Hashing 1

Today's Agenda:

* Hashmap & Hashset functionality
* Question on Hashing.

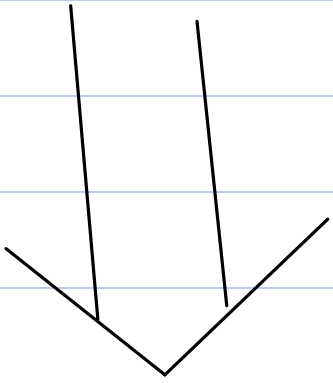
Hotel Business

R1 R2 R3

R4 R5

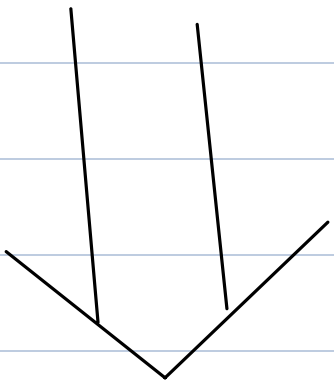
Register :

Rooms	Occupancy
R1	Occupied
R2	Occupied
R3	Vacant
R4	Occupied
R5	Vacant



1000 rooms

Array of size 1001



Room no.s are still 1000 but numbering to be done randomly between $[1, 10^3]$

Eg. 100035, 28, 34586

$$10^6 = 10^3 * 10^3$$

$$= 4 * 10^3 * 10^3$$

⇒ Create an array of size 10^9.

Problem → Memory Wastage (only 1000 rooms are there).

Solutions: Hashmap | Map | Dictionary.

→ Create a key value pair value for each room.

Key	Value
100035	vacant
⋮	
28	occupied
⋮	
34586	occupied

Advantages:

1) space → 10^3

2) Adding new room (a new key value pair) → constant time O(1)

```
Hashmap<Integer, String> hm;
```

3) Accessing availability of a room → constant time O(1)

Why? Constant time

Will be having dedicate session for it

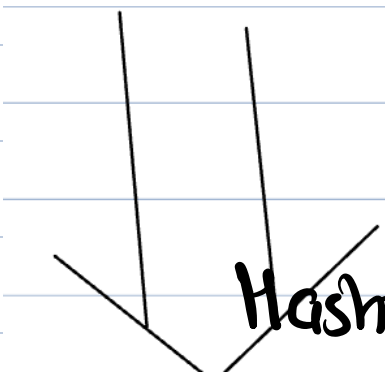
Quiz 1: Population for Every country.

String		Long
China	→	1,300,000,000
India	→	1,400,300,000
Nepal	→	300,000

HashMap<String, Long> hm

Quiz 2: No. of state of every country.

String		Int
India	→	29
Pakistan	→	10
USA	→	50
China	→	15



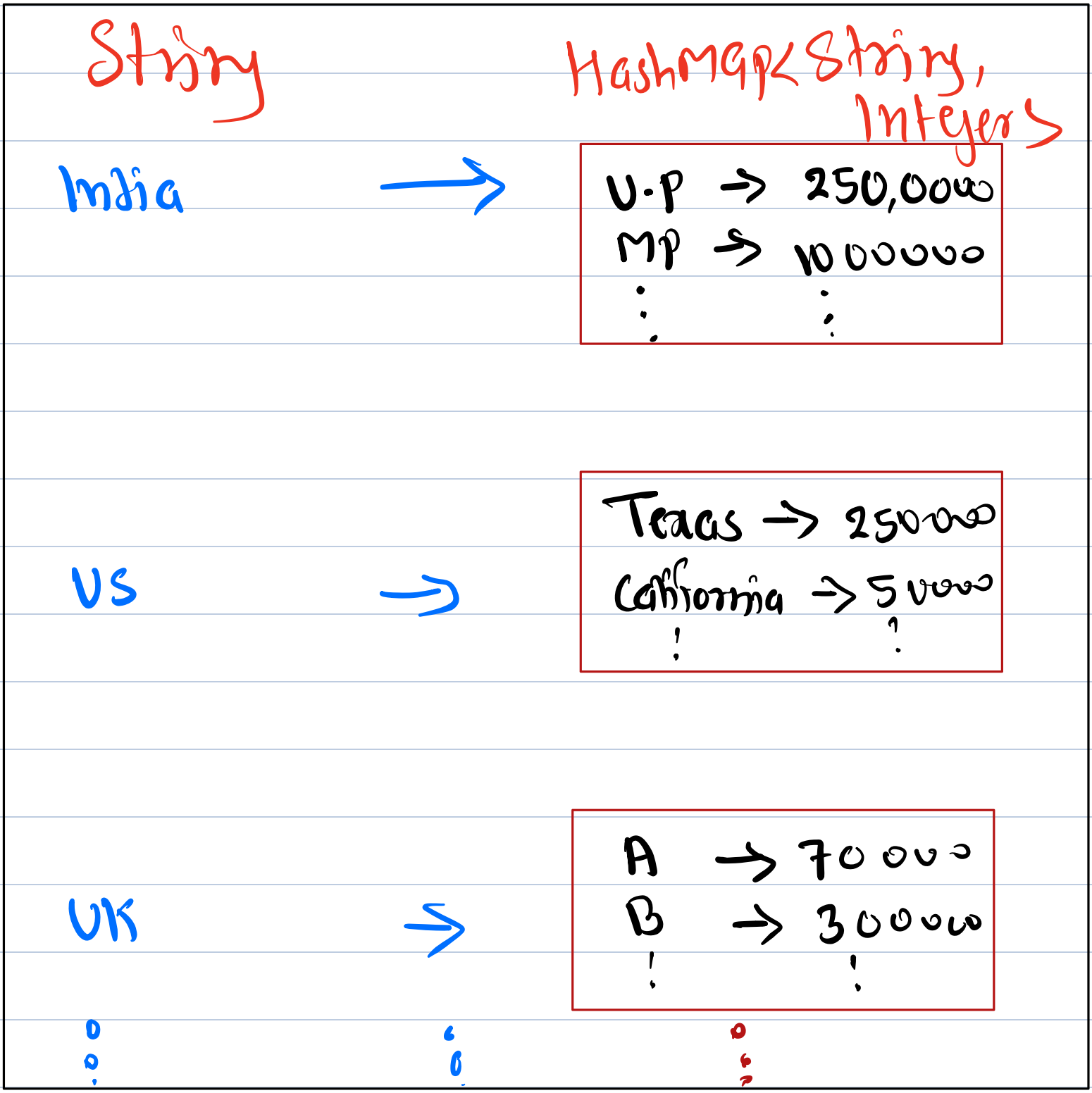
HashMap<String, Integer> hm

Quiz 3: Name of all state of every country.

String	List<String>
India →	UP, MP, Goa .....
USA →	California, Texas .....
Pakistan →	Karachi, Lahore .....
UK →	A, B, C .....

HashMap<String, List<String>> hm

Quiz 4: Population of each state in every country.



Properties:

1) Keys are unique.
2) Values can be anything.
3) Searching will take constant time.

* If we are only interested in unique keys, value are irrelevant.
  → Use Hashset

HashMap

- **INSERT(Key,Value):** new key-value pair is inserted. If the key already exists, it does no change.
- **SIZE:** returns the number of keys.
- **DELETE(Key):** delete the key-value pair for given key.
- **UPDATE(Key,Value):** previous value associated with the key is **overridden** by the new value.
- **SEARCH(Key):** searches for the specified key.

HashSet

- **INSERT(Key):** inserts a new key. If key already exists, it does no change.
- **SIZE:** returns number of keys.
- **DELETE(Key):** deletes the given key.
- **SEARCH(Key):** searches for the specified key.

Time Complexity of **all the operations** in both Hashmap and Hashset is **O(1)**.

Hashing Library Names in Different Languages

	Java	C++	Python	Js	C#
Hashmap	Hashmap	unordered_map	dictionary	map	dictionary
Hashset	Hashset	unordered_set	set	set	Hashset

Implementations :

	HashMap	HashSet
Java	HashMap<Integer, Integer> map = new HashMap<>();	HashSet set = new HashSet<>();
C++	unordered_map<int, int> map;	unordered_set set;
Python	dictionary = {}	set = set()
Js	let map = new Map();	let set = new Set();
C#	Dictionary<int, int> dict = new Dictionary<int, int>();	HashSet set = new HashSet();

Q) Given N elements and Q queries. find the frequency of elements provided in the query.

	0	1	2	3	4	5	6	7	8	9	10
A[] :→	2	6	3	8	2	8	2	3	8	10	6

Queries

2 → 3
8 → 3
3 → 2
5 → 0

BF: For each query iterate over an array and count its occurence.

TC: O(N*Q)
SC: O(1)

Optimisation: We use Hashmap.

	0	1	2	3	4	5	6	7	8	9	10
A[] :→	2	6	3	8	2	8	2	3	8	10	6

2 → ~~1~~ ~~2~~ 3
6 → ~~1~~ 2
3 → ~~1~~ 2
8 → ~~1~~ ~~2~~ 3
10 → 1

HM

HashMap<Int, Int>

⇒ first create Hashmap and store freq. of each element.
⇒ Iterate over query and check freq in Hashmap in O(1)

Code:

```
Hashmap<Int, Int> hm

for(i=0; i<N; i++)
{
    if (hm.search(A[i]) == true)
    {
        hm.update(A[i], hm.get(A[i])+1);
    }
    else
    {
        hm.insert(A[i], 1);
    }
}

for (i=0; i<q; i++)
{
    if (hm.search(Q[i]) == true)
    {
        Print (hm.get(Q[i]);
    }
    else
    {
        Print (0);
    }
}
```

TC: O(N+Q)

SC: O(N)

Q.2) Given N elements, find the first non repeating element.

Eg.

N = 6 { 1, 2, 3, 1, 2, 5 } → 3

N = 8 { 4, 3, 3, 2, 5, 6, 4, 5 } → 2

N = 7 { 2, 6, 8, 4, 7, 2, 9 } → 6

Brute force: For each query iterate over an array and find it's frequency The first element found with no repeating occurance is the answer.

TC: $O(N^2)$

SC: $O(1)$

N = 8 { 4, 3, 3, 2, 5, 6, 4, 5 } → 2

4:2 2:1 6:1
3:2 5:2

Optimisation:

1) Create a freq. Hashmap like previous question.

2) Iterate on the original array and check it's freq. The first element having freq. as 1 would be answer.

Code:

```
Hashmap<Int, Int> hm;
for(i=0; i<N; i++)
{
    if(hm.search(A[i]) == true)
    {
        hm.update(A[i], hm.get(A[i])+1);
    }
    else
    {
        hm.insert(A[i], 1);
    }
}
for(i=0; i<N; i++)
{
    if(hm.get(A[i]) == 1)
    {
        return A[i];
    }
}
return -1;
```

TC: O(N)

SC: O(N)

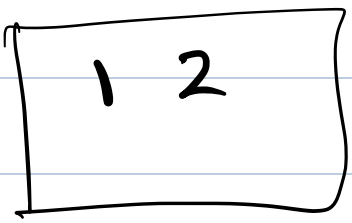
Break: 10:20

Q.3) Count of distinct elements.

N = 5 ⇒ { 3, 5, 6, 5, 4 } → 4

N = 3 ⇒ { 3, 3, 3 } → 1

N = 5 ⇒ { 1, 1, 1, 2, 2 } → 2

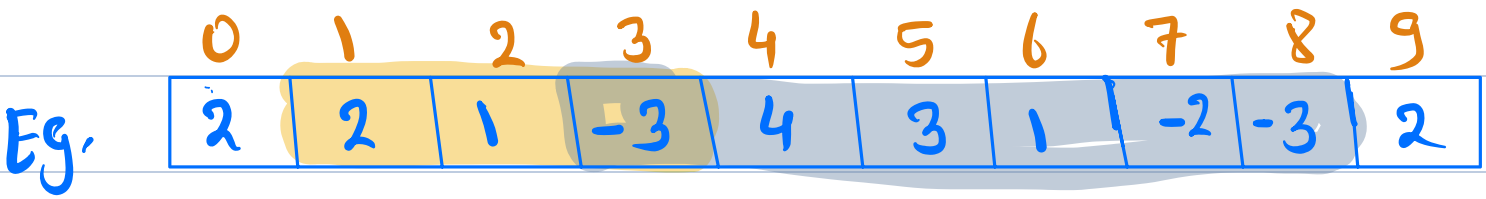


Approach: Create Hashset and store every element in Hashset, The size of Hashset will give the total no. of unique numbers.

```
Hashset<Int> hs;
for(i=0; i<N; i++)
{
    hs.insert(A[i]);
}
return hs.size();
```

TC: O(N)
SC: O(N)

Q.4) Given an array of N elements, check if there exists a subarray with sum equal to 0.



BF Approach: Consider all the subarrays.

```
for (s=0 ; s<N; s++)
{
    for (e= s; e<N; e++)
    {
        sum =0
        for(k=s ; k<=e ; k ++)
        {
            sum = sum + A[k];
        }
        if (sum = =0) return true;
    }
}
return false;
```

Tc: $O(N^3)$
Sc: $O(1)$

* using PF sum or CF → $O(N^2)$

Optimisation

	0	1	2	3	4	5	6	7	8	9
	2	2	1	-3	4	3	1	-2	-3	2
PF	2	4	5	2	6	9	10	8	5	7

Eg. {-2, -1, 3, 5}

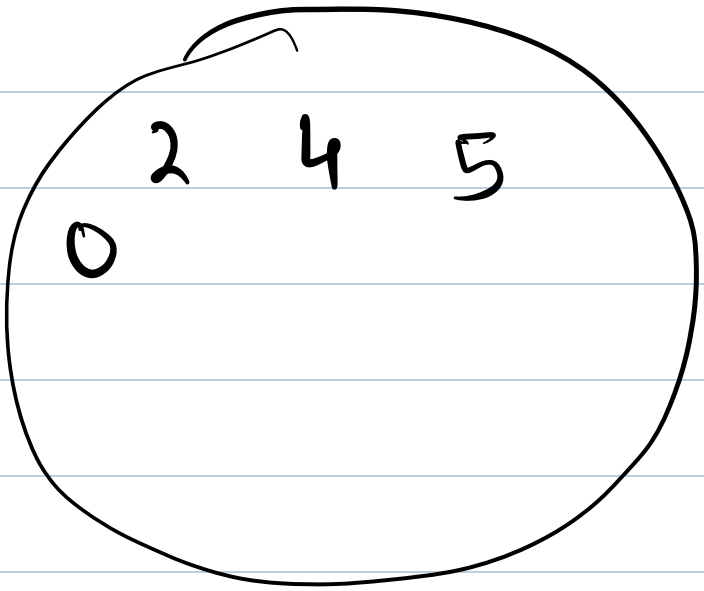
PF {-2, -3, 0, 5}

Code:

```
// Create Prefix Array.

HashSet<Int> hs;
hs.insert(0)

for(i=0; i<PF.length; i++)
{
    if(hs.search(PF[i]) == true)
    {
        return true;
    }
```

```
else
{
    ns.insert(PF[i]);
}
}
return False;
```

2	4	5	2	6	9	10	8	5	7



output: true.

TC: O(n)
SC: O(n)

HW: Try to solve without using PF.

Q.5)

SCALER has a series of contests to help learners improve their coding skills. Each learner tries these contests to practice, but not everyone participates the same amount. **SCALER** wants to know which learner is participating the least so they can help them more or figure out how to get them more involved.

Problem

You have a list of id where each id represents a learner who tried a contest. A learner's number shows up more times if they try more contests. Your job is to write a program that looks at this list and finds out who has tried the **least number of contests**.

A[] = {101, 102, 103, 101, 102, 101, 104, 105, 106, 105, 105}

Learner Id	Contest Tried
101	3
102	2
103	1
104	1
105	3
106	1

Output: 3

1) Frequency map where key -> Learner Id
value and value -> Contest attempted.

2) find the minimum count.
3) no. of entry/key having that minimum count.

Code:

```
HashMap<Integer, Integer> hm = new HashMap<>();

for(i=0; i<N; i++)
{
    if( hm.containsKey( A[i] ))
    {
        hm.put(A[i], hm.get(A[i])+1);
    }
    else
    {
        hm.put(A[i], 1);
    }
}
```

```
int minCount = Integer.MAX_VALUE;
int ans=0

for ( int tries : hm.value())
{
    if ( minCount > tries)
    {
        minCount = tries;
        ans = 1
    }
    else if (tries == minCount)
    {
        ans++;
    }
}
return ans;
```

Learner Id	Contest Tr
101	3
102	2
103	1
104	1
105	3
106	1

→

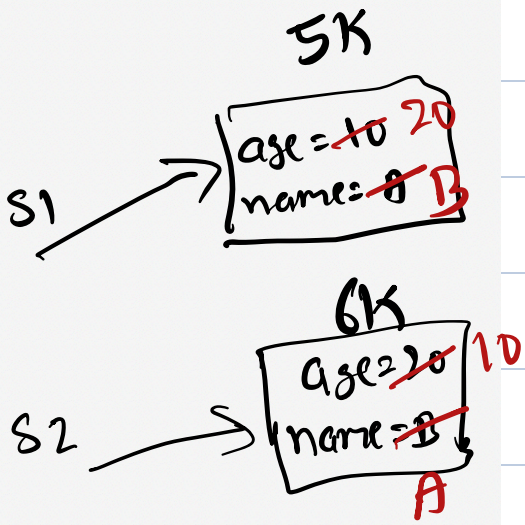
minCount = ~~∞~~ MAXVAL 3 2 1

ans = X X 3

```
counts = {}
for id in A:
    if id in counts:
        counts[id] += 1
    else:
        counts[id] = 1

minCount = float('inf')
ans = 0
for tries in counts.values():
    if tries < minCount:
        minCount = tries
        ans = 1
    elif tries == minCount:
        ans += 1
```

Doubt Session

```
class Student {
   int age;
   String name;

   void display(){
       System.out.println("My name is " + this.name + ". I am "  + this.age + " years old");
   }

   void sayHello(String name){
       System.out.println(this.name + " says hello to " + name);
   }
}

public class Client {
   public static void main(String[] args) {
       Student s1 = new Student();
       s1.age = 10;
       s1.name = "A";

       Student s2 = new Student();
       s2.age = 20;
       s2.name = "B";

       swap(s1, s2);

       s1.display();
   }

   private static void swap(Student s1, Student s2) {
       int tage = s1.age;
       s1.age = s2.age;
       s2.age = tage;

       String tname = s1.name;
       s1.name = s2.name;
       s2.name = tname;
   }
}
```

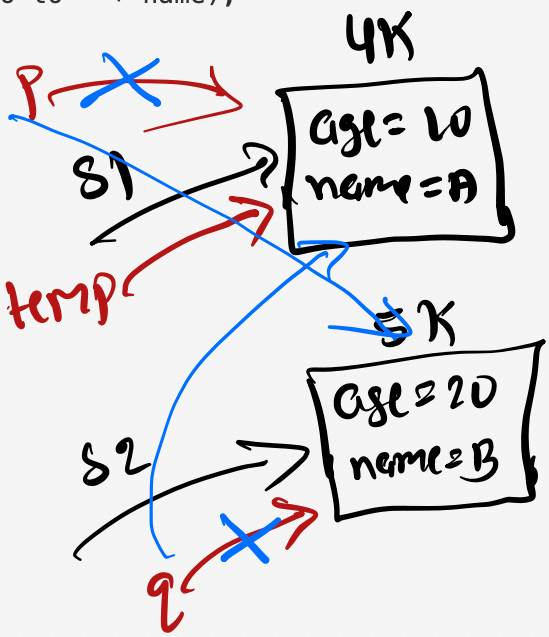


5K 6K

tage = 10
s1.age = 20
s2.age = 10

tname = A
s1.name = B
s2.name = A

```java
public class Student {
    int age;
    String name;

    void display(){
        System.out.println("My name is " + this.name + ". I am "  + this.age + " years old");
    }

    void sayHello(String name){
        System.out.println(this.name + " says hello to " + name);
    }
}

public class Client {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.age = 10;
        s1.name = "A";

        Student s2 = new Student();
        s2.age = 20;
        s2.name = "B";

        swap(s1, s2);

        s1.display();
    }

    private static void swap(Student s1, Student s2) {
        Student temp = s1;
        s1 = s2;
        s2 = temp;
    }
}
```



p q

temp = 4K

p -> q

q -> p

Shallow Copy

```
public class P {
   protected int d1 = 10;
   protected int d = 100;

   protected void fun1(){
       System.out.println("P's fun1");
   }

   protected void fun(){
       System.out.println("P's fun");
   }

   static protected void sfun(){
       System.out.println("P's sfun");
   }
}

public class C extends P{
   protected int d2 = 20;
   protected int d = 200;

   protected void fun2(){
       System.out.println("C's fun2");
   }

   protected void fun(){
       System.out.println("C's fun");
   }

   static protected void sfun(){
       System.out.println("C's sfun");
   }
}

public class Client {
    public static void main(String[] args) {
        P obj = new C();
        System.out.println(obj.d1);
        System.out.println(obj.d);
        obj.fun1();
        obj.fun();
        obj.sfun();
    }
}
```