6/2/25

Important Basics

→ What is Java?

→ Why is Java packet centric language? (collection of class)

→ C, C++ are portable, only source code is portable.

→ Naming for packages is done @ in Reverse domain naming convention. (learn more)

→ ** Packages in Java

→ ** Namespace in C++ (like packages in Java).

→ Java's WORA (Write Once, Run Anywhere) is Imp**

→ Can Java call other language codes? We can.
How?
⇒ JNI (Java Native Interface)
↳ can make a call to any non-java-code.

→ Can two methods have same name in Java? ⇒ Number of params, type of params, ~~return type~~ return

→ ~~Can~~ can we have two public class in one file? ⇒ NO

→ * Whenever we have a public class, the filename and the public class name should match. So we can't have more than 1 public class, but we can have many non-public class.

→ Anonymous Object, Reference Error [One public class, create obj of public class in another class]

Reference → on stack

```
basics obj // Ref Error
new basics() // Anonymous Obj
// obj.main()
```

OCA

→ When we compile a java file, all the classes present in that file, will create a .class file.

→ 'Run Time Exception' is raised when the JVM ~~JRE~~ won't find the main 'main' which it executes (the public class main).

→ 'main' method overloading in java

→ 'Public static void main()'
↳ To call itself using classname.

→ To create object we need main

Chicken or Egg Problem which came first.
→ If not static, we need an object to call
→ But to create object, we need main

→ In C++, main is not static. (why?)
In Java, main is static.

→ starting of every java applcn is 'main'.

→ Package, Classes, Methods, Objects, ...
(Instances)

OOPs → Data + Operations
(Functions)

Auto Boxing → Integer i = 10 → same like new Integer(10)

Integer j = i → [i=10, j=10]

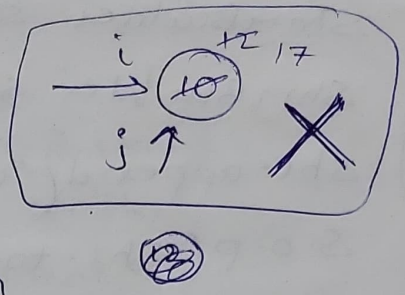
Boxing & Unboxing → i = i+2;

j = j+5;

S.O.P(i); ?

S.O.P(j); ?

→ [i=12, j=15]



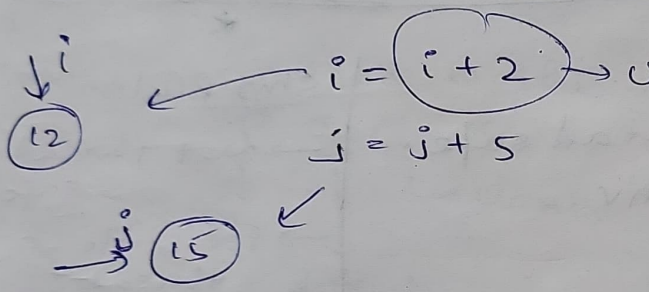
→ Wrapper classes : wraps their primitives

primitive → Object [Boxing]

→ Then is Auto Boxing in Java

⇒ All wrapper classes are immutable in java **

But References are mutable. (Addresses can change)



→ which objects are eligible for Garbage Collection here?

⇒ All wrappers, strings are immutable in java.

String s1 = 'hello';

String s2 = s1;

s1.concat('world'); [we did not assign it to s1]

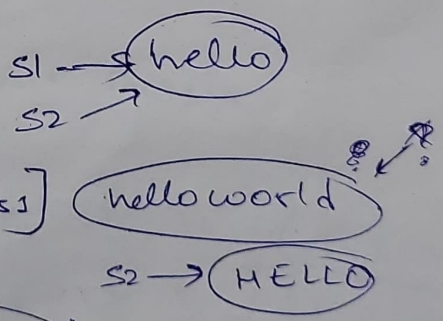
S.O.P(s1) // hello

S.O.P(s2) // hello

s2 = s2.toUpperCase();

S.O.P(s1) // hello

S.O.P(s2) // HELLO



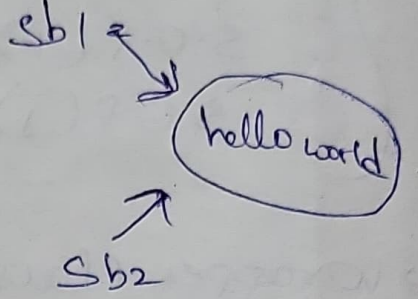
Did not assign to s1 here.

→ Here add & check :-

s1 = s1 + "world2";

[Strings created on String Pool]

```
StringBuilder sb1 = new StringBuilder("hello");
StringBuilder sb2 = sb1;
sb1.append("world");
S.O.P(sb1.toString());
S.O.P(sb2.toString());
sb2.reverse();
S.O.P(sb1.toString());
S.O.P(sb2.toString());
```



Content modified in-place here

StringBuilders are mutable

⇒ References are modified, not the content.

StringBuffer

→ StringBuffer is thread safe.

→ For Multi-threaded env

StringBuilder

→ Not thread-safe

→ For single threaded env

7/2/25

Packages -

→ collection of related class, interfaces etc
→ helps organize code in logical, hierarchical manner.
→ mainly is used to avoid naming conflict.
→ It gives a unique namespace to identify.
→ .jar (java archive)

Windows
els id

com → Component Object Model
- reusable components are made using this.
- No two components will ever conflict since there is a class id inside, which uniquely identifies each.

~~Steps~~

→ Always write code in the following manner -

* P I C *
- P → package
- I → Import
- C → classes

→ Explicit will have more preferance (precedance) than wild card, while importing packages.

import Pack1.subPack1.t1 > import Pack1.*;

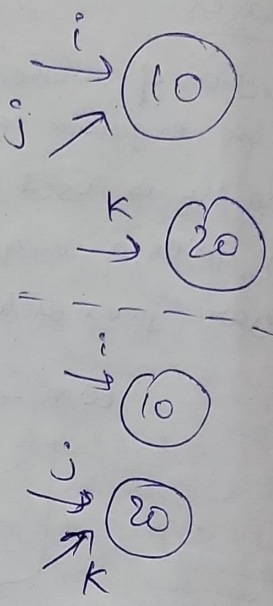
→ How is ["import module"] different from ["from module import *"] } Python

- "import module" → All define in that are available using module name
  > import np
  np.add()
- "from module import *" → using this kind of import, then need not use any Referenes to access the modules present in it. If Ref is used, gives error.

Using alias (as) in Python will avoid any naming conflicts of two functions

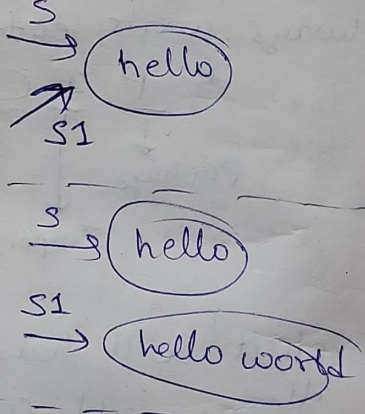
→ All numbers are immutable in java

```
i = 10
j = 10
k = 20
print(id(i), id(j), id(k))
j = j + 10
print(id(i), id(j), id(k))
```



→ All are immutable in java (Numbers, Strings)

```
s = "hello"
s1 = "hello"
print(id(s), id(s1))
s1 = s1 + "world"
print(id(s), id(s1))
print(s, s1)
```



s = hello
s1 = hello world.

[Immutable: Content cannot change, address can change]

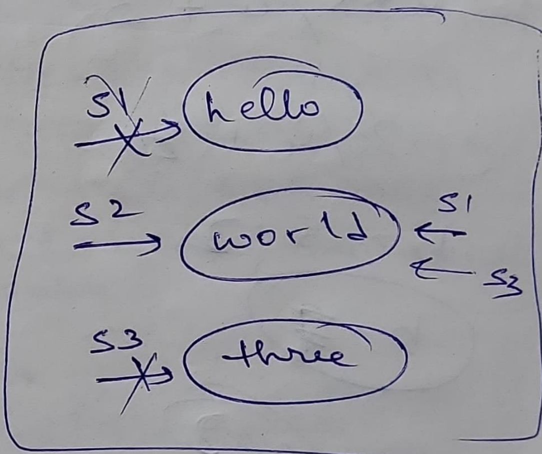
`s[0] = 'H'` → we cannot do this, since we cannot change the content.

`print(s[0])` → works, cause this is just reading a char, not modifying the content.

Garbage Collection

- → We dont need to free the memory
- → Java removes the unreferenced objects, freeing up memory.

```
class three
{
    public static void main(  )
    {
        1 new three(); // Anonymous obj, has no ref. becomes garbage collecti
        2 String s1 = new String("hello");
        3 String s2 = new String("world");
        4 String s3 = new String("three");
        5 s1 = s2;
        6 s3 = s1;
    }
}
```



which objs become eligible for garbage collecn?

> After line 1, the anonymous obj becomes eligible

> After 4,

13/2/25

Read Java Documentation

→ 4 byte ✅

Try

```
short x = 10
short y = 20
short res = x + y - (x * y)
```

Gives compilation error.

→ By default, all decimal numbers are treated as double unless we explicitly mention if, or other.

$4 * 5 / 4 + 3$

$20 / 4 + 3$

$b = 5 + 3$

$b = 8$

$a = 2$

$a = 3$

$b = 4 * 5 / 4 + 2$

Try

```
Integer str = 10;
S.O.P (str instanceOf Number)
S.O.P (str instanceof Object)
S.O.P (str instanceof String)
```

java.lang.Object
java.lang.Number
java.lang.Integer
Hierarchy

→ If any operand is 'short' or 'byte', when performed arithmetic operation on it, they will implicitly be converted to ~~the~~ ~~long~~ int * *
So the result storing the arithmetic result of them should be declared as 'int', inorder to store the result.

→ For any two operands of different data types, when performed arithmetic opn on them, then automatically the result of the opn will be of ~~type the~~ the larger data type among the two operands.
(When performing for double & float, then result of arithmetic will be of double.)

→ In C, C++, 0 → False
Anything non zero → True
~~anything~~ 0 is False, ~~non~~ anything non-zero is True

→ In Java, it has to be boolean.

→ 'if' always has to be boolean in java.

C, C++ [int x = 100 ; if (100 = x) { ... }] Run & Check [int x = 100 ; if (100 == x) { ... }]

Java [int x = 100 ; if (100 = x ; { ... }] Run & Check [int x = 100 ; if (100 == x) { ... }]

Integer z = 20 → z is a wrapper here.

~~Integer z = 20~~

```
x = 6
y = (x >= 6) || (++x <= 7)
s.o.p(x)
```

Short Circuit OR

This operator Evaluates only first expression if true, it continues

|| → Short circuit OR operator

→ 6

```
x = 6
y = (x >= 6) | (++x <= 7)
s.o.p(x)
```

| → OR. (Not a Short ckt OR)

This operator will evaluate all the expressions

→ 7

```
Integer z = 20
if(z != null && z.intValue() < 25)
{ s.o.p(z) }
```

Short Circuit AND

If any part of expression is wrong, it won't evaluate anything else

If first part true, evaluate the second part also

```
~~Integer~~ z = null;
if (z != null & z.intValue() < 25)
  { s.o.p(z) }
```

Leads to Null Pointer Exception

14/2/25

→ If there is no 'break' in switch-case, then there will be a fall through to other cases.

→ C, Java, C++ compilers will evaluate 'default' only if other cases are not matching.

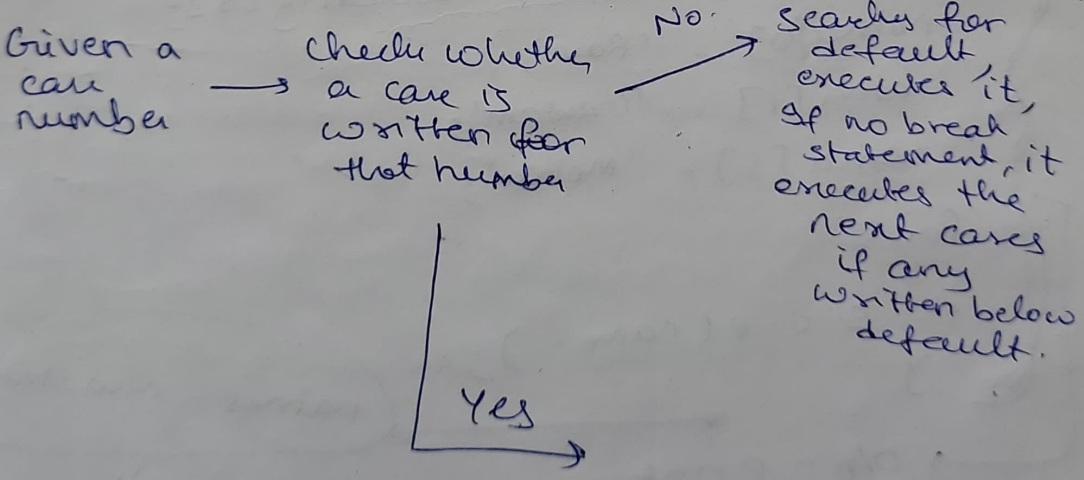
⇒ It will evaluate default only if there is no matching case.

→ 'default' can be placed anywhere in the switch-case code.

~~After default~~

→ If we dont want fall through, then we need to use 'break' statement.

→ We cannot have a 'continue' statement in switch-case code.
since when there is no break, automatically it is continue.



→ Switch Case can accept int, String, enum, Wrapper Classes,

"final" meanings in Java :-

(1) Variables declared with 'final' keyword are constants, cannot modify. (Final variables)

(2) 'Final' class cannot be a sub-class. All immutable classes are final. (Final class) Since if not final, we are telling the child class to change the behaviour. If not final, Inheritance happens & it will allow subclasses to modify the behaviour Ex- String class, System class.

(3) Method declared 'final', the subclass cannot be overridden

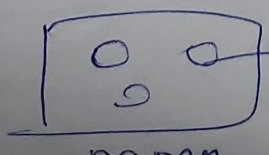
(4) Input arguments can be written with keyword 'final', then the ~~variable~~ args value (Final arguments) wont be modified [Not sure] [Read]

```
final String[] names = new String[3];
names[0] = 'Ravi';
names[1] = 'Gautam';
names[2] = 'Mohan';
for (          )
{
    S.O.P(name.);
}
```

names = new String[3]

Now gives error. (:: creating new instance, changing the ref.)

O/P - Prints -
Ravi Gautam Mohan



→ we can change data present in it, but should not change the instance.

→ Strings are not iterable.

→ Iterables are the ones where we can get a value out of the values present in it (array, etc).

→ * All References by default are null *

→ local variables are the ones declared inside the methods. (not sure, read)

→ variables declared in main, will not come under local variables (not sure, read)

java.util.List<String> values = ~~new ArrayList~~

↓ Interface

= ~~java.util.list<String>();~~

= new ArrayList<String>()

Class.

[Class implementing Interface]

Interview Questions

(1) Tell me about yourself. ** **

Do not tell anything which is not there in resume

- Name
- clg
- talk about achievements & certifications
- Talk about recent projects related to them

(2) Give a real time practical example of using 'static' keyword. Example of static data member.

(3)

Ex - static scooter,

Everybody can use, none of them own it

- taking the same scooter for clg by student
- for market by mother
- for movies by sibling

→ we can use static to solve any problem, which is having count.

Total count = 250 → total

static var = 170 → active member

** Always learn practical examples while learning java concepts **

(3) Static Initialization V/s Instance Initialization

(Static Block, Instance Block)

Static Block: comes once only when a class is loaded.

Instance Block: happens everytime even before the constructor is called. Everytime instance is created.

(4) What is function overloading, explain one function which is used for overloading (from api if possible).

constructors are ~~functions~~ special functions (methods)

* Java Documentation *

(5) ~~The~~ Explain about constructors.

(6) constructor is private in one class, we need to access that constructor in another class how?

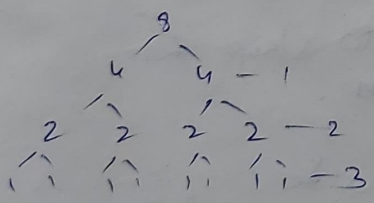
↳ Create a method, declare it static and call the constructor inside the method.

This will actually be like a chicken-egg problem if the method is not declared as 'static'.

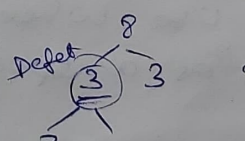
8) Why is the main method always created as public and static?

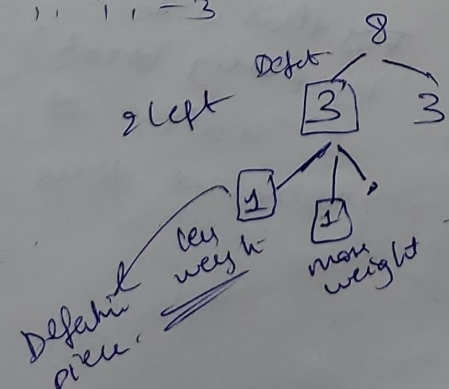
main() → public → so that JVM can access it
→ static → can access without creating Obj

(7) There are 8 marbles, one marble weighs lesser than other 7. There is a weighing machine to use. How many times can weigh to find the defective marble. Tell the least number count. (minimum count)



3 times





3 = = 3 (Check other 2)

(8) Why should I hire you?
- be modest
- whatever terms you mention, you should be able to justify them.

(9) What has been the most challenging task in 4 years.
(Professional challenges preferred)
[Talk abt challenges faced in your projects & how you resolved it]

(10) Where do you see yourself ~~in 5 years~~ 5 years from now?

18/2/25 — for Each V/s Traditional for loop

How does for-each work internally?

→ Internally compiler convert the for-each into a normal traditional for loop. (for(init, condn, increment/decrement))

→* When using for-each, we never get Index Out of Bound exception.

* → Code writing is easy, when using for-each.

→ Can only be used for iterable objects

→ Cannot use when we want to skip the loop for a specific iteration.

```
int x = 10
while(x>0)
{   do{
        x-=3;
        break;
    }
    while(x>5)
    x--;
    S.O.P(x);
}
```

try this code by playing 'break' & 'continue' at various lines

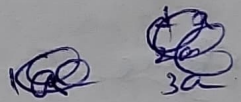
Labeled Block

→ we can 'break' a loop using that loop.
→ useful when dealing with nested loops.
→ Giving a name to a loop or block.
→ for readability, use Upper Case naming convention.
→ For first occurrence of unique elements, we use these labeled blocks & break them using the label.

```
PARENT_LOOP: for ( . . . )
             {
                 for ( . . . )
                 {
                     & &
                 break PARENT_LOOP }}.
```

↑ Break with label

↓ Continue with label



Anonymous Class

[Read Non-Anonymous & Class Anonymous Class also with constraint]

→ Class without a name.
→ Can use it for comparators.
→ Only one instance through which we can call - a method

~~Interface~~ @ Annotation :-

** → Annotations are those compiled time constructs which provide additional meaning to compiler ~~for run time~~

→ At compile time or run time.

@override : telling compiler, that is is an overidiy method

~~for class~~, try for classes once.

→ signature should be exactly same as parent.

→ when mentioned this, this method must be defined in parent & this is modifying the parent method. If not defined in parent, will result in compilation error.

this results in compilation error for Interfaces

```
Interface I1 {
  { void m1(); }

class Demo implements I1
{
   void m1()
   {
      S.O.P("m1 called");
   }
   publi stati void main(Str[] ag)
   {
      new Demo().m1();
   }
}
```

try by overloading this method (both Interface & with class)

Does NOT Compile

Make this as 'public', then it will compile.

```
public void m()
{ S.O.P(); }
```
will compile

→ All interface methods by default are public & abstract

All interface variables: public, static final

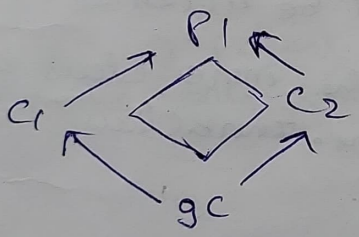
Interface
- By default → methods : public abstract
- By default → variables : public static final

- ~~Interface~~
- we cannot create Object for abstract class.
- A class must be abstract, if it implements abstract methods.
- Annotation provide additional info to compiler telling what to do.
- Every annotation has different meaning
- Try creating anonymous class. ~~Obj create class obj.~~

* Marker Interface in Java
* Serializability in Java

Interview Question

① Diamond ~~class~~ Problem (called as Repeated Inheritance in C++)



→ How is it solved in CPP?
(→ Virtual Derivation)

** Marker Interface [Interface Serializable]

→ An interface which has no methods defined.
→ we dont have to implement anything.
→ Read from documentation.

Serializability: Converting Obj data → to → Byte stream
(Serializability)

Deserializability: Byte stream → convert to → Obj Data

** Serialization **

→ Then interfaces help the framework do something.
Ex - Interface Remote
there are many other examples for this interface.

Inner Class

→ A class within a class is known as inner class
→ the inner class cannot be accessed directly
→ using the outer class, the inner class is accessed Ex - Outerclass.InnerClass.

Garbage Collection

-> Garbage Collector internally uses Reference

24/2/25 OCA - Ch 3

** What are things which is to be done in order to make immutable?

For a class to be immutable.
- For a class, we dont define any 'set' methods (setters) (getters) and keeping all the data members private.

- A String is a final class
  Integer is a final class
  } Why are they final?
- 'final' → when used, will not be able to create sub class

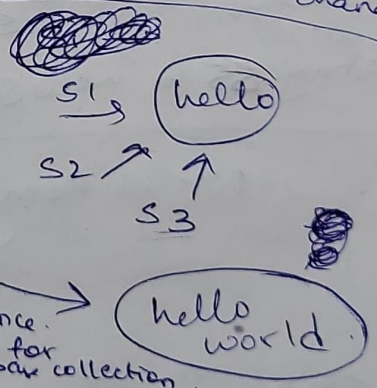
** TO Make Immutable ** ----------

1. data members should be private, and should not have setters
2. make the class final, so that ~~one~~ cannot inherit & so cannot modify the behaviour!
   - since when inherited, the behaviour of parent method can be modified, hence should not enable inheritance.

Strings -

Reference changes, but content of obj wont change

```
S1 = "hello"
S2 = "hello"
S3 = S1
S3.concat("world")
S.O.P(S1, S2, S3)
  → hello hello hello
```



→ JVM internally maintains String Pool. If any string is not present, it adds that string to String Pool.

✓
```
S3 = S3.concat("world");
```
S3 → hello world

String s4 = new String("hello")

↳ Not a string literal in pool
↳ Obj gets created in 'Heap'
↳ Heap object having value = hello
↳ Will not conserve memory.

[Use string literals to save memory]

⇓

Can have Multiple Referency to same pool obj (pool obj) instead of creating multiple object

---

s1 → 1 ← s2

s1 → 12

s1 → 123

s3 → 5

s4 → 56

? → 567

Diffn btw "==" & ".equals()".

- "==" → Check whether pointing to same literal or not.
- ".equals()" → Check value is same or not

String Pool also known as 'intern' Pool

intern() method → returns obj content into pool. → gets that string into String Pool.

String s5 = new String("Hello").intern()

literal from Pool & Obj from Heap → Not same

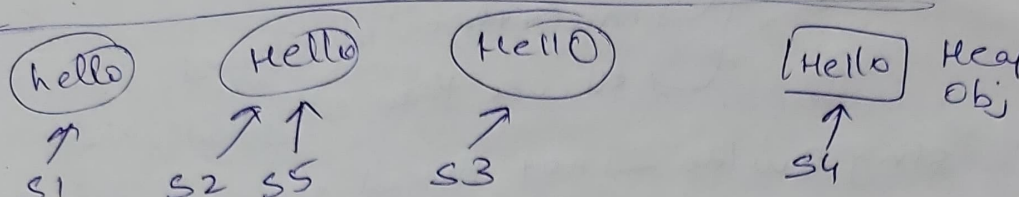
String s1 = "hello"

String s5 = new String("hello").

Is s1 == s5 ⇒ NO

```
String s1 = "Hello";
String s5 = new String("Hello").intern();
```

Is s1 == s5 ⇒ Yes.



** ⇒ Always check when there is a 'concat', check whether it is being assigned to some reference or not.

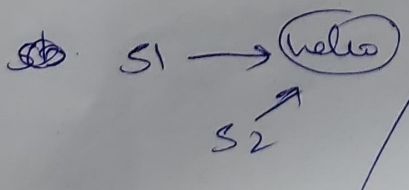
** ⇒ If not referenced to anything, that concatenated value will be eligible for garbage collection.

```
String ch = "";
for(int i=0; i<26; i++)
{
    ch = ch + (i+i);
}
s.o.p(ch)
```

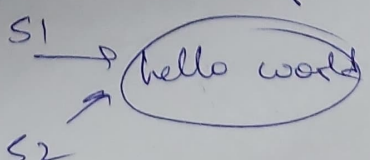
Inefficient code.
Every iteration, ch will point to new pool object.
At the end ch = "1 2 3 ... 26".
All the intermediate pool objs are eligible for garbage collection.
To overcome this memory inefficient code, use mutable data structure like StringBuilder or StringBuffer.

→ String methods are all overloading methods.
i.e with a single fn, by varying parameters we can fn with diffn purposes.

→ StringBuffer & StringBuilder are mutable, when changed content of s1, s2's content will also be changed



s1.append("world")



→ In multithreaded applcn, go for thread safe like String ~~Buffer~~ Buffer.

String Buffa → thread safe.
- → multi threaded allowed
- → slower than String Builda

String Builda → not thread safe
- → allow sigle threaded only
- → faster than Stng buffa

"==" & ".equals()"

→ both actually compare references, but in many cases the .equals() is overridden & will check the value
- ↳ In String Builda, the .equals() is not overridden.
- Hence .equals() & == will be same unless overridden. in String Builda.

25/2/25

→ Overiding { equals() hashcode() toString() } → these are not final methods

Overriding these methods when we create a class is a good practice.

→ equals() by default will check object equality i-e, references (like ==), we should overide & implement our functionality in it.

How to override override equals?

* Shallow * Equality Check → checks only whether references are equal or not; will not check values.

what is the use of writing @override?

* → Everything is a subclass of object in Java *.

→ Every class we create in java, are a subclass of the object class.

→ we have to override equals method, if we want to compare values & not references, else will not work. → @overide equals

One class two obj

First check whether the passed arg is of istance type, then check for value equality

Else we will get 'ClassCastException' * *

* Using 'isInstanceOf'

→ 'InstanceOf' is an operator

Class name with hexadecimal representation.

↳ default toString will return this, if not overridden.

** Prefer int[] a,b. ✓

Dont prefer int a[], b[] ✗

*

→ We cannot specify size at the time of array declaration in java.

→ We can initialize the array, but we cannot mention the size.

```
int a[] = new int[3].
S.OP(a[0], a[1], a[2])
        ↳ 0,0,0.
```

Not Valid

```
int [] aw = new int[3] {10, 20, 30}
```

we cannot give both at a time.

```
int[]arr2, arr3 = {3,9}
```

→ Only the right most one will be initialized with those values

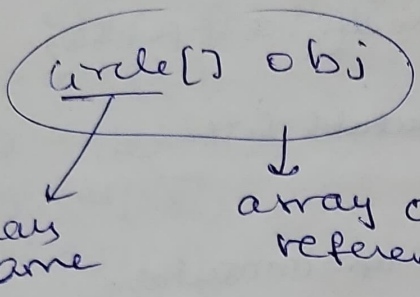
Reference arr2[0] → becomes invalid.

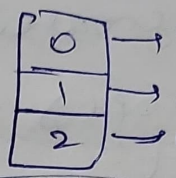
Arrays.deepToString → converts into 2D string array

.toString() → returns hash code.

Read About → Null Pointer Exception.

→ Instance variables (class variables) are - initialized to default values.



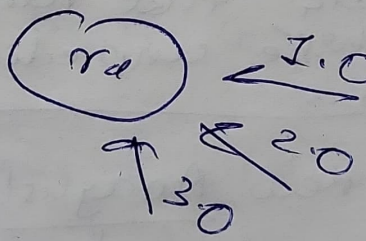


Referency Nothing.

Leads to Null Pointer Exception

to create array of objects, write a for loop &

→ Non-static members cannot be accessed in Static member can be called through an instance obj and also directly using class name



27/2/25 Array list

Can we have array list with fixed set of elements? Cannot add any more ele. Once created, cannot modify. Is it possible?

- Immutable Array List.
- From a list, an immutable list can be created.
- List.of() will create an immutable list.

In Binary Search, when given sorted data, for.

Negative of that index + 1

any ele which is not present in list the index of that not found number will be returned as -

$$-(\text{that index where number would be present if in case it was present}) + 1$$

For unsorted data that target element's index will be unpredictable.

ArrayList list = new ArrayList<>()

- list → Interface → Reference
- new ArrayList<>() → class implemented → Implementation class.

This is known as Polymorphism.

Same method call has diffn behaviour based on which class it is implementing it.

Set obj = new HashSet(); → can add any type.

This means it will accept any object → Add expects 'obj' as an object argument

```
Set obj = new HashSet();
obj.add(23);
obj.add(5);
obj.add(45);
obj.add(2);
S.O.P(obj)
```

HashSet → some random order

TreeSet → Sorted order.

LinkedSet →

When we add a string above & use a tree set, we get class cast Exception because, treeSet will sort all the added items, when sorting it sees others as Integer & one as String, unable to compare. So it is better to mention Generics in Angular brackets

** ⇒ It is better to get a Compile Time Error than a Run Time Exception *** *

* 6) Why do we prefer Generics over non-generics

When given generics → all data of same type. (type safety)

→ will get compile time Error

→ when iterating, we can know the next ele type. which is present in list.

list.remove(1); → this is considered as an Object (here Index)

.remove() will return boolean.

True → if removed successfully

false → if unable to remove.

→ When we create an array from a list using .toArray(), that created array will consist of the list data with the list size.

→ Later that array's content can be modified, but cannot modify the size of the array. change in content in array won't affect list

{ listRef.toArray() → converting list to Array
Arrays.asList() → Array to list

→ When we create a list from array, the list content can be modified, but the size of the list cannot be modified directly or indirectly

→ change one will change other

→ There is a tight binding both array & list here.

Creating Immutable List-

(1) List<String> li = List.of("a", "b", "c")

↓
will create an immutable list.
(cannot change - neither size nor contents)

(2) List<String> Immu = Collections.unmodifiableList(list)

↓
cannot add or change content or size

When added new ele → UnsupportedOperationException (Runtime).

For Map -

Collections.unmodifiableMap(m); → ref for map we created.

↳ gives error if added new keys & values; cannot modify. (Throws UnsupportedOperation)

Interview

Be aware of HR current affairs

a) What is an exception?

a) Diffn btw ~~Runtime~~ and compile time exception
(unchecked) (checked)

a) Diffn btw exception and error?
↳ subclass of Throwable.

a) Finally block.

a) How to create user defined exception?
How to create user defined unchecked & checked exception.
↳ create a subclass of Exception
↳ checked.
↳ create a subclass of RuntimeException
↳ unchecked.

File not found exception → through try-catch.

a) Can you prevent an exception from occurring.
↳ exceptions cannot be prevented.

a) What is an assertion in java? Real time example?
↳ assert will not throw exception.
↳ It will throw assert error
↳ Assert statements are assumed to be always true.

a) What is a thread? Give me real time examples of multithreaded applications.
↳ Smallest unit of program for execution.

Q) Difference between thread & process?

thread → light weight process among the process.

* * [Any multithreaded applcn improves user's responsiveness] *

* [Any real time applcn, will be built as a multithreaded applcn]

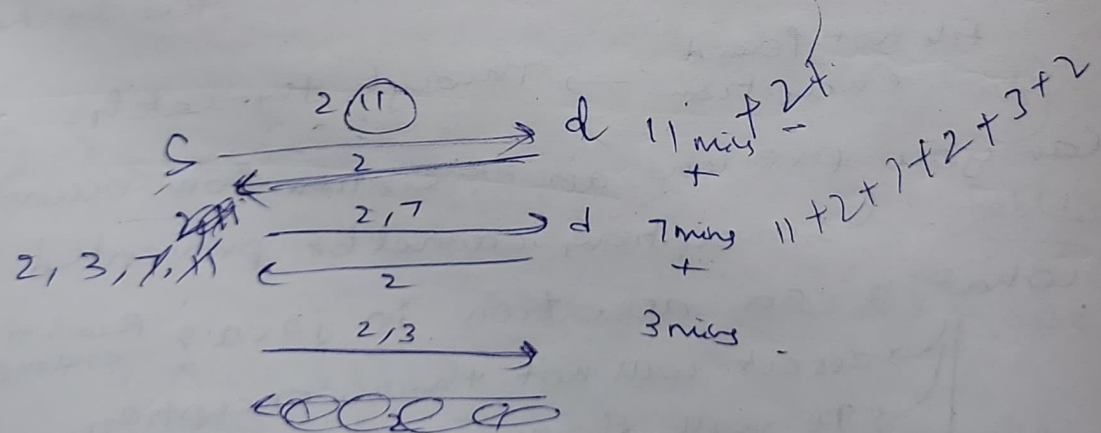
Q) What are typical problems with multithreaded applications.?

Join() → to include synchronity.

Puzzle -

(Q) [illegible]

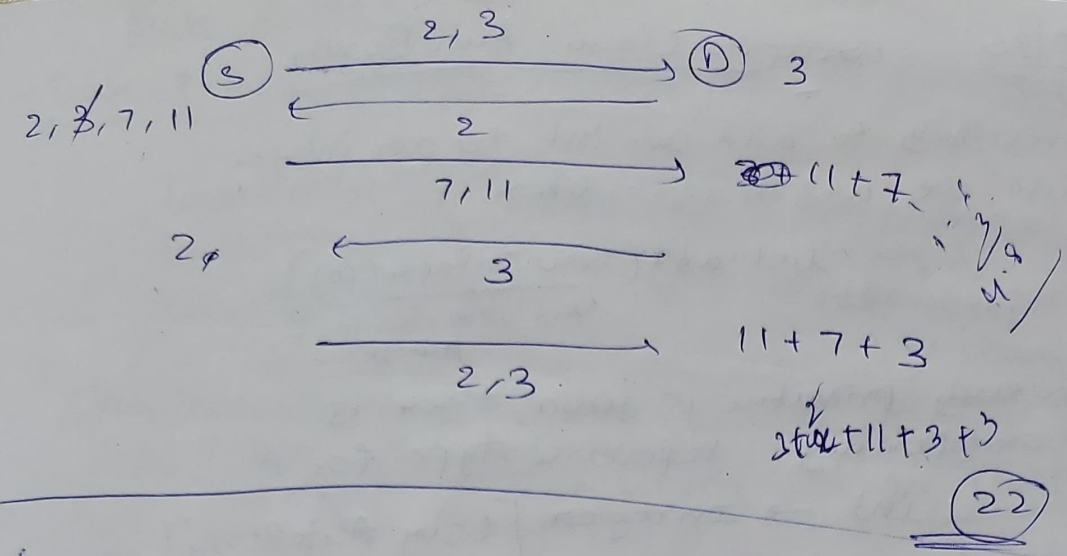
What is minimum total time required to cross the bridge by all of them



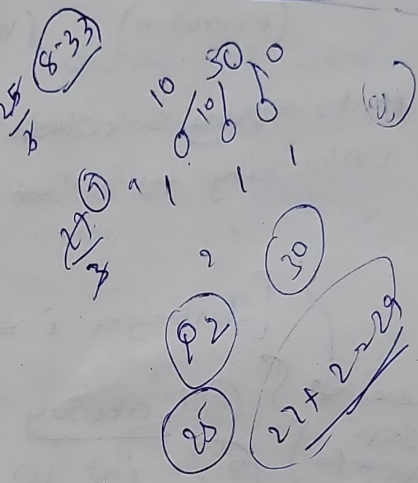
11 + 2 + 7 + 2 + 3

25 mins

11
2
7
5
25



22



28/2/25 Wrapper Class, AutoBoxing

→ Previously to add an int to an integer list, we used to write like -

```
list.add(new Integer(10))
```

now java does this by Autoboxing

→ for every primitive in java, there is corresponding reference type for it

int → Integer [Using AutoBoxing]
(primitive) (reference)

→ Autoboxing → when adding something to Collection
Unboxing → when getting something from Collections

```
[Integer i = 10; i++]
```

There 3 steps happen for above 2 lines:

① Unboxing will return int 10
② int 10 became 11 after i++
③ 11 is now Boxed (Autoboxing)

* Integer & all other wrapper class are immutable. *

" NumberFormatException " arises when we assign String to some numeric data type.

Date & Time

→ Date, Time, DateTime class are all Immutable

LocalDate
LocalTime
LocalDateTime
} immutable object → value cannot be changed, references can be changed.

class LocalDate
Date
Time
} → All they don't have constructor.

All methods are static

(Q) class with no constructor, how to call methods?

[classname.staticmethodname]

No constructor, so cannot create object, gives error.

LocalDate.now()
LocalTime.now()
LocalDateTime.now()

~~localdate1~~

LocalDate date1 = LocalDate.of(2017, Month.March, 5);

Month is Enum, Jan is value it

Enum (Enumerated).

[LocalDate date2 = LocalDate.of(2017, 3, 5)]

Enum → enumerated constant.

→ Date only works with Date parameters.
→ Cannot add time to Date. like Minutes.

→ Creates a new date obj; if already present, it modifies the existing ones.

3/3/25

Period → duration.

Period period = Period.ofWeeks(1);

Period.ofWeeks(2)

isBefore → checks both & returns which one is before or after.

while (start.isBefore(end))

Period.ofDays(2)

Period.ofYears(1)

Period.of(1,2,5)

Biggest Problem with Dates → Date format.
→ Time zone is not a problem

LocalDate date1 = LocalDate.now()

there are many APIs → date1.get
date1.get

End Date >= Start Date

date1.isBefore(date2)
date1.isAfter(date2)

Date Time formatter → long
→ medium (Enum)
→ short

DateTimeformatter.ofLocalizedDate(FormatStyle.MEDIUM)

Enum → full fledged classes.

* Time object cannot be used on date
Date object cannot be used on time *

LocalDateTime → used to get both date & time
→ using formatter we can get only date or only time

LocalizedDate
LocalizedTime
LocalizedDateTime
→ works only for Time
→ works only for Date

} ofLocalizedDate
ofLocalizedDateTime
ofLocalizedTime

LateTimeformatter dtf = DateTime formatter.ofPattern("MM:dd:yyyy:hh:mm:ss")

DateTimeFormatter.ofPattern("mm:dd:yyyy:hh:mm:ss")

DateTimeParseException
→ RunTime Exception
→ Invalid Value for HourOfDay

6/3/25

Constructor Chaining (read)

→ abstract methods are never implemented.

test(int , int... x) → var args

test(int , int[] y)

Both are same (identical)

→ We cannot have static methods as constructors.

→ Constructors should not be 'abstract'

protected → package → children

access specifiers

→ Final class cannot be subclassed.

→ Final classes: Strings, Wrapper Class, LocalDateTime, ...

→ Final methods cannot be overridden

→ Final methods: wait(), notify(), notifyAll() → Object Class

→ If a class is final, it cannot be sub-classed means methods cannot be overridden.

So implicitly, if the class is final the methods cannot be overridden.

→ Input arguments can also be 'final'. This will prevent if at all a re-assignment of that variable is done inside a method (which should not be changed), then will raise an error. Final vars cannot be modified.

* To make usa defined class are immutable –
  - No setters
  - make data members private.
  - constructors should be there.
  - making the class final (Not mandatory).

Do this only if the full functionality of the class with all the required methods are present.
If not, we can add those required methods in those sub-classes.

Can we overload a final method?

① There is only one copy of static data, irrespective of how many objects created.

```
static int val;
Test (int x)
{   val = x;
}
int getval()
{   return val;
}
p s v m(str[] arg)
{
    Test t1 = new Test(10)
    Test t2 = new Test(20).
    S.o.p(t1.getVal()) → 20
    S.o.p(t2.getVal()) → 20.
```

Initially it was 10, but was reset to 20, since only single copy of static data member will be maintained

Reason, read next page

```
t2 = null
S.o.p(t2.val) → 20.
```

→

→ Whenever a compiler sees a static method, it replaces that with class.

→ Everytime we refer to something which is static, when calling through an object the compiler will replace that with class.

Private Constructor

→

```
class test{
    private test() {}
    void m1() {}
}

class demo{
    Test t1 = new Test()
    t1.m1()
```

→ Constructors cannot be private

→ Since the method is private, we can call the constructor in it using createInstance()

Ex - JVM calling Main to create function

In C++, we can create global objects.

→ In Java, there is no way to create object (read not sure)

Why is the main method in Java 'static'?

→ JVM needs to access it,
for JVM to access that class, that method needs to be static.

Design Patterns

→ solutions to commonly occurring problems.

3 main
- Creational
- Structural
- Behavioral

Singleton

Gang Book
Design Patterns by the Gang of Four

To create a singleton ~~test~~ design pattern, there are 3 steps -

1. Declare a data member which is static
2. Define the constructor as private.
3. Define a static member function which the end users can call, (which internally will create an instance for ~~cls~~. you.

Interview

- How does an LLM work? (Since LLM project is there in resume)
- What was your role in that grp project?

[Mention only what you know.
You have to know in-depth of whatever you put in your resume]

- You prefer Java? From what topic should I start asking questions?
- Multithreading Concepts. Thread Priorities, can they be changed?
- Scheduling algorithms? Round Robin et et
- Thread States.
- When done t.run() what state is it in?

* [start() → multithreading
run() → single thread.
→ no multithreading here.

* [start is a callback method.
start will internally call run method

Don't tell terminology or concept which you don't want the interviewer to ask.

→ Multithreading is call back method model.
→ start is call-back threading.
→ Never call run (callback)

** Difference between t.run() & t.start()?

Q) What is meant by a Daemon thread?
→ A daemon thread is a background thread
→ This thread runs until all the other threads complete their execution.
→ read.

Q) What is a producer-consumer problem? how do you address the producer-consumer problem?
↳ wait & notify:
↳ these have to be called in a synchronized block.

Q) Give me real time example of where multithreading is required?

Q) Benefits of multithreading?
↳ Better Responsiveness. **.
Even if one thread fails, other threads need not to wait. they can continue.

Come up with example of your own which you can tell

→ @
Q) Annotations for runtime?
annotation for compile time?

Q) Runnable V/s ThreadClass, which to choose? Why?
↳ Runnable is preferred. Why?
→ In Runnable, we are forced to implement 'run' method
→ In Thread Class, where we are not forced to implement 'run', the main thread's default run is implemented (by default).

HR
Q) Your senior is doing wrong in some technical issue, you know that. How could you get that to notice?
↳ Empathy works here.
↳ Dont break interpersonal relationship.
↳ Ask him, can we do the other way... ok?

19/8/25 TCL (Transaction Control Language)

1. Commit
2. Roll Back
3. Savepoint - instead of complete roll back, we can roll back to a particular state

→ After data is committed, the savepoints no longer exist.

Commit → makes all changes permanent

↳ Automatically savepoints are destroyed since after committing, there is no way to roll back.

Stored Procedures (stored at RDBMS)

If database changes, procedures needs to be re-built

→ like methods (APIs) in Java

→ It is like a Java API → since, we start using which others have already created.

→ Stored Procedures are stored at server side of the database

→ Block of code, identified with a name with any number of arguments, when called the functionality will be implemented.

→ It is built once, can be used by anyone.

1. Easy of Usage
2. Consistency in Output (Each developer need not know the schema)
3. Reduces Network Traffic

   Compilation time reduced.
4. Re-usable & transparent
5. Secure → other than permissions for procedures, no other permissions are given to user so data secure

20/3/25

Triggers

→ Trigger get triggered for Insert, Update, Delete operations

→ Only a total of 6 triggers can be applied to a table.

6 triggers:
1) Before Insert 2) After Insert
3) Before Update 4) After Update
5) Before Delete 6) After Delete

```
Create Trigger <trigger name> <trigger time> <trigger event>
on <table_name>
for each row
Begin
    --- // event code.
End;
```

<trigger time> → Before or After

** → Used for Audit purposes.

Views [A table which is virtual]

→ Virtual tables. (when done show tables, even views will be visible) **

→ no need to know the schema

→ Everytime updated the main parent table, the views also get updated.

Views → presentation layer

Parent Main Table → backend.

→ Views are not only queryable but also updatable.

Why Views?
- Restricting access to data
- Making complex queries simple
- Ensuring data independency
- Providing different views of same data

** → When joined two or more tables, we can use views to store the result.

Are Views Updatable? Yes And also NO

* We can also insert into View, which will update the Base table [Updatable Views]

to create updatable view, select statement should not contain these -

1. Aggr. fns. like Min, Max, Sum, Avg & Count
2. Distinct
3. Group By
4. Having
5. Left Join &

21/3/25

Normalization

→

Problems without Normalization
1. Updation Anamoly
2. Insertion Anamoly
3. Deletion Anamoly.

Dependencies: Deths

Multivalued Attributes

Partial Dependency

Transitive Dependency - when a non key attribute determins another non-key attribute

Normal forms

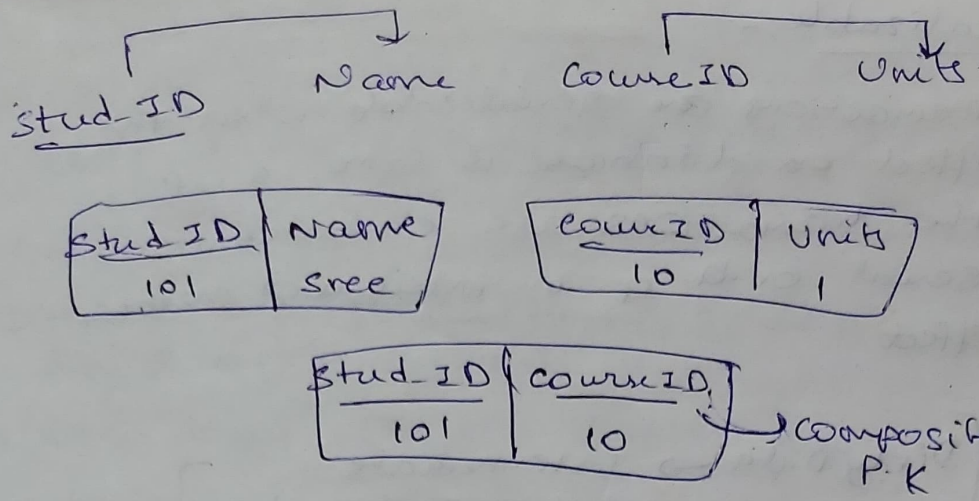
① Unnormalized – there are multivalued attributes or repeating grps

② 1 NF – No multivalued
(any row must not have a col in which more than one value is saved)
↓
All are directly or indirectly dependent on Primary key

③ 2 NF –

② No partial dependencies, therefore atleast 2NF

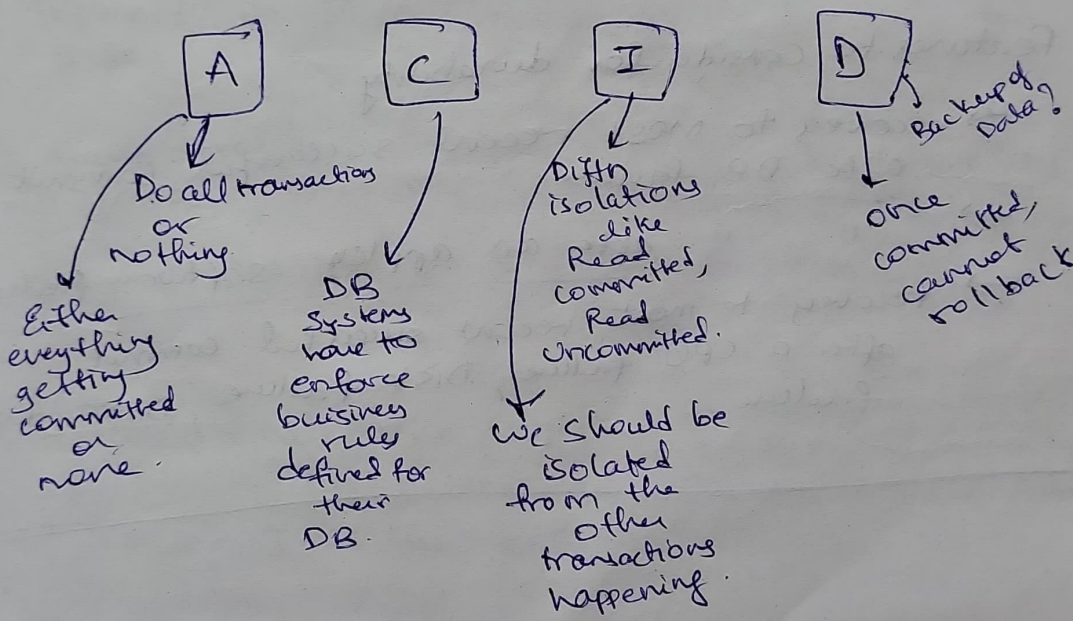
③ there is transitive, so



ACID Properties * used in transactions **

A - Atomicity
C - Consistency
I - Isolation
D - Durability

} Every transaction should fulfil the ACID properties.



↳ A single logical operation on data is called a 'transaction'. (Debit from one acc, credit to another acc)

↳ Each transaction is said to be atomic, if when one part of transaction fails, the entire transaction fails.

↳ All DB's will maintain a 'Write Lock' so that no two users access the same one & perform transactions at same time.

Serializable -

→ transactions are serializable when the effect on database is same whether the transactions are executed in serial order or in interleaved manner

→ effect

[Dirty Data → Intermediate transaction data]

Degrees of Isolation

- Degree 0
- Degree 1
- Degree 2
- Degree 3

Features to consider for durability.

⇒ recovery to most recent successful commit after DB failure.

⇒ " " after an applcn software failure.

⇒ recovery to most recent sucessful commit after a CPU failure, Disk failure, Data disk failure.

Interview

- HTTP is stateless, why?
- Session Management. (when doing a MERN project, understanding this ~~for~~ related to our project is imp)
- Exceptions in SQL.
- Difference between functions & stored procedures. Adv & disadv of each.
- When do we use a self join? how? → using alias
- When do we use subquery over a join? → subquery: when no relation btw tables. join: to join multiple tables
- Difference between left join & right join.
- Difference between Union & Union All? → takes duplicates too.
- 'Distinct' in Java (implementation how?) & SQL → Set
- We have a table in db, convert that table to a class in java. → getter & setter → for data. insertion → constructor (create objects for each row)
- ~~D~~ what are getters & setters in java. → for updations
- Internationalization (I18n) → Any msg in one language shd be available in other language too. Design a DB system to address internationalization ~~how?~~ Tell the schema design.
- How did you handle a challenging situation in your 3/4 years of college?

26/3/25 OCA Ch-5 Class Design

→ Java support single, multiple level inheritance.
↳ parent, child, grandchild

→ Inheritance: extension of a class or restriction of a class.
- extension → changing functionality of methods by overriding.
- restriction → restricting access of data members by making them private

→ Implicit extension from Object class which is done by compiler for every class by default.

→ 'extends' keyword is used to extend from other class & not 'Object class'.

Hence we will not have any class combination of these both
- abstract → have to subclass
- final → cannot subclass

→ In every constructor, first line is super() by default
super() → will call parent constructor.
- By default will call the parent constructor with no argument unless we mention super() given with parameter.
- If we dont write anything in constructor, super() will be called.

→ We call constructor using 'this' keyword

private method → only available to that class only, not even subclass.

super → keyword – used to refer to parent's methods & variables

super() → statement – for parent constructor.

** When is super used?

→ When we want to access one method from parent though the same method is written in child.

→ Because parent's method functionality & the same method of child ~~class~~ class may have different functionalities, though the number of arguments are same.

* super() has to be the first statement of the constructor.

When no constructor is written → Default constructor gets called → will call → parent constructor (super())

this() → constructor within same class

super() → Parent constructor.

→ Everything to be accessed from static, needs to be static.

** Super cannot be referenced from a static content

** A static member can only access static data

* A static method can only access static data *

Why?

→ static method → call through class ⇒ 1 copy

we dont need Obj to access static members

* Static methods are only available to that class

Rules for Overriding –

1. Same method name & signature (Return type shd be covariant)
2. accessibility should be more or should be of same level (public etc)

3. Child cannot throw exception broader than parent's exception. It can be same or narrower than parent.
4. The return type of child should be able to derive from Parent - If a method returns a value it must be same or sub-class of parent

* [Covariant Return Types] *

[Every shiy is an obj, but every obj is not shiy]

↳ covariant

** should fulfil IS-A relationship **

Overriding → comes into picture only when there is Inheritance

Overloading → can be within same class & also can happen in the inherited class.

→ If the child is raising an exception, it should be either of same level exception as parent or narrower. But if parent is not raising any exception, then child can raise an exception.

27/3/25

Abstract class

→ Abstract class is a class which may not have any abstract methods.
↳ since they dont want instance to be created.

*

→ we cannot have a ~~method~~ abstract method without class being abstract.

→ we cannot have final & abstract in both method & class.

→ we cannot have body for abstract method ({}).

→ Abstract methods should not be private or final can be public or protected.

→ Abstract class cannot be instantiated.
we cannot create instance of abstract class.

→ Abstract class in C++ } ⇒ Making function as 'virtual' virtual function

→ If a concrete class inherits from abstract class consisting of abstract methods, then the concrete class should either ~~be~~ made as 'abstract' or the concrete class should implement the inherited abstract methods.

only meant for inheritance purpose only.

Polymorphism → same method call, diffn behaviour.
→ multiple forms with same method invocation but different behaviour.

→ Reference of abstract can be used to call concrete class (not sure).

* type & content *

type → type is checked at the compilation time

content → content is ~~can~~ at the time of runtime.

Interface ('implements' keyword)

→ We cannot create instance of interface.

→ In interfaces all methods need to be abstract, by default.

('extends' & 'implements')

method is default or static } → Implementation of these methods is done.

→ Interfaces used to ~~introduce~~ address multiple inheritance problem. (we can implement multiple interfaces)

can (or) cannot have abstract keyword defined

Interface extends Interface

→ Interface methods cannot have a body

{public & abstract} → Interface methods are these by default * (can be 'default' & 'static' too)

{Remote, Serializable} Marker Interface ** → Interface which does not have any methods implemented.

→ Interfaces cannot be final.

{public static final} → for variables *

↓

can access using ~~abstract class~~ interface name.

extends - class
implements - interface

→ Java cannot extend multiple classes but can implement multiple interfaces.

→ Once an interface is created, it cannot be modified by adding a method, unless that method is implemented by all the classes implementing that interface.

→ If we ~~want~~ dont want to break the code, but want to add a new method, then we can add that method using 'default' keyword. The method created by 'default' may be overridden in classes, else default implementation is done, without disturbing classes.
'Default' methods should have a body.
'default' & 'abstract' cannot go together.

→ here 'default' is interface method.

→ There are two interfaces (I_1 & I_2).
I_1 extends I_2. I_1 has default method, with some functionality. Now I_2 also has a method with 'default' & same method signature but different functionality. This will work.
↳ Means two interfaces having same default methods, can override within the interface.

```
Ione obj = new Test()
```

Ione ↙ Interface
- During compilation ↙ checks whether a method is present in the interface or not
- obj = new Test() → During Runtime ↳ content which will be displayed will be of this.

* 'Default' methods can be ~~overridden &~~ inherited & overridden.
static → class level.

* Static methods are only available through interface defn.

Compile Time Polymorphism and Run time polymorphism

- Compile Time Polymorphism
  - Static Binding
  - Ex- Overloading
- Run time polymorphism
  - Dynamic Binding

10/2/25

→ to Octal
→ to Hexadecimal

→ Constructor is a special fn since it has the same name as it's class
→ Objects cannot be created, if there is - No constructor.

→ access modif → public
→ private
→ protected
→ default (package privat)

→ In C++, if not used any keyword → private
In Java, if not used any keywords → default

→ In any good design OOPs related code, when declard any data member, there are always getters & setters function.

→ Getter and setter are public methods through which indirect accey to data members is given, protecting the data.

```
class Circle
{   float rad ——→ data member
    Circle (float rad)
    { radius = rad }

    float getRadius()
    { return rad; }

    float setRadius (float rad)
    { radius = rad;
    }
}
```

~~OOPS~~ ASK

* ⇒ Ask not what the system does, ask what it does it to

* [Ask not what the system does, ask what it does it to.] *

→ Do not bother about functionality of system. As what it does it to.

→ We focus on the data to capture, everything will be defined around the data.

→ use OOPs concepts while programming.

(Data once defined, other members can access it without re-declare again & again in their respective methods.)

[→ Define Data
→ Use setters & getters] *

(Data ++ Encapsulation is imported)

6) Can a constructor in a class be private?

→ Constructor can be private.

→

(we say no, because without ~~constructor~~ accessing constructor, objs cannot be created)

→ All instance variables have default values

short -
int -
double -
float - 0.0
long -

→ Default constructor won't take any arguments

→ Default constructor initializes the data members with default values.

→ We use getters and setters, in order to modify the default values.

⇒ Default constructor is a constructor with no arguments and no return type

→ Default constructor is by compiler.

⇒ Default constructor does not exist once the user defines a constructor with argument.

→ We have to write both the constructor, with argument and without arguments, in order to compile it correctly when we want to create objects of both the types (with args & without args)

→ Write a constructor with no arguments too along with constructor with args (good practice to write code).

⇒ If a function has a return type, it is not a constructor.

⇒ Constructor is a member function with the same name as class name and no return type.

Q) Why doesn't the constructor have a return type?

→ A constructor doesn't have return type is because every constructor returns ~~of type~~ ~~of its~~ an object of it's own type.

→ Constructors can be overloaded.

→ If we write a constructor, the default constructor won't exist anymore.
Means when created a constructor with args, we also need to create a constructor without args.

TO initialize object data → Constructors
→ Initialization Blocks [static Blocks] (Initializers)
→ [Instance Initialization Block]
→ Static Initialization Blocks

"Instance"
⇒ Initialization Blocks run ~~even before~~ an obj every time an obj is created

⇒ "Static" Initialization Blocks } only when class is loaded.

Instance Block → every time instance is created
Static Block → only when class is loaded.

→ All the static blocks execute, when class is loaded.

→ All instance blocks execute, ~~or~~ when an instance is ~~too~~ created.

Learn
what are class variables?
what are instance variables?

→ Non static variables cannot be accessed in static methods.

→ Printing an object in java calls the 'toString()' method.

(classname & hexadecimal representation of class

↳ when printed an object