**Explain the internal working of HashMap (or HashSet).**

1. How is hash collision handled?

* It can be handled by two ways:

i)Separate Chaining

ii)Open Addressing

**Separate chaining** is defined as a method by which linked lists of values are built in association with each location within the hash table when a collision occurs.

**Open addressing** is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself

1. What is load factor?

* The Load factor is a measure that decides when to **increase** the HashMap capacity to maintain the get() and put() operation complexity of **O(1)**. The default load factor of HashMap is **0.75f** (75% of the map size).

1. Explain resizing / rehashing.

* **Rehashing means hashing again**. Basically, when the load factor increases to more than its pre-defined value (default value of load factor is 0.75), the complexity increases. So to overcome this, the size of the array is increased (doubled) and all the values are hashed again and stored in the new double sized array to maintain a low load factor and low complexity.

**Design custom String class. Why is it an immutable class?**

public final class Student

{

    final String name;

    final int regNo;

    public Student(String name, int regNo)

    {

        this.name = name;

        this.regNo = regNo;

    }

    public String getName()

    {

        return name;

    }

    public int getRegNo()

    {

        return regNo;

    }

}

Immutable class means that once an object is created, we cannot change its content. In Java, all the [wrapper classes](https://www.geeksforgeeks.org/wrapper-classes-java/) (like Integer, Boolean, Byte, Short) and String class is immutable. We can create our own immutable class as well.

Following are the requirements:

* The class must be declared as final (So that child classes can’t be created)
* Data members in the class must be declared as final (So that we can’t change the value of it after object creation)
* A parameterized constructor
* Getter method for all the variables in it
* No setters(To not have the option to change the value of the instance variable)

**Design custom ArrayList.**

A custom ArrayList can have multiple types of data and its attributes in general are based on the user requirements.

class CustomArrayList

{

    int n=4;

    class Data

    {

        int roll;

        String name;

        int marks;

        Data(int roll, String name, int marks)

        {

            this.roll = roll;

            this.name = name;

            this.marks = marks;

        }

    }

    public static void main(String args[])

    {

        int roll[] = {1, 2, 3, 4};

        String name[] = {"Shubham", "Atul", "Ayush", "Rupesh"};

        int marks[] = {100, 99, 93, 94};

        CustomArrayList custom = new CustomArrayList();

         custom.addValues(roll, name, marks);

    }

**Explain internal working of TreeMap / TreeSet. Which data structure is utilized internally and why?**

The **TreeSet** class **internally** uses a TreeMap to store elements. The elements in a **TreeSet** are sorted according to their natural ordering. ... The elements in a **TreeSet** are sorted as per their natural ordering, or based on a custom Comparator that is supplied at the time of creation of the **TreeSet**

TreeSet is basically implementation of a self-balancing binary search tree like [Red-Black Tree](https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/). Therefore operations like add, remove and search take O(Log n) time. And operations like printing n elements in sorted order takes O(n) time.

Public class TreeMapTest {

Public static void main(String args[]){

Map<key,String> treemap= new TreeMap<>();

treemap.put(new key(“Anamika”),Anamika);

treemap.put(new key(“Rushika”),Rushika);

treemap.put(new key(“Dinesh”),Dinesh);

treemap.put(new key(“Arnav”),Arnav);}

public key implements Comparable{

final int data=12;

Private String key;

Public key(String key){

Super();

this.key=key;}

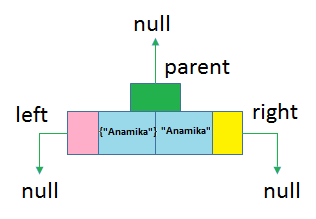
@override

Public int compareTo(key obj){

Return key.compareTo(obj.key);}

**Step 1:** Initially when we create a TreeMap object .

**Step 2: Adding the first element into TreeMap**

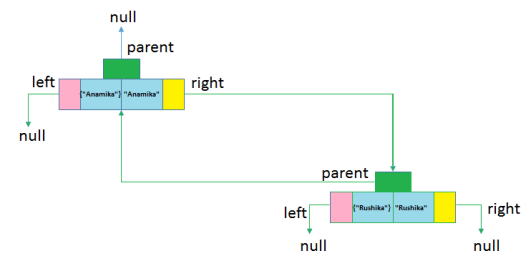


**Step 3: Adding the second element into TreeMap**

Now, {“Rushika”} is logically greater than {“Anamika”} and hence according to our rules,

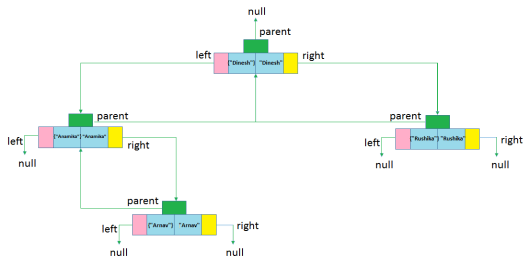
* {“Rushika”} will be placed to the right of {“Anamika”}.
* {“Anamika”} will be a parent of {“Rushika”}.

After inserting this element, the structure for TreeMap becomes as below.

. 

**Step  4: Adding third element into TreeMap.**

So {“Arnav”} is the first key object being inserted as key. After adding this element, a structure for TreeMap becomes as below.



**Explain working of**

1. **CopyOnWriteArrayList : CopyOnWriteArrayList** is a thread-safe variant of ArrayList in which all mutative operations (add, set, and so on) are implemented by making a fresh copy of the underlying array. It's immutable snapshot style iterator method uses a reference to the state of the array at the point that the iterator was created.

CopyOnWriteArrayList<Integer> list = new CopyOnWriteArrayList<>(new Integer[] {1,2,3});

1. **CopyOnWriteArraySet:** CopyOnWriteArraySet is a Set that uses an internal [CopyOnWriteArrayList](https://www.geeksforgeeks.org/copyonwritearraylist-in-java/) for all of its operations. It is introduced in JDK 1.5, we can say that it is thread-safe version of Set.

Multiple Threads are able to perform update operation simultaneously but for every update operation a separate cloned copy is created. As for every update a new cloned copy will be created which is costly. Hence if multiple update operation are required then it is not recommended to use CopyOnWriteArraySet.

1. **Deque:** Java Deque is an interface you need to instantiate a concrete implementation of the interface in order to use it. You can choose between the following Deque implementations in the Java Collections API:

* java.util.LinkedList
* java.util.ArrayDeque

The LinkedList class is a pretty standard Deque and Queue implementation. It uses a linked list internally to model a queue or a deque.

The Java ArrayDeque class stores its elements internally in an array. If the number of elements exceeds the space in the array, a new array is allocated, and all elements moved over. In other words, the ArrayDeque grows as needed, even if it stores its elements in an array.

1. **IdentityhashMap:** IdentityHashMap implements Map, Serializable and Clonable interfaces and extends AbstractMap class.  
   This class is not a general-purpose Map implementation. While this class implements the [Map](https://www.geeksforgeeks.org/map-interface-java-examples/) interface, it intentionally violates Map’s general contract, which mandates the use of the equals method when comparing objects.

IdentityHashMap uses equality operator “==” for comparing keys and values

**Explain the concept of String pool.** **What is flyweight design pattern?**

The string pool is where Java caches String objects for future reuse as part of its implementation of [String interning](http://en.wikipedia.org/wiki/String_interning).  Because Strings in Java are immutable... we can save ourselves a memory allocation if we simply reuse the same String object for both *foo* and *bar*.  Indeed, Java will automatically intern all constants like this that you use in your code.  In other words:

1. String foo = "hello";
2. String bar = "hello";
3. System.**out**.println(foo == bar); // prints 'true'

Contrast this with the following:  
String foo = **new** String("hello");

String bar = **new** String("hello");

1. System.**out**.println(foo == bar); // prints 'false'

**flyweight design pattern**:- flyweight design pattern enables use sharing of objects to support large numbers of fine-grained objects efficiently. A flyweight is a shared object that can be used in multiple contexts simultaneously. The flyweight acts as an independent object in each context.

**Explain the following type of iterators**

**i)Fail-Fast:-** Iterator in java are used to iterate over the Collection objects.Fail-Fast iterators immediately throw ConcurrentModificationException if there is structural modification of the collection. Structural modification means adding, removing or updating any element from collection while a thread is iterating over that collection. Iterator on ArrayList, HashMap classes are some examples of fail-fast Iterator.

**ii)Fail-safe:-** Fail-Safe iterators don’t throw any exceptions if a collection is structurally modified while iterating over it. This is because, they operate on the clone of the collection, not on the original collection and that’s why they are called fail-safe iterators. Iterator on CopyOnWriteArrayList, ConcurrentHashMap classes are examples of fail-safe Iterator.