PEPCODING

PURSUIT OF EXCELLENCE AND PEACE

MODULE

OOPS

+91 11 4019 4461

PepCoding, 3rd Floor, 15 Vaishali,
Pitampura Opposite Metro Pillar 347,
Above Karur Vysya Bank, New Delhi,
Delhi 110034

www.pepcoding.com

/pepcoding

OOPS

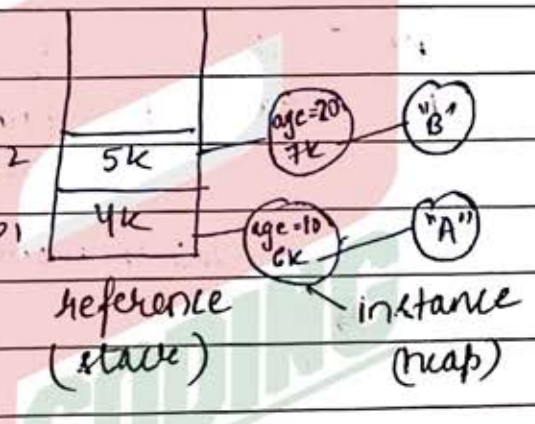
Encapsulation: Wrapping of data under single unit.

classes (data, functions) objects

(Mechanism that binds together code and the data it manipulates)

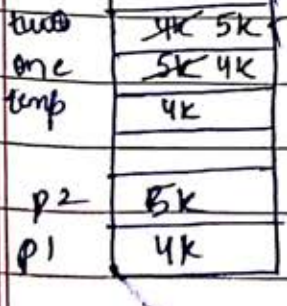
⇒ JAVA

```
person p1 = new person();
p1.age = 10
p1.name = "A"

person p2 = new person();
p2.age = 20
p2.name = "B"
```



Ques: Predict the changes when swapped like -

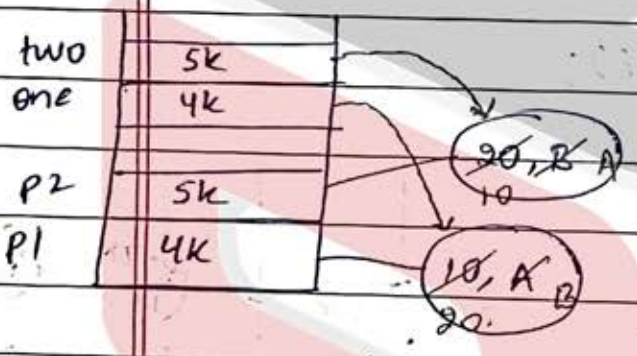
a)
```
swap ( person one, person two)
{   person temp = one
    one = two
    two = temp   }
```



As per the stack diagram, all the changes are limited to stack only.
Hence, no change ⇒ NOT SWAPPED!

b) swap (person one, person two)
{ String tn = one.name
one.name = two.name
two.name = tn

int tage = one.age
one.age = two.age
two.age = tage
}

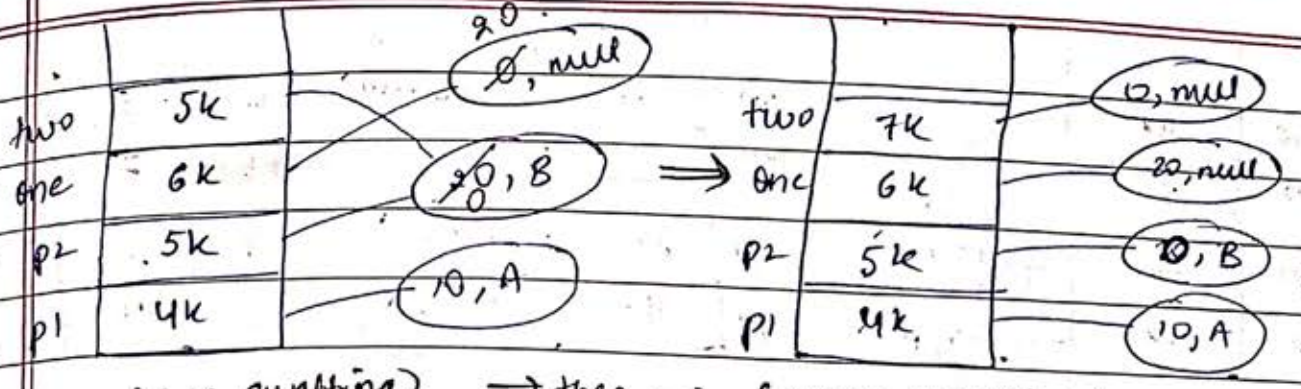


(swapping of values)
SWAPPED ! because changes are made directly on address of objects.

Since memory is accessed → hence permanent changes.

c) swap (person one, person two)
{ one = new person();
int tage = one.age
one.age = two.age
two.age = tage

two = new person();
String tname = one.name
one.name = two.name
two.name = tname
}



(age swapping) → then → (name swapping).

Output : p1 = 10, "A"

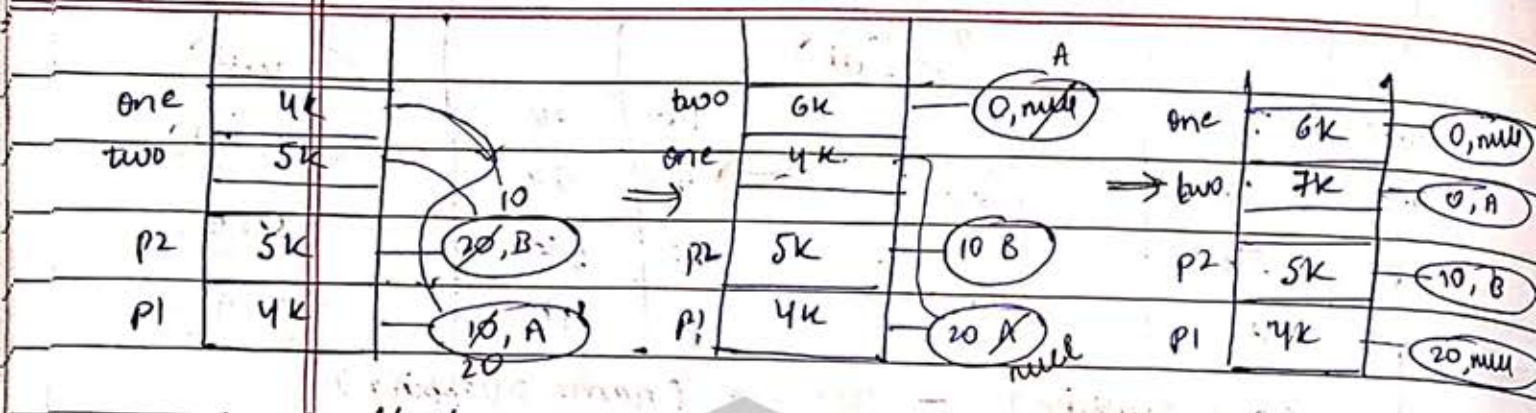
p2 = 0, "B"

Only age of p2 changes to 0 as "one = new person()" made 'one' to point to some new address 6k, hence original p2's age is swapped with new 'one' object.

Ily, no visible changes after "two = new person()".

d)

```
swap ( person one, person two)
{   int tage = one.age
    one.age = two.age
    two.age = tage

    two = new person()
String tname = one.name
 one.name = two.name
 two.name = tname

one = new person()  }
```



age swapped

name swapped

After dropping down, the memory of one and two will release and GARBAGE COLLECTOR will swipe out heap storage of 'one' and 'two'.

At last we will be left with -

p1 = 20 null

p2 = 10 B

(as per stack diagram)

e) swap (person one, person two)

{

int tage = one.age

one.age = two.age

two.age = tage

one = new person()

two = new person()

String tname = one.name

one.name = two.name

two.name = tname

}

Only age will get swapped

↳ p1 = 20 A

p2 = 10 B

⇒ C++

person p1 ; person p2 ;

p1. age = 10 ;

p1. name = "A" ;

p2. age = 20 ;

p2. name = "B" ;

String = size, capacity, data pointer

In C++, objects are made directly on stack.

(No concept of reference - instance)

Ques: Predict the changes when swapped like :-

1. swap (person one, person two)

{ person temp = one

one = two

two = temp

}

temp	10	A
two	~~20~~ 10	~~B~~ A
one	~~10~~ 20	~~A~~ B
p2	20	"B"
p1	10	"A"

changes made on stack but are not permanent as when function is completed, its effect disappears.

Hence, NOT SWAPPED !

But this kind of swapping is dangerous. Why ?

(i) If a string / vector / object is passed without reference in C++, then it is $\Theta(n)$ operation in place of $O(1)$ because when string is copied, its copy constructor will fire which will create a copy of your string at different addresses.

eg: person p1;
p2 = p1;

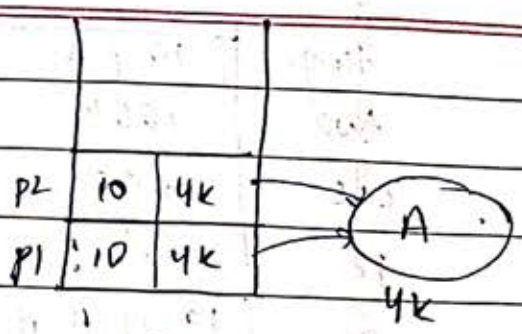
p2	10	5k → A
p1	10	4k → A

So, HEAVY STRINGS CAN MAKE THIS SLOW

(ii) Consider the following case:-

person p1;
p1.age = 10
p1.name = "A"

```
if (true)
{ person p2 = p1;
  break;
}
```



If copy constructor wouldn't be working the way it works, i.e. 2 objects pointing to same address, then it could cause the problem of DANGLING POINTER.

which means when we drop down and scope of 'if' ends, destructor will be fired, deleting all introductions of 'if' ⇒ p2 will be deleted, leaving behind a dangling pointer for p1.

⇒ RULE OF 3 ⟶ ~~constructor~~ operator =

↓ destructor

→ copy constructor

while writing the ~~destructor~~ in C++, you must write copy constructor and operator =, otherwise it will cause above mentioned problem.

2. swap (person ~~&p1~~ &one, person ~~&p2~~ &two)
{ // swap
}

(& ⇒ Hidden pointer)

temp	10 | A		
two	1004		
one	1000		
p2	~~20~~ 10 | ~~B~~ A	1004	
p1	~~10~~ 20 | ~~A~~ B	1000	

Since, this time address reference was received, hence SWAPPED!

⇒ Difference b/w copy constructor and assignment operator?

```
Person p1;
person p2;
person p3 = p2;   ← copy constructor
p3 = p1;          ← assignment operator
```

- If already exists - assignment operator (after clearing previous memory)
- If new allocation - simply copy - copy constructor

⇒ What if no copy constructor & destructor is written?

In this case, default copy constructor and default destructor will be used.

person p1
person p2 = p1

p2	20 | 4K	
p1	10 | 4K	

But nature of default destructor ⇒ it only removes pointer and itself but not material in heap.

i.e.

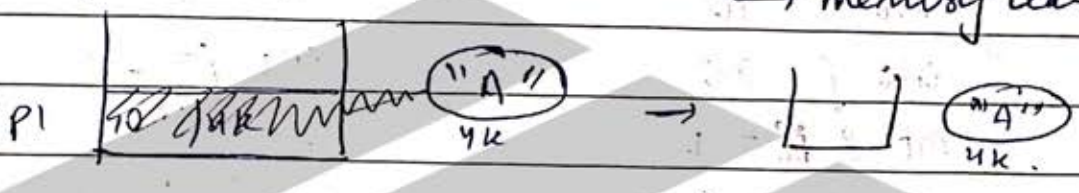
p2	~~20~~ | 4K	
p1	10 | 4K	

⇒

p1	10 | 4K	

But problem of MEMORY LEAK occurs when, if there arise a case when P1 vanishes (for example, its initialized in 'if', and scope of 'if' ends) then; P1 too will disappear leaving the material of heap intact

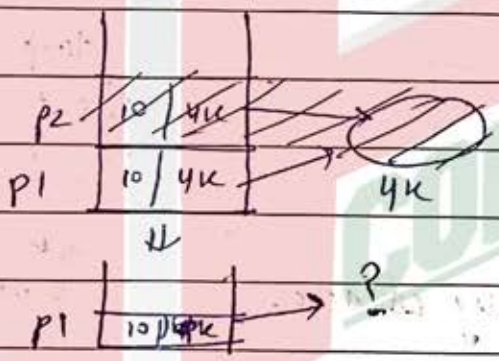
↳ memory leak.



So we need to write our own destructor
↳ but remember to write copy constructor and assignment operator along with it otherwise problem of DANGLING POINTER will arise.

person p1 ⇒

```
person
if(true)
{ person p2 = p1;
  break;
}
```



Only destructor will do this

* SO WE NEED DESTRUCTOR and OTHER TWO WHEREVER WE HAVE POINTER TO HEAP or ALLOCATION OF MEMORY ON HEAP

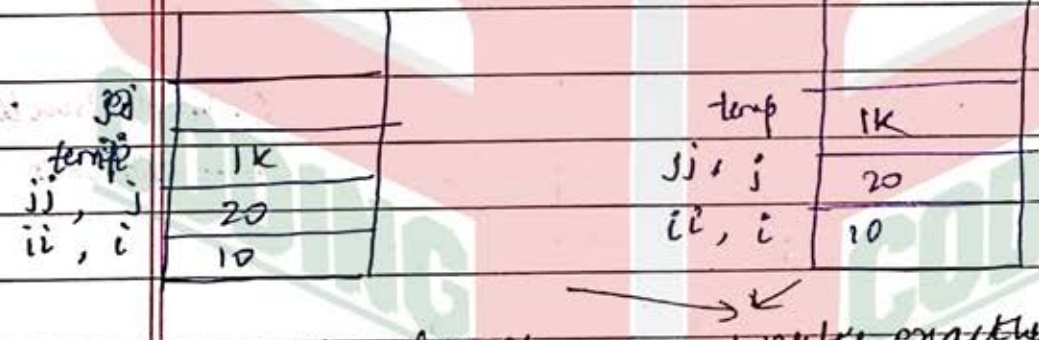
eg: wherever 'new' is used.

what if we use all the 3 properly?
Then performance will be bad as all copy operations will be of cost O(n) in place of O(1).

So, USE CALL BY REFERENCE in place of CALL BY VALUE for better performance.

Difference between "*" and "&":

REFERENCE	POINTERS
int i = 10	int i = 10
int j = 20	int j = 20
int & ii = i	int* ii = &i
int & jj = j	int* jj = &j
int temp = ii	int temp = *ii
ii = jj	*ii = *jj
jj = temp	*jj = temp;



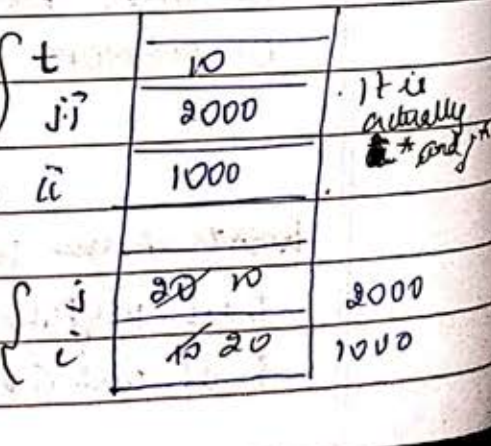
slightly cleaner

works exactly the same

"DUMMIES"/
"DUMMY POINTER"

example: fun (int &ii, int &jj)
{ int t = ii
ii = jj
jj = t }
main ( ) {
int i = 10
int j = 20
fun (i, j) }

(reference version)



→ Pointer version

```
fun ( int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
main ( ) {
    i = 10
    j = 20
    fun (&i, &j) ;
```

temp	10.		
b*	2000		
a*	1000		
j	~~20~~ 10	2000	
i	~~10~~ 20.	1000	

Both works exactly the same way, its just that '&' version is slightly easy to understand.

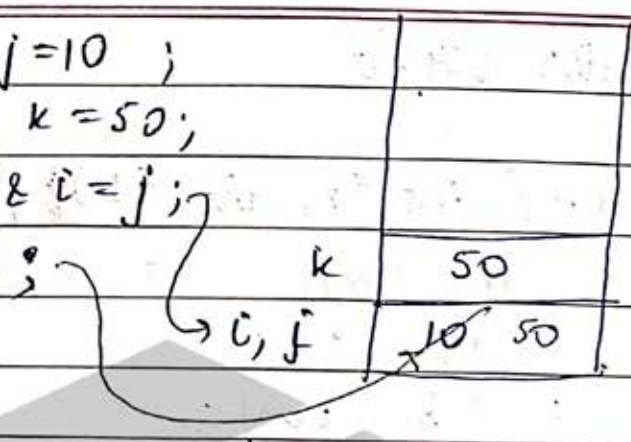
&a ⇒ prints address of 'a'

Constant Pointers

Reference vs Pointers

1) Cannot repoint them	1) Can repoint them
2) Cannot be null	2) Can be null
3) Needs to be initialized at declaration	3) ~~Have~~ Can be initialized later
4) Usage - convinient { primitive → normal, other → . }	4) Usage - tricky { primitive → *, other → '→' arrow }

ex → int j = 10 ;
int k = 50 ;
int & i = j ;
i = k ;



Important : 'i' just becomes another name for 'j'

NOTE :

i = 10	m	3000	
j = 20	k	1000	3000
int * k = & i	j	20	2000
int & l = i	l, i	10	1000
int ** m = & k			

l becomes another name of i

1) * k → prints value at add of k = 10
2) & k → prints address of k = 3000
3) cout << * l ⇒ WRONG ✗
4) cout << & l ⇒ 1000
5) cout << l++ ⇒ 11
6) l = j ⇒ l become 20
7) cout << m ⇒ 10

It only increments value at l

Segmentation fault → pointing to a memory which do not hold anything.

& k ⇒ address of k
* k ⇒ value at value of k

NOTE : (& i)++ is not allowed because its equivalent to incrementing an address

classmate
Date
Page

of 1000 → 1001

&i = &i+1

↳ error : expression must be a modifiable value.

→ Concept of repointing ?

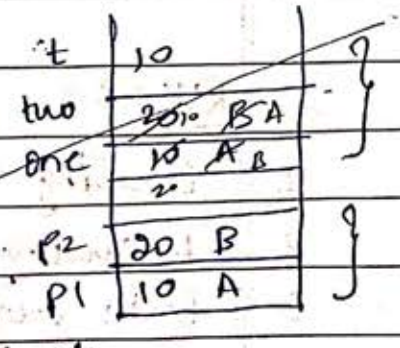
Reference :	Pointers :
A reference variable cannot be reassigned.	This property comes in handy when a developer is implemented data structure like linked lists, trees etc.
eg: int x = 5 ; int y = 6 ; int &r = x ;	eg: int x = 5 ; int y = 6 ; int *p ; p = &x ; ; p = &y ;

⇒ ~~Swapping Operations / ways~~ Is Swap possible ?

1) swap (person* one, person* two)
{ person t = one*
one* = two*
two* = t
}

Swap possible, working → just like reference version.

2) swap (person one, person two)
{ // swaping age
// swapping name seperately
}



NO SWAPPING POSSIBLE, since all changes only on stack

Date
Page

SUMMARY

Editor

⇒ [illegible]

[illegible]

Java [illegible]

INHERITANCE

⇒ JAVA :

parent class :

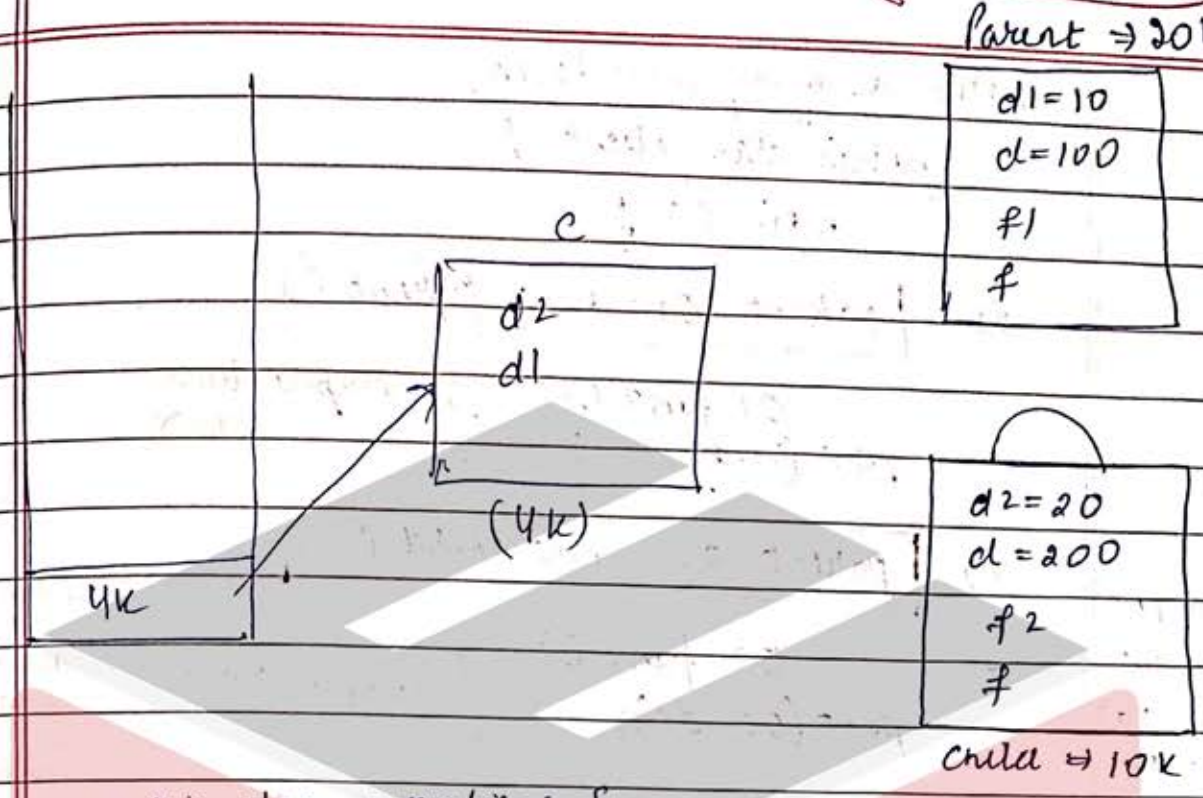
```
public class parent {
    int d1 = 10;
    int d = 100;

    void fun1 () {
        print ("parent fun1") }
    void fun ()
        { print ("parent fun") }
}
```

child class :

```
public class child {
    int d2 = 20
    int d = 200;
    void fun2 () { print ("child fun2") }
    void fun () { print ("child fun") } }
```

RULES :.. (IMPORTANT)

1. Editing → compiletime → Compiler → (LHS) or reference
2. Executing → Runtime → JVM → (RHS) or instance
3. Data members are always resolved on LHS



→ public class parentClient {

child c = new child (); // 4K

print (c.d1) } compile time error
~~print~~ c.fun1();

}

→ Now applying inheritance,

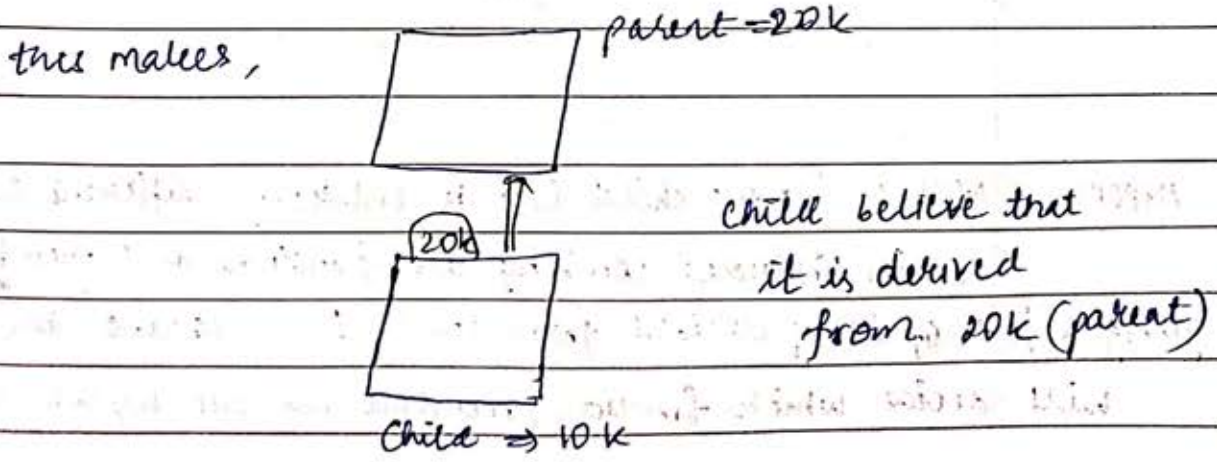
public class child extends parent {

int d2

int d

:

}



After applying inheritance,

```
public class client {
    main () {
```

1) `parent o1 = new parent ();`

```
o1.fun2 ();
print (o1.d2) ;
```
} compile time error

2) `parent o2 = new child ();`

d1 = 10
d2 = 20
c.d = 200
p.d = 100

→ `o2.fun2 ()`
`print (o2.d2);`
} compile time

because parent do not have access to child's members

→ `o2.fun()` → child's fun()
(// runtime decision based on instance and instance → child)
or
(// fun() found at child, then why go up ?)

→ `o2.fun1()` → parent's fun1()

→ `print (o2.d)` → parent's as data members are always allowed on LHS
`print (o2.d1)`
↓
parent's as d1 present only in parent

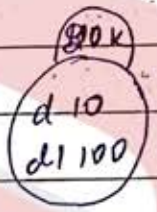
IMPORTANT NOTE : "new child ()" is entirely different event which includes combing all ; functions and members and giving a feel of being derived from child BUT "parent" (reference (LHS)) will decide which functions / members can the object use.

3) `child o3 = new parent();`

```
o3.fun();
o3.fun1();
o3.fun2();
print(o3.d)
print(o3.d1)
print(o3.d2)
```

} compile time error

Because child can never be reference to its base. Had it been allowed,



not able to find d2, f2()

BASICALLY, Child class's reference can access more elements/ members which are not present in parent/base class instance.

4) `child o4 = new child();`

```
o4.fun()  → child  (// because rule says instance)
o4.fun1() → parent (// obviously)
o4.fun2() → child
print(o4.d)  → child  (// rule says instance)
print(o4.d1) → parent (// obviously)
print(o4.d2) → child
```

}}

Is there any way to access data members and functions of child in the combination

parent o2 = new child ();

typecast o2 in child

↳ but this will alter the functionality/ or rule slightly.

NOW o2.d will of child rather than parent.

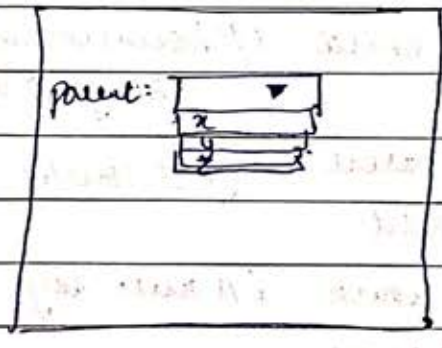
* There is no way to use values of common functions of parent while using combination -

child o = new parent();

RUNTIME POLYMORPHISM

If there are a number of instances available as option to a particular reference of an object, then runtime whichever instance is chosen, those functionalities can be used.

eg: drop down option application




```
Parent p = null;
If x chosen
    p = new parentX();
If y chosen
    p = new parentY();
If z chosen
    p = new parentZ();
```

classmate
Date
Page

Runtime Polymorphism → dynamic dispatch
→ dynamic polymorphism
→ dynamic binding
→ late binding

method overriding
or
polymorphism
same name, different functionality

instance is not known before and is known only at the runtime based on the instance that has been chosen.

COMPILE TIME POLYMORPHISM

Which instance to be chosen — decided by COMPILER.

METHOD OVERLOADING — at the time of compiling

eg: fun (int a, int t)
return a+t;

eg: fun (double a, double t)
return a*t;

Why developers have provided this feature of keeping the name same for different input / data types?

- To keep the API's compact
- to provide ease of use and coding to users and club all sorts of "SORT" together instead of sortBoolean, sortInt etc.

Date
Page

NOTE: Method overloading not to be done just on the basis of Return type as its not mandatory to receive values at the time of calling.

Invalid method overloading –

```
int fun (int, int)
string fun (int, int)
```

X

their argument type must be different

→ number

order type

OPERATOR OVERLOADING: Same operator, different meanings.

Not allowed in Java,

but still,

```
String s = "Hello"
s = s + "hello";
```

'+' is used for concatenation. How?

Because java uses `.append` whereve '+' is used.

IMPORTANT : { non-virtual, data members – difference
virtual – instance

classmate
Date
Page

⇒ C++ :

Polymorphism works only when base functions are VIRTUAL. (By default Java – all are virtual)

If these functions are not marked virtual, then EARLY BINDING will take place i.e. reference will return / use reference's function and will not wait for run-time (instance).

1. Parent* o1 = new Parent ();
   - o1.d → parent
   - o1.d1 → parent
   - o1.d2 → X
   - o1.f1 → parent
   - o1.f2 → X

2. Parent * o2 = new Child ();
   - o2.d1 ⇒ parent
   - o2.d2 = X
   - o2.f() → if virtual → child; if non-virtual → Parent
   - o2.f2 = X
   - o2.f1 = parent
   - o2.d = parent

Because non-virtual functions are as good as data members

3. Child* o3 = new Child ();
   - o3.d1 – parent
   - o3.d – child
   - o3.d2 – child
   - o3.fun – child
   - o3.f1 – parent
   - o3.f2 – child

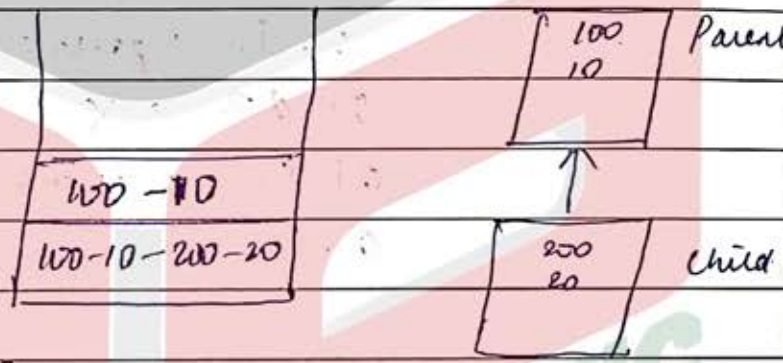
C++ rules summary

1) During editing → reference

2) During executing
   Non virtual ; data members → reference
   Virtual function → instance

3) Object slicing
   ↳ when pointers not used

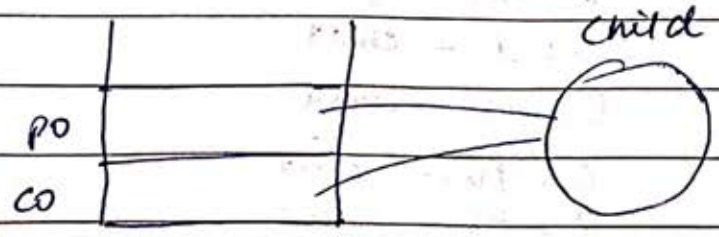
Child co ;
Parent po ;
↓
thus sliced the object co to 100-10



↳ separate object (smaller)

Now when 'd' is pointed or 'fun' is called parent's function / members will be called.

WHEREAS
Earlier when handling through pointers both po and co pointed towards the same instance and no separate objects were created.



Here, PD cannot access d2 and f2 but would give/use f of child.

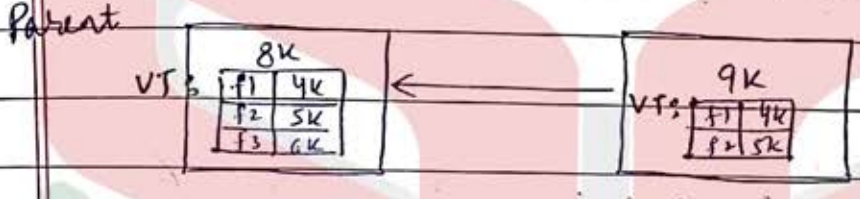
Virtual Pointers and Virtual table:

If f1, f2, f3 be virtual functions of class parent (4K) (5K) (6K) and

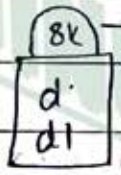
f1, f2 be virtual functions of class child (4K) (5K)

respective.

then these functions reside in VIRTUAL FUNCTION TABLE of respective classes and pointer to this virtual table is known as VIRTUAL POINTER.



When, Parent O1 = new Parent ()



— virtual pointer of parent which will use virtual f^n of parent only

IIly, for Parent O1 = new Child ()

ex→

```
class A {
    int d1
    func() {
    }
}
```

```
class B {
    int d1
    virtual func() {
    }
}
```

(Here sizeOf(A) means size of an object of class A)

Is sizeOf(A) < sizeOf(B) ?

YES, because object of B has hidden virtual pointer and hence its size is greater.

NOTE: Virtual pointer is called with the CONSTRUCTOR i.e. reference to virtual table is made when constructor of virtual function is ~~initialized~~ constructed.

* Without pointers, run-time polymorphism is not possible..

eg: → class A {
virtual fun() { }
}

sizeOf(A) = 8

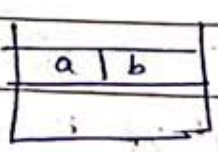
→ class A
{ int a;
virtual fun() { }
}

sizeOf(A) = 16
due to padding

4 used, 4 unused.

→ class A
{ int a;
int b;
virtual fun() { }
}

sizeOf(A) = 16



4-4 used

1) by, if one more 'int' introduced / initialised
sizeOf(A) = 24

Static data members

member → variable
→ data member

'STATIC' keyword → function

1) STATIC VARIABLE : (Only in C++)
↳ initialised and used within a function

eg: func()
{ int i = 10;
static int j = 20;
i++;
j++;
cout << i;
cout << j;
}
main()
{ func();
func();
func(); }

Output : 11 21
11 22
11 23.

Each time, i is initialised by 10 but 'j' is only initialised once, and gets incremented in next calls

because static variables are stored in DATA SEGMENT - separate space in heap.

Life ~~Scope~~ of static variable = life of program
Scope of static variable = function.

GLOBAL VARIABLES
→ Scope = entire program
→ life = life of program

⇒ A static variable is used as an instance variable

2) STATIC DATA MEMBER :

Ex:
```
class Bank                                   ( Memory Sharing )
{   int account no;
    int money;
    static double roi;                       // static data member
}
```

Bank bank1 = new Bank() { int acc.no; int money }
bank2 = new Bank()
bank3 = new Bank()

Each having its own value of account no and money but rate of interest for all banks will remain same.

If roi of a single bank change, roi for all banks will automatically.
This is an example of MEMORY SHARING.

Basically, "This memory is stored in class and not in general memory space" – Static data member (only in C++).

2) STATIC FUNCTIONS: These function can be called directly on class without creating objects.

* STATE OF AN OBJECT: Snapshot of data members inside it.

* STATELESS FUNCTIONS: Functions which are ~~it~~ can be used independently ~~from~~ and does not depend upon previous usage.
  Ex – Transaction on a shop etc, Math.pow ( )

* STATEFUL FUNCTIONS: Functions whose usage depends on previous usage / transaction.
  Ex – Bank Transactions
  ↳ depends on previous balance and so on.

- Can static function use 'this' pointer inside it?
  No, ~~because~~ since no objects are created for the access of static functions, there is no concept of "this" pointer here.

- Can static function use a non-static data member?
  No, because we need an object for calling and using non-static data members and no objects are created for static functions.

- Can a non-static ~~data~~ function use a static function / data ?

Yes, using non-static functions means object is created but for usage of static functions, no even this object is required, static functions can be used anyway.

4) STATIC CLASS :- Just like static functions can only access static data members, static class can be accessed through static functions without creating object of outer class.

ex →
```
Linked list class
{
    static class Node
    { int data; Node next }

    :

    static main()
    { }
}
```

'main' can ~~only~~ use 'Node' class only if it is marked static.

IMP: * Objects of static class can be easily made

```
Node* node = new Node();
```

~~Point~~ Only point to be noted is -

```
class LinkedList
{     Node class {
          int data;
          Node next;
      }
```

→ If Node class is static
Object of node can be made directly.

→ If Node class is non-static
To access Node, we need to first create object of linkedlist class and then access Node through that object.

(linkedlist object) `LinkedList ll = new LinkedList();`

(node object) `LinkedList.Node Node = ll.new Node();`

FINAL Keyword

⇒ JAVA

final → variable, data member

class function

1) VARIABLE: We cannot change value of these variables

2) DATA MEMBER: Same as variable but have rules for initialisation.

```
class A{
    final int a;
    A()
    {
    }
}
```

// needs to initialised either at declaration or in constructor else ERROR!

3) FUNCTION: Final functions are opposite of virtual functions.
You cannot OVERRIDE final functions once written i.e. you cannot even write the body of a function again which is marked FINAL which was otherwise VIRTUAL by default in JAVA.

```
class Parent
{ int d;
  final func() { }
}
class child
{ int d;
  // here func() cannot be written
}
```

4) CLASS : Final classes cannot be inherited

eg: final class P
{
}

then, C extends P ⇒ not allowed

⇒ C++

CONST keyword

- variable (CONST)
- data members (CONST)
- function (By default all C++ functions are like CONST final functions)

* There is nothing available in C++ to ~~achieve~~ achieve the impact of FINAL class

	Java	C++
1) Variable	Final	CONST
2) Data member	FINAL Initialization → parsing / → constructor	CONST Initialization → only parsing
3) FUNCTION	FINAL (Can't override)	By default C++ functions
4) Class	FINAL (can't inherit)	- Not available -

** Example of final classes in JAVA ?
String, Double, int etc.

OBJECT CREATION

Person p = new Person()

steps : [Space allocation] → [Parsing] → [Constructor] → [Returns "this" keyword]

(allocate memory for ~~non static~~ data members and initialise them with default values of language)

(Initialise them with provided values)

(Implement constructor)

(Object ready and returned through "this")

ex→ public class account {

int amount = 10;

int ano ;

static double roi = 0.04;

static int count=0;

account ()

{ count++;

ano = count; }

account (int amount)

{ this ();

this. amount+= amount ; }

Steps of Object creation

step 1: Space allocated for amount and ano and initialized with defaults

amount = 0
ano = 0

NOTE :

1) Jora defaults → space allocation

2) class creator → (Fixed defaults) → Parsing

eg: amount = 10 .

3) class creator → (Dynamic Defaults) → Default constructor

eg: static int count ++ ;

4) class user → (previous values) → Parametrised constructor

ABSTRACT Keyword.

Functions / Classes

JAVA

⇒ Abstract

Step 2: Initialising with fixed defaults - Parsing

```
amount = 10
ano = 0
```

Step 3: Dynamic defaults - default constructor

```
amount = 10
ano = 1
```

Passing values - parametrised values

```
amount = 10 + x
ano = 1
```

NOTE: Default constructor within parametrised constructor

```
account (int amt)
{ this();
  this.amount += amt }
```

EXAMPLE:

```
class A
{ int p=20;
  A(p)
  { pp=50
  }
}
```

This runs before

ABSTRACT KEYWORD

function (no body)

class (no object creation allowed)

+ If a function is abstract then it is necessary for its class to be abstract?
Yes, because abstract function means function has no body and non-abstract class means its class can be created. Now calling an empty function via through an object would make no sense - null pointer.

- It is necessary for an abstract class to have atleast one abstract function?
No, an abstract class may have all non-abstract functions. It hardly matters.

NOTE: If an abstract class is inherited by some non-abstract class, ~~either you provide defn~~ and all the functions of abstract class are abstract then, —
either you define all the functions of abstract class or make the derived class as abstract too.
and this will continue unless definition for all functions ~~are~~ is provided.

→ Difference between library and framework.

Library : Called and used by application only.
Framework : Cross calls - framework can call and use application as well.
Eg - Priority queue is a framework which calls our app using comparator.

- Where do we use abstract class?
TEMPLATE DESIGN PATTERN - A parent template class is created with abstract function which might ~~have~~ contain different logics for different applications.
⇒ Basically, it provides empty body / skeleton of an operation in terms of a number of high-level steps and leaves the details to be implemented by child class.
The overall structure and sequence of the algorithm is preserved by parent class.
Template - Preset format

- Can abstract class have constructor?
Yes, in the scenario of inheritance, to initialise the properties which may get derived down the chain.

eg→
```
abstract class A {
    p1
    abstract fun() { }
}
abstract class B {
    p2
}
abstract class C {
    p3
    fun() { }
}
```

In the given scenario, if body of fun() is defined in class C where

A
↑
B
↑
C

then we need to have constructor in abstract classes A and B to initialise p1 and p2.
Constructor will be called in the sequence →

class A →
```
A (p1) {
this.p1 = p1; }
```

class B →
```
B (p1, p2)
{ super (p1)          // super means constructor of class just above it.
this.p2 = p2;
}
```

class C →
```
C (p1, p2, p3)
{ super (p1, p2)      // if parametrized constructor super(p1,p2) is not called, then its default constructor will be fired automatically.
this.p3 = p3 }
```

NOTE: `super(p1, p2)` ≠ `this.p3 = p3`
`this.p3 = p3` ≠ `super(p1, p2)` } THIS IS WRONG!

Date
Page

Constructor chaining : In an inheritance scheme of type →

A
↑
B
↑
C

constructor will be fixed in sequence ⇒ A → B → C
whereas destructor sequence → C → B → A

Abstract keyword → Abstract (Java)
→ Pure virtual (C++)

⇒ Example of constructor chaining -

```
class A {
    A()
    { print (A) };
}
```

```
class B {
    B()
    { print (B) }
    A a = new A() }
```

```
class C {
    C()
    print(C) }
    B b = new B();
```

```
class D {
    D()
    print (D) }
    C c = new C(); }
```

```
main() {
    D d = new D();
```

OUTPUT ⇒
A
B
A
C
A
B
A
A
B
A
C
A
B
A

ITERATOR

Consider the question of finding pair sum equal to 100 in binary tree.

Approach	BST	Binary	Space
1) Recursion in recursion ('find' function)	nlogn	n^2	O(h)
2) Sorted array/list	n (Inorder of BST)	nlogn	O(n)
3) Hashmap	n	n	O(n)
4) Iterator (recursion from both side = "freezable")	n	n	O(h)

INTERFACE: Pure abstract class with all body-less functions and no data members.

Interface → implements
Abstract class → extends

NOTE:
- Interface may have data members but of STATIC type.
- C++ does not have interfaces.

- Use of interface in Java.
Since java doesn't support multi-inheritance, so to achieve that purpose/functionality - interfaces can be used.

ITERATOR Vs ITERABLE (JAVA)
Both of them are interfaces

⇒ Iterable - to be implemented by our main class (BST)
↳ It has a single function "iterator()"
↓ returns
ITERATOR
(Iterator is again an interface)

Since we need to return an Iterator on the iterator() function, so we will be writing a BSTIterator class and it will have 2 functions of ITERATOR interface
↙ ↘
hasNext() next()

So basically,
1) Our class implements Iterable<> { }
It's function → iterator() will return an Iterator named bstiterator

2) class bstiterator will have 2 functions
↙ ↘
hasNext() next()

Algorithm for applying iterator on BST

USING STATES :- for inorder traversal

state 0 → push in stack and
↓
state 1 → push left
↓
state 2 → print and push right
↓
state 3 → pop from stack

To implement the above, observers apply iterators on pair class,

```
class Pair {
    Node node;
    int state;
    Pair (Node node, int state)
    { this.node = node;
      this.state = state;
    }
}
```

~~For implementing iterator on BST,~~
For iterating BST, we need two iterators

- BstIterator (front order)
- ReverseIterator (reverse order)

Date
Page

```
public Iterator <Integer> iterator ()
    return new bstIterator (root);

static class bstIterator implements Iterator <Integer>
{   static class Pair {
        Node node
        int state; }

    Stack<Pair> st = new Stack ();
    int val;

    bstIterator ( Node node) {
        if (node != null) {
            st.push ( new Pair ( node, 0));
            next (); }
    }

    public boolean hasNext ()
        return val != -1;

    public Integer next ()
{       int rv = val;
        val = -1;

        while (stack.size > 0) {
        Pair top = st.peek ()
        if ( top.state == 0)
            ↳ add top.left if it exists
              and state ++
```

else if (state == 1)

↳ state++;

val = top.node.data

BREAK;

else if (state == 2)

↳ state++

add top.node.right if et exists

else if (state == 3)

↳ st.pop();

}

return rv;

⇒ Similarly, in reverse Iterator.

↳ only difference ⇒

at state = 0

↳ add right

at state = 2

↳ add left

How to use these iterators. for finding pair/sum equal to k?

Iterator<Integer> itr = tree.iterator();

Iterator<Integer> ritr = tree.riterator();

```
int left = itr.next();
int right = itr.next();
while (left < right)
{
    if (left + right == sum)
        print & increment both and print
    else if (left + right < sum)
        increment left
    else
        right = itr.next();
}
```

HASHCODE and EQUALS

Needed when our custom object is used as key in hashmap.

Hashcode → which bucket to choose

equals → decides whether the given value is matching with any of the element present in that bucket or not.

Consider the problem of finding ~~maximum~~ count of maximum no. of points lying on a straight line.—

```
class Line {
    int dx;
    int dy;
}
```

Here class 'Line' objects will be used as 'keys' ~~for~~ of ~~the~~ hashmap.

Hence, overriding HASHCODE and EQUALS functions -

⇒ public boolean equals (Object o)
{ Line lo = (Line) o; (typecasting of object)

if (this.dx == 0 && this.dy == 0)
return true;
else if (this.dy == 0 && o.dy == 0)
return true;
else if (this.dx == o.dx && this.dy == o.dy)
return true;
return false;
}

⇒ public int hashcode ()
{
return this.dx + this.dy;
}

TO STRING FUNCTION (toString())

→ toString() method returns the string representation of the object.
It is overrider to print the desired output of custom class.

example:-

```
class student {
    int rollno;
    String name;
    String city;

    student (int r, String n, String c)
    { this.rollno = r;
      this.name = n;
      this.city = c;
    }
```

Now; if

```
student s1 = new student ("100", "ABC", "GOA");
```

and we want

"System.out.print(s1)" to print rollno, name and city in sequence, then we need to override toString function like -

```
public String toString()
{ return rollno + " " + name + " " + city; }
```

Otherwise it will print the hashcode values of these objects.