPriority().

run()

start()

Min-priority → 1

Max → 10

Always → main run as parent thread and other act as child thread

* daemon thread always runs in background
* setDaemon(boolean) :- followed by start (compulsory)
* priority of child cannot greater than parent
* if child process fails then complete process fails
* this cannot be used in static methods.
* sleep, yeild, stop, start

* Best Example of overloading

```
Void inorder(){          |  Void inorder(Node cur){
    ↗                    |
}                        |  }
```

as root is private not accessible outside to start the Recursion

Overriding

→ websecurityAdapter / object class /

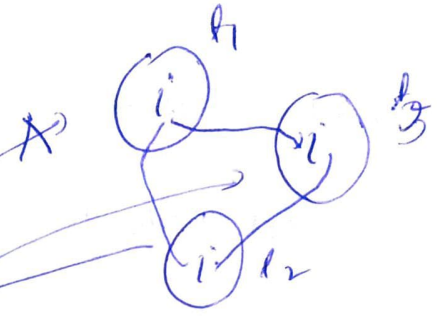
* Inheritance → dao extends, Spring Security
* Encapsulatn → linked list, head root, Node class
* polymorphism → List returning
  - Run time → Response entity, List, Object.
  - static → linked list calling method

Core java

Code of magic number

Island of Isolatn :-

```
class Test
{  Test i;
     main(){
       Test t1 =
       Test t2 =
       Test t3
```




```
t1.i = t2;
t2.i = t3
t3.i = t1
```

→ t1 = null, t2 = null, t3 = null → Complete island eligible to Garbage Collectn

Eligibility for Garbage Collection?

① remove all reference to that object

String s1 = "deepak"; → eligibl.

s1 = null

② s1 = "deepak", s2 = "Ranjan" ✓eligibl.

s1 = s2;

③ Island of isolation

Q) o/p of sysо('b' + 'c' + 'i') → sum of ascii values will be printed as " " is not used hence concatinatn not used.

→ Double brace initialisatn.. used to initialis object using anonymous inner class

```
List<Integer> list = new ArrayList<>(){
                        {
                          add(12); add(13); add(14);
                        }
                    };
```

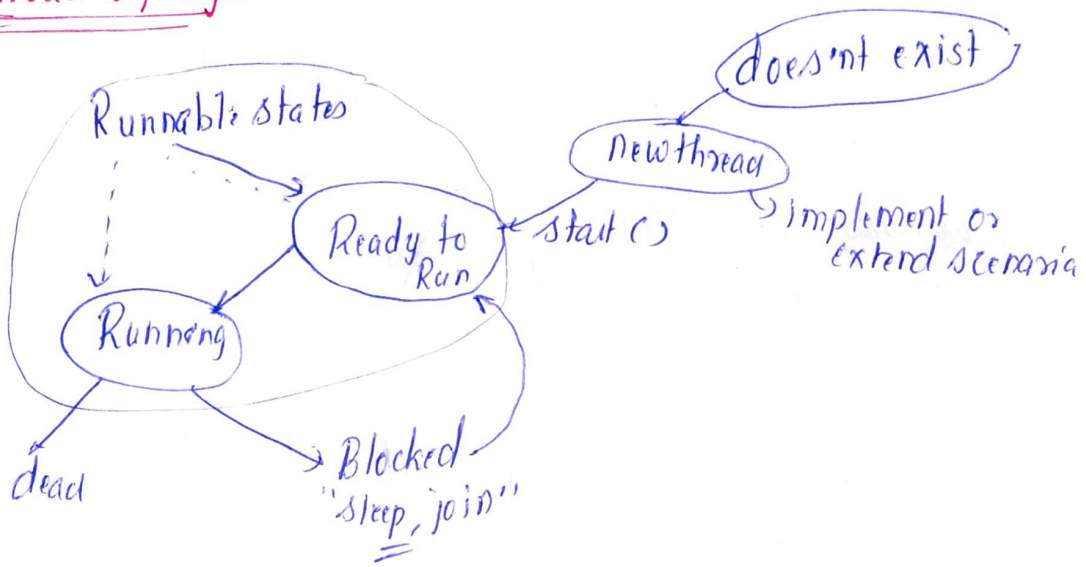
→ marker interface or tagged interface : An interface that has no members e.g Serialisable, clonable, this is to provide some information for "JVM"

Q6 will finally execute after System.exit(); is written at end of try block?
no, program is terminated after System.exit(), hence finally block never executed

⊗ pa will importing parent packages will also import child class as well
No, it will import only parent classes.

⊗ we can load ~~one~~ import one class twice but it is reduclant as jvm loads for one class only.

Thread life cycle



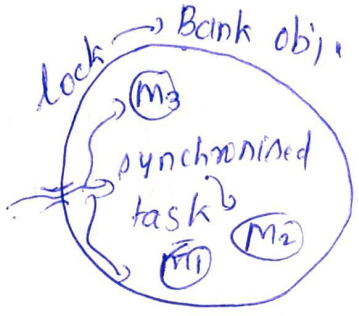
Variable Args :- fooBarMethod("foo", "bar"), fooBarMethod("foo", "bar", "Goo") all will work for
public void fooBarMethod(String ... Variables)

---

why is synchronised necessary? Example is necessary?
Critical section :- a section of code that leads to race condition
e.g hospital_management database, update, fetch "api"
when required → it is required if multiple threads are sharing same resource (common) Resource (any db table, socket, reservation) & one thread is accessing and other is modifying.
keyword → synchronised → method level or block level
↳ applies lock at object level (i.e instance of shared resource.
" joint account

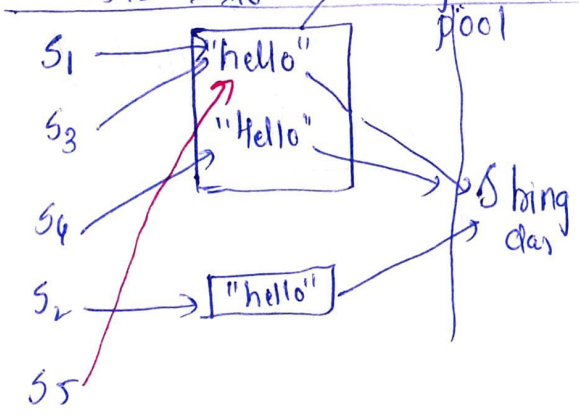
→ Single thread can apply multiple locks. e.g when insertion a patient lock is acquired on user, patient, ward table.

- Blocking trigger → when thread not able to acquire lock it enters monitor (locked on monitor)
- unblocking → when lock is released or synchronized block over.
- sleep inside monitor :- If thread invokes sleep it sleeps with all locks, (i.e user, patient table)
- method level or block level



How is new String and literal are different?

```
S1 = "hello"
S2 = new String("S1");
S3 = "hello"
S4 = "Hello"
S5 = "hello"
```



S1 = S2 = S3 = S4 = null (only S2 GC)

S5 = Hel "hello".

what is associatn?

relationship between seperate classes. using Object references. Has-A-relatn.

e.g patient service has ward Dao, Medicine Dao, Doctor Dao

Association

Composition	Aggregatn
part of or belongs to	child can exist independently of parent
child cannot live without parent	Service has patient Dao, ward Dao
e.g user entity has employee, employee has doctor	
Strong	weak

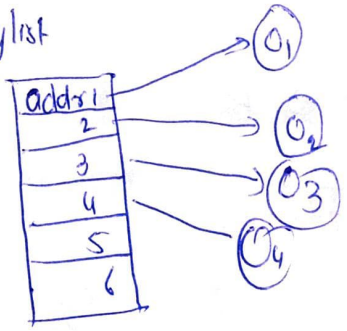
Why association preffered over Inheritance

To overcome multiple inheritance, we can write as private member of class

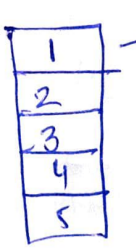
unit testing is not possible in heritance

Contiguous memory locatn are used in array whereas in arraylist it is not contiguous

Arraylist



only references are kept

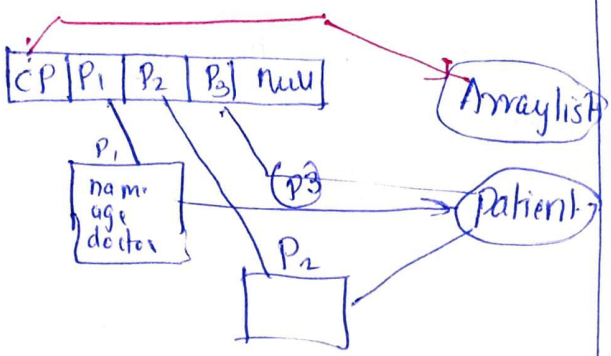


→ actual primitive data is kept

Internal working of Arraylist

```
Arraylist<Patient> patients
patients.add(P1)
        add(P2)
```

Account



Constructor chaining

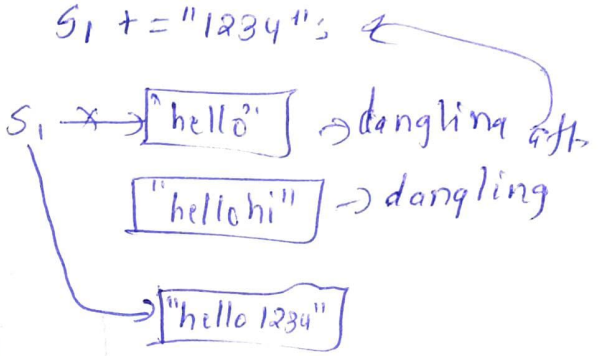
```
public class InterviewBit {
    InterviewBit() {
        this("String")
    }
    InterviewBit(String str) {
        Sys o(str)
    }
```

How To avoid Serialisatn of some fields

Transient key word

for updates which to prefer String or String Buffer.
String Buffer preffered because on every operatn new object of string is created hence many object gets collected

```
String S1 = new String("hello");
S1.concat("hi")
S1 += "1234";
```



finally → will be executed under every circumstance except System.exit()

```
try {
    return
}
catch {
    return
}
finally { }
```

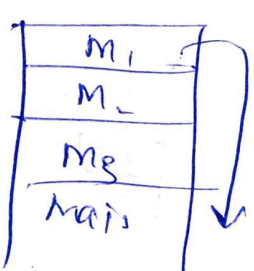
Pass by Value & pass by reference

* java is always pass by value but when object passed to method its address is passed hence changes are reflected in original code

exception propogation

If try{} catch present then resp. catch is executed or else

Or else except is propogated to main delegated



What are difference b/w constructor and method

Constructor	method
① same name as that of class	may have different name
② called implecitly	called explicitly
③ if not declared compiler provides	no such provision we have to declare method
④ cannot be final b/c when	can be final
⑤ no return type	may or may not have return type

Ways of creating thread.

① Extends scenario :- we extend directly

extends	Implements
myClass extends Thread { };	class myRun Implements Runnable { };
* myClass is a thread	myRun is not thread
⊛ directly uses thread api	to use thread private Thread t;
const used Thread (str name)	Thread (Runnable, name)
Start()	e.g t = new Thread (task, "t"); to start.
must override public void run()	public void run()

What is importance of Reflectn in java?

⊛ It provides ability to inspect and modify the run time behaviour of applicatn.

⊛ reflection is of little use 4 programmers but its a backbone for Java, J2EE, framewors, IDEs, debuggers.

Hashmap VS HashTable

Hashmap	hash table
⊛ not synchronized	its synchronised
⊛ allows only one null key and any number of nulls	⊛ it does not allow nulls in keys as well as values.
⊛ subclass LinkedHM support order	no order of insertion maintains

Confidential info → charaacter ✓ clear.
not string ✗ → remains in pool

unstopped

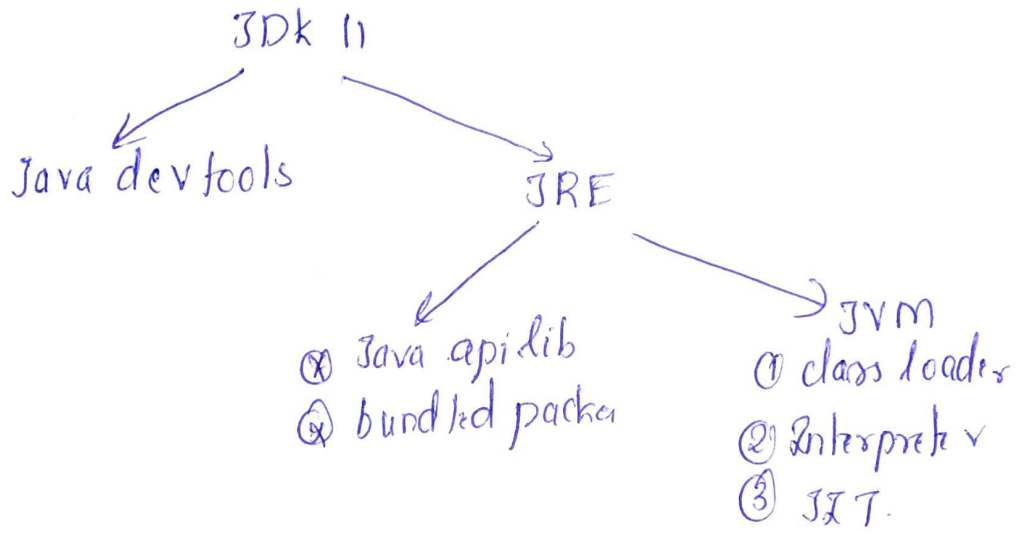
How to execute if and Else both if else both execute
In 'c' use fork() to child process

⊛ Treeset → all elements are stored in sorted manner

Hashset	Treeset
⊛ ~~sorted~~ not sorted	⊛ ~~not~~ sorted
⊛ constant time access	⊛ (olog n)
⊛ hashcode & equal for comparision	compareTo & Compar
⊛ null are ~~not~~ allowed	⊛ null are not allowed

50

JDK	JRE	JVM
Java development kit	Java runtime environment	Java Virtual machine.
e.g javac, java, javap, jar,	Java api library in Jar format jre/lib/rt.jar ① bundled packages. java.lang. java.util	① load and execute ② class loader → loads required classes in JVM's method area ③ ② Interpreter (Translator) ③ JIT → just in time Compiler



Static method and private method overriding

Static means belongs to class hence object may have same function with name, but they cannot be overridden

① private methods cannot be overridden

Abstract class	Interface
① methods may be abstract or non-abstract	all methods are abstract.
② 	
③ if not declared class as abstract	implementing all methods is compulsory
④ may be private, public, non-final	④ variables are by default final
⑤ may be private, public, protected	⑤ members are public by default
no instantiation constructor allowed	no instantiation no constructor
	static final variable allowed

String builder, String, String buffer

in string stringpool serves as storage area, for string builder, string buffer heap area is used.

mutability :- string is immutable where as builder & buffer are mutable

efficiency :- StringBuffer > Builder > String appending

Thread safe :- only string is thread safe

Why String Immutable?

① Security :- string is widly used datastructure ① urls ② database connections, ③ loading class. ④ store passwords ⑤ n/w connectn hence they are passed through diff methods for validatn parsing where it can be tempered

Synchronisatn :- inheritedly thread safe no external sync required

HashCode Caching :- # strings are abundantly used in hashmap implementatn here only once hashmap calculated & stored they reductn in huge time as hascode is frequently called while bucketing

optimation by Compiler :- as string is widly used DS

key of Collectn is string :-

Can static methods overridden?

No it is not allowed, overiding is runtime phenomena but static methods get thier bonds during compile time.

⑧ if we override then who is calling his method will be called

```
Parent p = new child()
p.display() → will call parent method
p.display() → child will be called
```

Can static method be overloaded

yes it can be overload with different signatures

Readm.

→ as method overriding is related to type of object at runtime as static has nothing to do with object → no object no overriding

When can we use super keyword?

To access public and protected methods & data of parent class.

② to initialise parent constructor

③ it is used when parent and child have same named field using this/super

④ finalized, final, finally.

finalized() → is protected method garbage collector which GC called on an object before collecting that object

```
e.g main(){
    String bd= "krushna"
        be = null
        System.gc()
    public void finalize(){
    }
```

finally :- it is block in try ladder which gets compulsorily called under any circumstances

```
try(){
}
catch(){ }
finally{
}
```

final :- it is const of java

it in class, variable, method level. overriding is restricated by these key word

---

all methods are virtual

static, final, private no overriding

final on →

Variable → values cannot be changed,

Methods → cannot be overridden

class → cannot inherited

Multiple try catch

```
try{
    %0
}
catch (divideByZero)
catch (Arithmatic)
catch (Exception)
```

---

method overriding or Overloading

```
Dfs()
Dfs(Node root)      } → overloading
```

```
List getPatientDetails(){
        Return ArrayList;
}
```

Runtime polymorphism

```
ResponsEntity getPatient{
        Return Respons.
}
```

overloading → can be in same class or subclass

① same method Name.

② different signature

③ ignored → ret type

Overriding

```
List a = new arraylist();
a.add()
```

→ method of AL is call.

overridding

① same name, same signature
② ret type → same or subtype
③ scope → same or wider

*** Javac → resolves by reference.
** JVM → resolves by object type it is reffering to

Object class methods

equals → does reference equality not content equality for that we overrid equals.
Tostring → To get detail of class.

Why constructor in abstract class?
To initialise its private members

e.g Emp → empid, dept private
Mgr → to init → require const

== → reference equality, operator
equals → content equality, object class

JIT Compiler

as 80% of execution time is spent on 20% of code, and JIT compiler works on this 20% code, i.e it statistically finds this code and save it in native code readable directly by 'OS' hence, lot of optimisation done by JIT

* this is done parallel running threads
* smallest Compilation unit is method

Instance Variable ≠ ~~method~~ Variable
↓ ↓
Variables of class Variable of class
Constructor, local
In scope

MCQ

① primitive type → 8 primitive int, char boolean, byte, long, float, short, double
② float → 4
double → 8.

byte → short → int → long → float → double

char → int

long → float ✓ accepted

Type Conversion in arithmatic

byte short → int
long float → float
byte short → int
float double → double

* byte → 8, short → 16 bit, int → 32
long → 64, float → 32 bit
double → 8 byte
char → 16 bit, boolean → 1 bit

-128 to 127 (i++)
cyclic nature of byte data type

* char ch = new char[5] —
* int[] A = {1, 2, 33} ✓ (81 82)

Return type of Constructor → class object

Constructor → public Solution()

Infinite loop → for(;;) A-1
for(int i=0; i<1; i--)
for(int i=0; ; i++)

System class defined in java.lang.package

Notebook imps

Legal access specifier for class → public & default

- when we have public class B { } here file name & class name should match ∴ one public class
- java unitialised variables not allowed
- JJJ → if sysO → 100 times (hotspot) JIT → compiles it once and uses 'qq' times
- if we start single thread twice than illegal thread start except occurs.
- yeild() → Running to Ready to run
- Run → Block → Ready.
- When we try to start dead thread → illegal thread state exc
- Terminatn of JVM :- When all Daemon/user threads are over
- join() throws interrupted Exc
- Interrupt signal cannot Interrupt blocked on I/O sleep, join, wait
- synchronisatn at method may be harmfull hence block level preffered
- wait → release cpu and monitor and monitor owner wait outside
- notify → unblock one wait
- notifyall → all wait are unblocked

Daemon	user
internal to JVM	thread created by user main()
e.g Service thread Garbage Collector	

- JVM terminates after all user threads are over
- Collection
  - array list is internally array
  - Random access interface
  - Initial content of list → iterator over size '0'

Explicit iterator

it returns a cursor which can go in forward directn

- hasNext() returns true if there is next element
- Next() → To access next element
- Iterator itr = list.iterator()

- iterator goes over size and not capacity
- Collection → interface
- Collections → class to manipulate DS
- Collection framework → implementing DS
- Arraylist → size, get, set, iterator
  list, iterator,
- We can create AL from collection
  → one next → has next
  Returns
- whenever iterator is attach no modifications allowed
- Itr.Remove() and next() go hand in hand (one R one N)

---

list iterator (-1) (size)

while(has.previous)
itr.previous } same has next

Sorting

limitation

Collection.sort(list)

Natural →
Comparable → compareTo()
Custom → Comparator() → compare()

Annonymous Inner class

```
Comparator <? super> c)
new Comparator<Vehicle(){

});
```

- in generics runtime polymorphism doesnot work
  - ? → unbounded wild card
    → equivalent to object

? Extend? III allowed

? super E II allowed
→ all super to E
→ all who extend T.

SET (no duplicates)

- unordered unsorted.
- Treeset is sorted
- only single null allowed

Java

Identifier may start $ or _

Method local variable → stack
uninitialised

Instance Variable → Initialised
object Variable → heap

static Variables → Method area.
class variable Initialised

Garbage Collection

System class

System.gc() → void ret

Object class Api

protected void finalize() throw Throwable → called before Object gets garbage collected

---

Static initialization block

{
//called only once
}

Static nested class

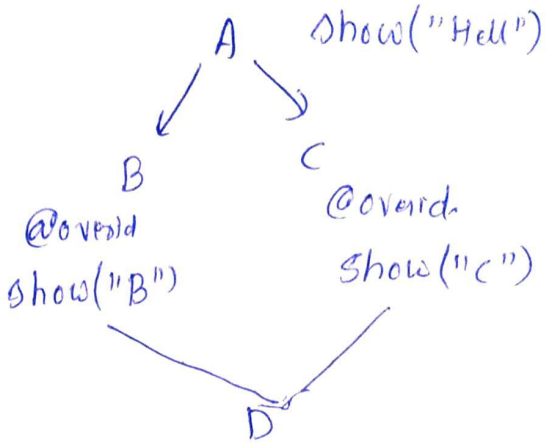
class outer {

static class Nested
{
//can access only
{ static member of outer
w/o ans.

Inheritance

→ methods → inherit
methods → overrid.
@overrid

Diamond Problem



```
class A {
  void show() {
    sop("in A");
  }
}
class B extends A {
  void show() {
    sop("B"
    super.show()
  }
}
B b1 = new B();
b1.show →
```

---

this() } → cannot both come in
super() } constructor

this., super. → used to access visible members

AAA An

* when compiler does not find exact match, tries to resolve it by closest avg type (just wider)

short → int
byte → short

equals → true → if both ref (address same). else false to same object.

(==) → address comparision

* to have content equality @override equals

→ Instance of

True → if is instance of

false → for null or not equal

final method → wait, notify, notifyAll

final class → String, StringBuffer, StringBuilder

protected → in same class default

↳ these member cannot be accessed from object

↳ only access through inheritance

---

non static inner class

* it can access private members of outer class
* cannot contain static members b/o this class created with outer object
* can have any access specifier
* from outer inner member not allowed (all members)

static inner class

* can access only static member of outer class.
* can have any type of member static or non static

? → any type

? Extend ↑↑

? super ↓↓

Collection framework

Collection ②

① Set ↓ SortedSet

List ③

Map ④ ↓ SortedMap ⑤

List → duplicate allowed, order is significant

add, get, remove set,

Set → no duplicate no order

↓

HashSet, TreeSet, LinkedHashSet

↓ unordered ↓ sorted ↓ ordered

Map :- no duplicate keys

→ put remove, clear()

Hashmap TreeMap "Sorted manner keys"

Hashmap → at last convert to TreeMap

Contract

ref1.equals(ref2) → True

↓↓

ref1.hashCode = ref2.hashCode

String, Number, Wrapper already Implemented ↑

Method reference

s → System.out.print(s)

existing method

System.out::println.

Streams

Source → Filter → Sort → Map → Collect

array.stream()
Collection.stream()
Collection.parallelStream()

Hot Seat

(Q) why one public class

→ 'java spe language specificat'

↳ one class allowed in java

pub

↳ pkg1 → file.java

(A) ~~Use if public~~

(A) To make compilation efficient/faster

JLS force to have public class name as file name

(Q) How can we pass argument as 'ref'

↳ In java all argument are passed by all.

↳ objects are passed by refs

↳ refs are passed by all

(A) pass by value

fu

fun(a, b, c)

int a, b, c : copied

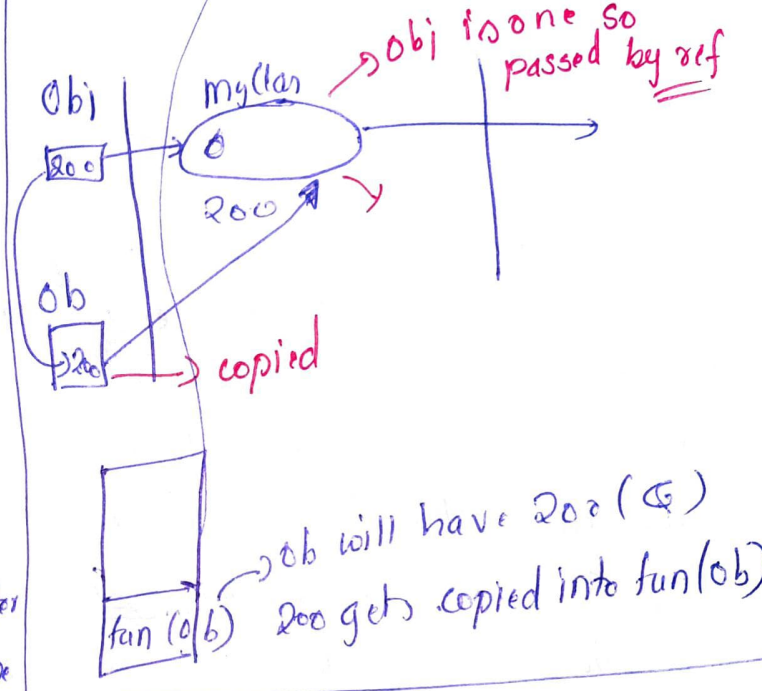
fun(int x, int y, int z){}

(A) myclass obj = new myClass()

fun(obj) → object pass by ref

fun(myClass obj){ ref passed as Value

}



Pass by ref

↳ idea → when pass by ref actual argument can be modified

↳ to achieve this effect → we can

↳ use wrapper class obj instead of primitive

↳ wrap obj ref in array & pass them.

e.g. object hence ref is passed & original values are swapped

```
void swap(int[] arr) {
    temp = arr[0]
    arr[0] = arr[1]
} arr[1] = t;
```

Q) adv & disad of Generics ?

↳ Advantage

↳ type safety. (avoid class cast exp)

eg

```
List list = new ArrayList(); list of obj
list.add("nilesh")
        ("nitin")
        ("Rahul")
        (1234)
for (String el : list) // element 1234
    syso(ele)          throw class
                       cast exception
```

```
List<String> list = new ArrayList<>();
```

In this case we get compiler error gives opportunity to improve at compilatn, because it is before running app, which is safe than failing at runtime. & crashing system may lead to failing of DB's & other resource

---

disadvance of generic

→ cannot instantial Generic type with primitive typ

```
List<int> list = new ArrayList<>()
```

→ cannot declare static field whose types are type parameter

```
static T obj; // error
```

→ Cannot use casts or instanceof which parameterized ty.

```
obj = (T) obj2
```

⊗ cannot create arrays of Parameterized type

```
T[] arr = new T[5].
```

⊗ Cannot overload method where the formal parameter types of each overload erase to the same Raw typ.

not allow

```
Void fun(Set<String> obj)   X
void fun(Set<Integer> obj)
```

Internal

at jvm level all collectn are still of object type.

⊗ List obj = new ArrayList()

⊗ List<String> obj = new AL()

⊗ type safety is implemented only at compiler level

⊗ Compiler removes param info type erasure

x - due to this int → obj X

⊗ Comparator & Comparable

↳ Comparable → compareTo(T other)

↳ Comparator → compare(T1, T2)

⊗ Comparable → natural sorting

Comparator → custom sorting

⊗ natural → defined by owner of class

e.g String --> Comparable → Asc

↑ comparable

⊗ Comparision logic is inside the class.

Comparator → customised sorting. decided by user of the class.

Array.sort(arr, (s1, s2) → s1.compareTo(s2))

- there can be multiple comparators depending on requirement
- Array of emp → sort on sal, sort on name, sort on dept,

Owner of class → where Comparable is implement in UDT

User of class → we use string class

- In industry some one defines class and mostly we use the class, and if that has resigned and we dont know what was logic so we extend and implement,

fail-fast fail safe

fail-fast

~~List<String> itr = list.iterator();~~

```
List<String> list = new ALC();

Iterator<String> itr = list.iterator();

while(itr.hasNext()) {
    String ele = itr.next();
    sysO(ele)
}
```

Assume two threads are working on same collectn list then it results in structural modificatn when one thread removes element. and since iterator fails giving exception ConcurrentModificatn (exp → itr.remove) this is due to iterator is already deciding traversing on original collectn

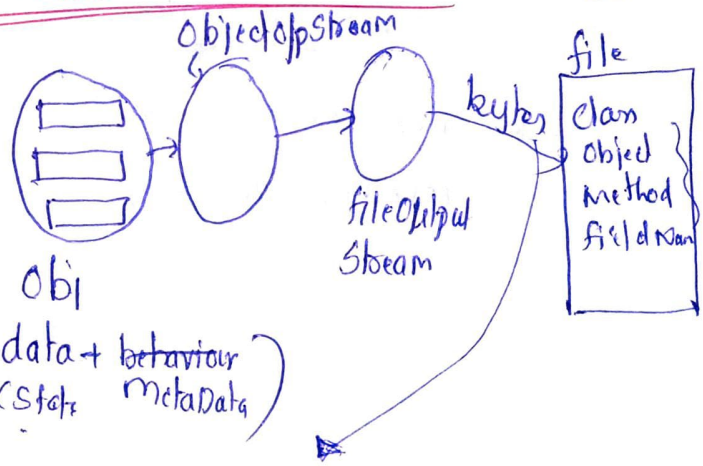
fail-safe

- iterator that works on copy of collection, even if collectn is modified no exceptn is raised
- failsafe iterator are available for specialised collection Concurrent Collection

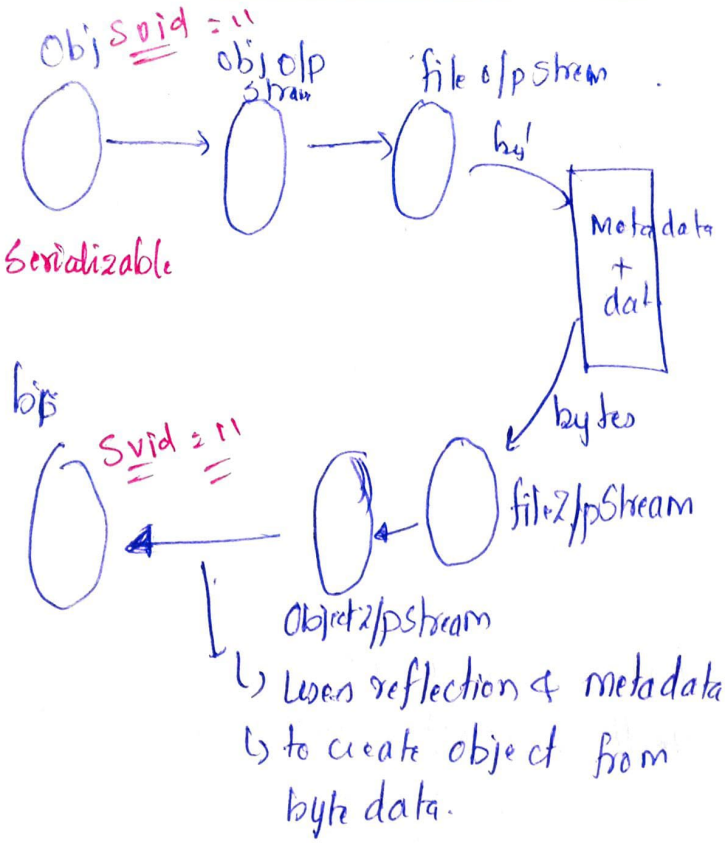
ConcurrentHashMap, copyOnWrite

- requirement is just change collection to make iterator fail-safe

serialisatn & de serialisatn new video



- byte → data + metadata



↳ uses reflection & metadata
↳ to create object from byte data.

Why marker interface (Serializable)

↳ it is like a flag, making the Object

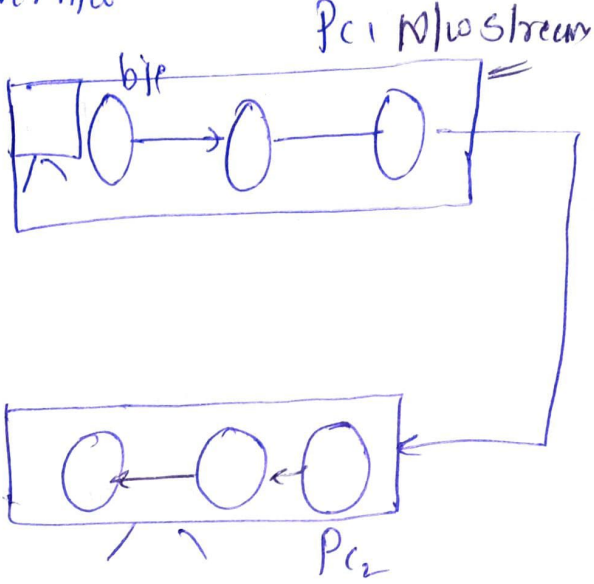
↳ To notify JVM

Security

1) along with data we are sending meta-data (object-structure) object graph
2) capable person can read object graph and read all data
3) JVM says dont blame me for security breach, developer allowing Security breach

4) e.g assume 11:19

⊗ serialisation is heavily used over n/w



⊗ pc with different version cannot communicate.

⊗ while serialisation it is must to give version to it

SVID

11:25 **OOP**

object slicing (not java in oop)

```
class person {
    // name addrss age
    public void setAge(int age){ }
    public void display(){ }
}

class emp extends Person {
    empid sal dept
    public void setSalary(d sal){ }
    publ void display(){ }
}
```

Overriding

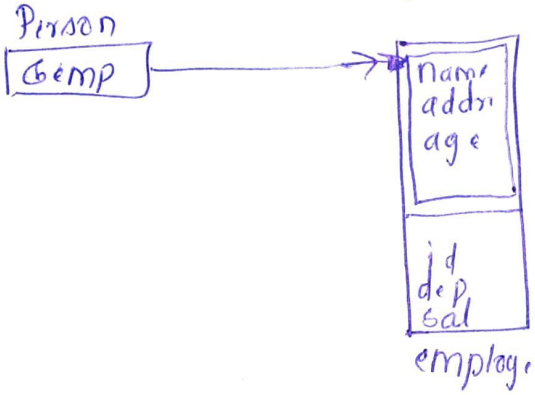
```
Emp e1 = new Emp()
e1.setSalary();
e.setAge();  → Inheritance

Person p1 = new P();
P1.setSalary(); → X/error
P1.setAge();
P1.display(); //->persondisplay
```

Every Employee is person

upcasting

```
Person p = new Employee();
```



person technically points person inside employee

- only person level info allowed to access. Object slicing

when subclass object reffered through a super class refere, it can only access super class functionality (and not subclass info) this is object slicing.

```
P.setSalary() X //java error
P.setAge() ✓
```

- at compile time p can see its own methods only because java is statistically typed, (check at compile time)

- always recommended to use interface type

Singleton Class

Design Pattern :- solution to common problems

- oop designpattern
- Web programming
- ~~Distr~~

Singleton

- global are not allowed in oop and even recommend global
- Hibernate session factory heavy weight.
  - ↳ singleton → all logic inside constructor private, → not allowed to create object,

```
Private static myClass.
    Static {
        called only once
        obj = new myclass;
    }
```