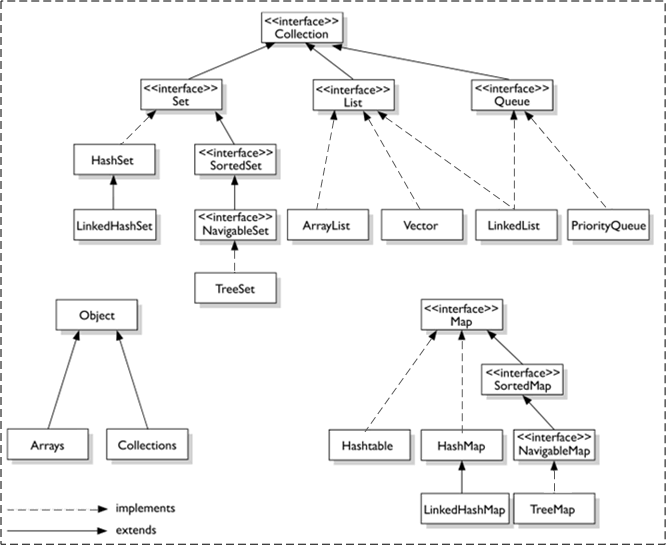
**Collection**

**Q-1: Write the collection Hierarchy and map Hierarchy**

**A: Collection** is an interface which can be used for representing a group of individual objects as single entity  and it acts as root interface of collection frame  work.

****

**Q-2: Write the any 5 legacy class.**

**A:**

* Enumeration ---Interface
* Dictonary --------Abstract class
* Hashtable -------Concrete class
* Properties -------Concrete class
* Vector ------------Concrete class
* Stack  -------------Concrete class

**Q-3: What is the diff below the statements**

**List l1 = new ArrayList();**

**ArrayList l2 = new ArrayList();**

**Q-4: What is the diff b/w ArrayList and LinkedList. Where will you use ArrayList and Where will you use LinkedList .Write any three difference. And How ArrayList work.**

**A: 1-**ArrayList is a Best choice  for retrieval purpose and worst if our frequent operation is insertion or deletion in the middle

**2-**ArryList class implements RandomAccess , Serializable , Cloneable interfaces

**3-**LinkedList is a Best choice  if frequent operation is insertion or deletion an objects in middle  but worst choice if frequent operation is retrieval.

**4-** LinkedList class implements Seriallizable and Cloneable interface but not RandomAccess interface

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1. The underlying data structure is resizable or growable array. | 1. The underlying data structure is Double Linked List. |
| 2.  This is Best choice if frequent operation is retrieval and worst choice if frequent operation is insertion or deletion in the middle. | 2.  This is Best choice  if frequent operation is insertion or deletion in the middle and worst choice if frequent operation is retrieval . |
| 3. This class implements Serializable , Cloneable and RandomAccess interfaces. | 3. This class implements Serializable , Cloneable but not  RandomAccess interface. |

**Q-5: What is difference between ArrayList and Vector?**  
 **A:**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| 1. No method is synchronized in the ArrayList class | 1. All methods in Vector are synchronized. |
| 2. ArrayList object is not thread safe. | 2.  Vector is thread safe. |
| 3. Relatively performance is high | 3. Relatively performance is low |
| 4. it is non legacy | 4. it is legacy |

**Q-6: What is the diff b/w iterator and enumeration.and which one is faster**

**A:Iterator faster than enumeration**

|  |  |
| --- | --- |
| **Enumeration** | **Iterator** |
| It is legacy interface | 1 It is non-legacy |
| 2Applicable only for legacy classes and it is not universal cursor | 2Applicable for any Collection implemented class object. |
| 3While iterating the elements we are not allowed to remove the objects just we can perform only read operation | 3While iterating we can perform removal also in addition to read operation. |
| 4By using elements() method we can get Enumeration object | 4.   By using iterator() method we can get Iterator    object |

**Q-7: Why ArrayList is faster than Vector?**

**A:**All methods present in the Vector are synchronized  and hence  any method can be executed by only one thread at a time. It slows down the execution.

But in ArrayList,  no method is synchronized and hence multiple thread are allowed execute simultaneously which speed up the execution

**Q-8: How we can get synchronized version of ArrayList?//%%%**

**A:**Collections class contains synchronizedList() method for this

                Public static List synchronizedList(List l)  
        **EX**  
                ArrayList l= new  ArrayList();  
                List l2=Collections.synchronizedList(l);

Similarly we can get synchronized versions of Set and Map objects by the following methods.

Public static List synchronizedSet(Set s)  
Public static List synchronizedMap(Map m)

**Ex:**

**import** java.util.\*;

public class Demo

{

**public** **static** **void** main(String args[])

{

// An ArrayList which is not synchronize

List<String> l1 = **new** ArrayList<String>();

l1.add("RELIANCE");

l1.add("TATA");

l1.add("TECHMAH");

l1.add("HDFC");

l1.add("ICICI");

// Synchronizing ArrayList in Java

l1 = Collections.*synchronizedList*(l1);

// While Iterating over synchronized list, you must synchronize

//on it to avoid non-deterministic behavior

**synchronized**(l1)

{

Iterator<String> i1 = l1.iterator();

**while**(i1.hasNext())

{

System.*out*.println(i1.next());

}

}

}

}

**Q-9: How to find number of occurrence of duplicate element into ArrayList.**

**A:**

import java.util.\*;

public class Lab2

{

public static void main(String[] args) {

List list = new ArrayList();

list.add("a");

list.add("b");

list.add("c");

list.add("d");

list.add("b");

list.add("c");

list.add("a");

list.add("a");

System.out.println("\nExample 1 - Count 'a' with frequency");

System.out.println("a : " + Collections.frequency(list, "a"));

System.out.println("\nExample 2 - Count all with frequency");

Set s1 = new HashSet(list);

for (String temp : s1)

{

System.out.println(temp + ": " + Collections.frequency(list, temp));

}

}

}

**Q-10: How to find the repeated element of ArrayList**

**A: import** java.util.\*;

**class** Lab2

{

**public** **static** **void** main(String[] args) {

List list = **new** LinkedList();

list.add("A");

list.add("B");

list.add("B");

list.add("A");

list.add("C");

list.add("C");

list.add("D");

System.*out*.println("My List : " + list);

Set set1 = **new** HashSet();

Set set2 = **new** HashSet();

**for** (Object s1 : list) {

**if** (!set1.add(s1)) {

set2.add(s1);

}

}

System.*out*.println(set2);

}

}

**Q-11: How to remove the repeated element from ArrayList**

**A:**

import java.util.\*;

public class RemoveDuplicate {

public static void main(String[] args) {

List l1 = new ArrayList();

l1.add("object1");

l1.add("object1");

l1.add("object2");

l1.add("object2");

l1.add("object3");

);

List newList = new ArrayList(new LinkedHashSet(sourceList));

Iterator it = newList.iterator();

while (it.hasNext()) {

System.out.println(it.next());

}

}

}

**Q-12: How to convert ArrayList to Set.**

**A:**

import java.util.\*;

public class Lab1 {

public static void main(String[] args) {

ArrayList t1=new ArrayList();

t1.add("garima");

t1.add("jyoti");

t1.add("sachi");

t1.add("garima");

t1.add("jyoti");

System.*out*.println(t1);

Set s1 = new LinkedHashSet(t1);

System.*out*.println(s1);

}

}

**Q-13: What is difference between size and capacity of a Collection Object?**

**A:**size means number  of objects present  where as capacity means no of objects it can accommodate.

**Q-14: What is the default capacity of ArrayList and what will happened beyond That object is added.then what will the capacity.**

**A :** the default capacity of ArrayList is **10**

If you add element beyond capacity ArrayList uses a same algorithm but different growing ratio: increased by ~50% of old capacity.  
    **int newCapacity = (oldCapacity \* 3)/2 + 1;**

After add 11th element capacity 16

After add 12th element capacity 25

**Q-15: How HsahSet internally work.**

**A:** **HashSet** uses HashMap internally to store it’s objects. Whenever you create a HashSet object, one **HashMap** object associated with it is also created. This HashMap object is used to store the elements you enter in the HashSet. The elements you add into HashSet are stored as **keys** of this HashMap object. The value associated with those keys will be a **constant**.

When you call to add(Object) is delegate to put(Key, Value) internally, where Key is the object you have passed and value is another object,  called PRESENT, which is a constant in java.util.HashSet  
  
**1-** Set subclasses does not allow duplicate element . (list allow duplicate element).

**2-**Set use hashCode() and equal() method will be used internally to identify the unique object in set i.e if two object have the same hash code and the equals()

method is also returning true then the object will be considered as duplicate object

**Q-16: If we are trying to insert duplicate values in Set what will happen?**

A: If we are trying to insert duplicate objects to the HashSet  , we wont get any compile time or run time errors just the add(Object o) returns false and it doesn’t add that object.

**Q-17: How to set find duplicate element.it is mandatory to write equals() along with hashCode(). If we will write only equals() then what happened.**

**Or**

**Q-18: insert the Customize class object in HashSet**

**A: import** java.util.\*;

**class** Person {

String name;

**int** age;

**public** Person(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

@Override

**public** **boolean** equals(Object o){ //how we can override equals method?

System.*out*.println("In equals");

**if** (o **instanceof** Person) {

Person pp = (Person) o;

**return** (pp.name.equals(**this**.name) && pp.age == **this**.age);

} **else** {

**return** **false**;

}

}

@Override

**public** **int** hashCode() {//how can override hashCode() method

**int** result = name.hashCode();

result = 5\*result + age; // we can give any number either no

System.*out*.println("Hash Code"+result); // need to give number

**return** result;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** String toString(){

**return** "Hashcode"+ **this**.hashCode()+"Name: "+name+" age: "+age;

}

}

**class** Demo

{

**public** **static** **void** main(String[] args) {

Set<Person> Person = **new** HashSet<Person>();

Person p1 = **new** Person("raghu", 12);

Person p2 = **new** Person("rimmu", 21);

Person p3 = **new** Person("rimmu", 21);

Person p4 = **new** Person("raghu", 12);

Person.add(p1);

Person.add(p2);

Person.add(p3);

Person.add(p4);

Iterator i1 = Person.iterator();

**while**(i1.hasNext())

{

System.*out*.println(i1.next());

}

}

}

**Ex-2:**

**import** java.util.\*;

**class** Emp

{

**private** **int** eid ;

**public** Emp( **int** eid )

{

**this**.eid = eid;

}

**public** **int** hashCode()

{

**return** eid;

}

**public** **boolean** equals( Object obj )

{

Emp emp = ( Emp )obj;

**if**( emp.eid == eid )

**return** **true**;

**else**

**return** **false**;

}

**public** String toString()

{

**return** ""+eid;

}

}

**public** **class** TestEmp

{

**public** **static** **void** main(String[] args)

{

Set s1 = **new** HashSet();

Emp emp1 = **new** Emp(23);

Emp emp2 = **new** Emp(23);

Emp emp3 = **new** Emp(23);

s1.add(emp1);

s1.add(emp2);

s1.add(emp3);

System.*out*.println(s1);

}

}

**Q-18: What will be the problem if you don't override equals() method ?**

**A:** it will allowed duplicate

**Q-19: What is the need of override equals() method with hashCode.**

**A:** First HashMap uses hashcode to find bucket for that key object, if hashcodes are same then only it checks for equals method   
1- If you are overriding equals method then you should override hashcode() also.

2- If two objects are equal then they must have same hashcode.

3- If two objects have same hashcode then they may or may not be equal

4- Always use same attributes to generate equals and hashcode as in our case we have used name.  
hash code is an unique id number allocated to an object by JVM. But actually speaking, Hash code is not an unique number for an object. If two objects are equals then these two objects should return same hash code. So we have to implement hashcode() method of a class in such way that if two objects are equals, ie compared by equal() method of that class, then those two objects must return same hash code. If you are overriding hashCode you need to override equals method also.

**Q-20: What is the diff b/w List and Set.**

**A:**

|  |  |
| --- | --- |
| * **List** | * **Set** |
| 1Insertion Order is preserved | 1Insertion Order is not preserved |
| 2Duplicate Objects are allowed | 2  Duplicate Objects are not allowed |
| 3The implemented classes are ArrayList,LinkedList , Vector and Stack classes | 3   The implemented classes are HashSet,  LinkedHashSet and Tree |

**Q-21: Differences  between HashSet and LinkedHashSet?**

|  |  |
| --- | --- |
| **HashSet** | **LinkedHashSet** |
| 1The Underlying datastructure is Hashtable | 1The underlying datastructure is combination of LinkedList and Hashtable |
| 2Insertion Order is not preserved | 2 Insertion order is preserved. |

**Q-22: What is difference between HashSet and TreeSet?**

|  |  |
| --- | --- |
| **HashSet** | **TreeSet** |
| 1The underlying data structure is Hashtable | 1The underlying data structure is balanced tree |
| 2Heterogeneous objects are allowed | 2Heterogeneous objects are not allowed  bydefalut |
| 3Insertion order is not preserved and it is based on hashcode of the objects | 3   Insertion order is not preserved and all the objects are inserted according to some sorting order. |
| 4null insertion is possible | 4   As the first element only null insertion is   possible and in all other cases we will get NullPointerException |

**Q-23: What is Fail-Fast and Fail-Safe Iterator**

**A: Fail-Fast Iterator (java.util package - HashMap, HashSet, TreeSet, etc)🡪**

Fail fast iterator while iterating through the collection , instantly throws Concurrent Modification Exception if there is structural modification  of the collection .  Structural Modification means adding, removing any element from the collection, merely updating some value in the data structure

Fail-fast iterator can throw ConcurrentModificationException in two scenarios :  
**1-Single Threaded Environment** After the creation of the iterator , structure is modified at any time by any method other than iterator's own remove method.   
   
**2-Multiple Threaded Environment**  
If one thread is modifying the structure of the collection while other thread is iterating over it .

Most collections in package java.util are fail-fast by Design. These are not meant for multi-threading/ structural modification while iteration

**Fail-Safe Iterator (java.util.concurrent - ConcurrentSkipListSet, CopyOnWriteArray List, ConcurrentMap)🡪**

Fail Safe Iterator makes copy of the internal data structure (object array) and iterates over the copied data structure.Any structural modification done to the iterator affects the copied data structure.  So , original data structure remains  structurally unchanged .Hence , no ConcurrentModificationException throws by the fail safe iterator.

**Q-24: What is the diff b/w iterator and for-each loop. And why we use iterator if**

**We have for-each loop.**

**A:** if we want to remove element while accessing the element then we will go for **iterator** and if we want to access the collection as well as array then we will go for **for-each** loop

Both is use for traversing the element.

**Q-25: What is the diff b/w Comparable and Comparator. Write the code also.**

**A:** **Comparable interface->**

(1)it is an interface comes from java.lang package

(2)Comparable interface is used to order(sort) the objects of user-defined class i.e it is used to specify that the object of the class can be compared or not.

(3)if you have the requirement to compare the object for sorting purpose or to identify the higher and lower object then java vendor has provided "Comparable interface"

(4) the folowwing absract method available in the Comparable interface

------ public int compareTo(Object obj)-----

(5)the object you want to compare must be the subtype of Comparable interface and must have to provide the implementation for the compareTo() method

Ex: import java.util.\*;

class Student implements Comparable

{

int rollno;

String name;

int age;

Student(int rollno,String name,int age){

this.rollno=rollno;

this.name=name;

this.age=age;

}

public int compareTo(Object obj) // this will sort on age basis

{

Student st = (Student)obj;

return this.age-st.age;

}

}

class Lab1{

public static void main(String args[]){

ArrayList al=new ArrayList();

al.add(new Student(101,"Vijay",23));

al.add(new Student(106,"Ajay",27));

al.add(new Student(105,"Jai",21));

System.out.println("Before sorting");

Iterator itr=al.iterator();

while(itr.hasNext()){

Student st=(Student)itr.next();

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

System.out.println("After sorting");

Collections.sort(al);

itr=al.iterator();

while(itr.hasNext()){

Student st=(Student)itr.next();

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

}

}

**Ex.2->**

import java.util.\*;

class Student implements Comparable{

int rollno;

String name;

int age;

Student(int rollno,String name,int age){

this.rollno=rollno;

this.name=name;

this.age=age;

}

public int compareTo(Object obj) // this will sort on name basis

{

Student st = (Student)obj;

return this.name.compareTo(st.name); //compareTo methods

comes from String class

}

}

class Lab1{

public static void main(String args[]){

ArrayList al=new ArrayList();

al.add(new Student(101,"Vijay",23));

al.add(new Student(106,"Ajay",27));

al.add(new Student(105,"Jai",21));

System.out.println("Before sorting");

Iterator itr=al.iterator();

while(itr.hasNext()){

Student st=(Student)itr.next();

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

System.out.println("After sorting");

Collections.sort(al);

itr=al.iterator();

while(itr.hasNext()){

Student st=(Student)itr.next();

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

}

}

**\*Comparator interface->note->**in this we create the seperate class that implements Comparator interface

(1)if you have the requirement to compare the object that is not the subclass of Comparable,java vendor has provided java.util.Comparator interface.

(2)you need to write a seperate class by implementing Comparator interface and you need to override the following method

\* public boolean equals(Object obj)

\* public int compare(Object ob1,Object ob2)

(3)within the compare() method you can typecast to the object into the corresponding type and can compare the specified object

(4)It provides multiple sorting sequence i.e. you can sort the elements based on any data member. For instance it may be on rollno, name, age or anything else.

**note->** using comparable we can define only one sorting criteria but using comperator we can define multiple sorting crieteria.

**ex.1->**

import java.util.\*;

class Student{

int rollno;

String name;

int age;

Student(int rollno,String name,int age){

this.rollno=rollno;

this.name=name;

this.age=age;

}

}

class AgeComparator implements Comparator{

public int compare(Object ob1 ,Object ob2)

{

Student s1=(Student)ob1;

Student s2=(Student)ob2;

return s1.age-s2.age;

}

}

class NameComparator implements Comparator{

public int compare(Object ob1 ,Object ob2)

{

Student s1=(Student)ob1;

Student s2=(Student)ob2;

return s1.name.compareTo(s2.name); //compareTo methods

comes from String class

}

}

class Lab1{

public static void main(String args[]){

ArrayList al=new ArrayList();

al.add(new Student(101,"Vijay",23));

al.add(new Student(106,"Ajay",27));

al.add(new Student(105,"Jai",21));

System.out.println("Sorting by Name...");

Collections.sort(al,new NameComparator());

Iterator itr=al.iterator();

while(itr.hasNext()){

Student st=(Student)itr.next();

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

System.out.println("sorting by age...");

Collections.sort(al,new AgeComparator());

Iterator itr2=al.iterator();

while(itr2.hasNext()){

Student st=(Student)itr2.next();

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

}

}

**Q-26: in this what this.age indicate and how this work.**

**public int compareTo(Object obj)**

**{**

**Student st = (Student)obj;**

**return this.age-st.age;**

**}**

**A:**

**Q-27: What are differences between Comparable and Comparator?**

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1This can be used for natural sorting order | 1-This can be used for implementing customized sorting |
| 2This interface present in java.lang package | 2-This is present in java.util package |
| 3Contains only one method:  public int compareTo(Object obj1) | 3 - It contains two methods.  public int compare(Object ,Object) public Boolean equals(Object) |

**Q-28: Explain About TreeSet.**

**A:** (1)it stores the element in sorted order.

(2)TreeSet internally implements Comparable interface.

(3)compareTo() method will be used to identify the object uniquely and

to manage the order of element in tree set.

(4)null value can not be stored in TreeSet

(5)Heterogeneous objects are not allowed violation leads to ClassCastException

(6)For an Empty TreeSet as firs element null value can be inserted but after inserting that first value if we are trying to insert any other objects then we will get NullPointerException

\*note-> (7)TreeSet allows to store elements of similar type.

**Q-29: if we are storing element in TreeSet by default it shows in assecending**

**order.if we want in descending order then what have to do.write the code**

**A:**

import java.util.\*;

public class Demo1 {

public static void main(String[] args) {

System.out.println("\*\*\*\*\*student info\*\*\*\*\*");

TreeSet t1=new TreeSet(new StringCompare());

t1.add("piya");

t1.add("nishi");

t1.add("radha");

t1.add("sachi");

t1.add("garima");

t1.add("jyoti");

Iterator it=t1.iterator();

while(it.hasNext())

{

System.out.println(it.next());

}

}

}

class StringCompare implements Comparator

{

public int compare(Object ob1 ,Object ob2)

{

String s1=(String)ob1;

String s2=(String)ob2;

// return s1.compareTo(s2); // show ascending order

return s2.compareTo(s1); // show descending order

}

}

**ex.2->**

import java.util.\*;

public class Demo1 {

public static void main(String[] args) {

System.out.println("\*\*\*\*\*student info\*\*\*\*\*");

TreeSet t1=new TreeSet(new IntCompare());

t1.add(45);

t1.add(32);

t1.add(76);

t1.add(84);

t1.add(25);

t1.add(37);

Iterator it=t1.iterator();

while(it.hasNext())

{

System.out.println(it.next());

}

}

}

class IntCompare implements Comparator

{

public int compare(Object ob1 ,Object ob2)

{

Integer s1=(Integer)ob1;

Integer s2=(Integer)ob2;

// return s1-s2; // ascending order

return s2-s1; // descending order

}

}

**Q-30: insertion of Customize Object into TreeSet**

**A:**

**Ex-1:**

**import** java.util.\*;

**class** Student

{

**int** rollno;

String name;

**int** age;

Student(**int** rollno,String name,**int** age){

**this**.rollno=rollno;

**this**.name=name;

**this**.age=age;

}

}

**class** NameComparator **implements** Comparator{

**public** **int** compare(Object ob1 ,Object ob2)

{

Student s1=(Student)ob1;

Student s2=(Student)ob2;

**return** s1.name.compareTo(s2.name);

}

}

**class** AgeComparator **implements** Comparator{

**public** **int** compare(Object ob1 ,Object ob2)

{

Student s1=(Student)ob1;

Student s2=(Student)ob2;

**return** s1.age-s2.age;

}

}

**class** Demo{

**public** **static** **void** main(String args[]){

TreeSet al1=**new** TreeSet(**new** NameComparator());

al1.add(**new** Student(101,"Vijay",23));

al1.add(**new** Student(106,"Ajay",27));

al1.add(**new** Student(105,"Jai",21));

System.*out*.println("Sorting by Name...");

Iterator itr=al1.iterator();

**while**(itr.hasNext()){

Student st1=(Student)itr.next();

System.*out*.println(st1.rollno+" "+st1.name+" "+st1.age);

}

TreeSet al2=**new** TreeSet(**new** AgeComparator());

al2.add(**new** Student(101,"Vijay",23));

al2.add(**new** Student(106,"Ajay",27));

al2.add(**new** Student(105,"Jai",21));

System.*out*.println("Sorting by Age...");

itr=al2.iterator();

**while**(itr.hasNext()){

Student st2=(Student)itr.next();

System.*out*.println(st2.rollno+" "+st2.name+" "+st2.age);

}

}

}

**Q-31: How HashMap internally work.**

**A:** (1) HashMap works on principle of hashing

.(2) It is a data structure which allows us to store object and retrieve according to the "Key-Value" format as a pair.

(3) Objects are stored by calling put(key, value) method of HashMap and retrieved by calling get(key).

(4) When we pass an both key and value to put() method to store on HashMap , it uses key object **hashcode() method** to calculate hashcode and they by applying hashing on that hashcode it identifies **bucket** location for storing value object.

(5) While retrieving it uses key object **equals() method** to find out correct key value pair and return value object associated with that key

(6)the key will be always unique in the map.

(7)the value can be duplicate.

(8)Each key can have only one value at one time.

(9)one "key-value" is known as entry.

(10)internally the data will be stored as Entry object.

(11)if you are using the entrySet() to access the data then the object accessed from set need to be typecasted into Map.Entry type.

(12) if you want to take only key value then you can use keySet()

**Q-32: What will happen in the following code.**

**HashMap h1 = ne9w HashMap();**

**h1.put(1, "a");**

**h1.put(1, "b");**

**System.*out*.println(h1);**

**A:** {1=b}

**Q-33: What is concurrent HashMap**

**A:1-** ConcurrentHashMap is the combination of HashMap and HashTable

2-ConcurrentHashMap is thread-safe that is the code can be accessed by single thread at a time .

**3-** ConcurrentHashMap only locks a portion of the collection on update.

4-ConcurrentHashMap is better than Hashtable and synchronized Map.

5-null is not allowed as a key or value in ConcurrentHashMap.

**Q-34: What is the diff b/w concurrent HashMap and HashMap.**

**A:** These are some diff b/w concurrent HashMap and HashMap

**1.  Thread -Safe :**  
ConcurrentHashMap is thread-safe that is the code can be accessed by single thread at a time .  while HashMap is not thread-safe .  
  
**2.  Synchronization Method :**  
HashMap can be synchronized by using   synchronizedMap(HashMap)  method .

By using this  method we get a HashMap object which is equivalent to the HashTable object . So every modification  is performed     on  Map is locked on Map object.

ConcurrentHashMap synchronizes or locks on the certain portion of the Map . To optimize the performance of ConcurrentHashMap , Map is divided into different partitions depending upon the Concurrency level . So that we do not need to synchronize the whole Map Object.  
  
**3.  Null Key**  
ConcurrentHashMap does not allow NULL values . So the key can not be null in ConcurrentHashMap .While In HashMap there can only be one null key .  
  
 **4.  Performance**   
In multiple threaded environment HashMap is usually faster than ConcurrentHashMap . As   only single thread can access the certain portion of the Map and thus reducing the performance .   
While in HashMap any number of threads can access the code at the same time

**Q-35: What is the diff b/w concurrent HashMap and HashTable**

**A:** Hashtable uses single lock for whole data.

ConcurrentHashMap uses a database shards(Segment<K, V>[] segments) logic, i.e. divide the data into shards(segments) than puts locks on each shard(segment) instead of putting single lock for whole data(Map).

ConcurrentHashMap and Hashtable are both thread safe. But the mechanism for thread safe is different between them. Hashtable is synchronized, it utilizes the synchronization mechanism; while ConcurrentHashMap uses segmentation to lock the data, it uses concurrent locks operations for thread safety instead of synchronized.

Since ConcurrentHashMap introduced concept of segmentation , how large it becomes only certain part of it get locked to provide thread safety so many other readers can still access map without waiting for iteration to complete.

So if you want to store large amount of data in a multithreaded program, you should consider to choose ConcurrentHashMap.

**Q-36: How to access HashMap using iterator.write the code also .**

**A:**

import java.util.\*;

public class Demo1

{

public static void main(String[] args)

{

HashMap hm=new HashMap();

hm.put("sid","kc-102");

hm.put("name","subham");

hm.put("email","subham@gmail.com");

hm.put("phone", "9616754075");

hm.put("add", "btm");

System.out.println("hash map is "+hm);

Set set = hm.entrySet();

Iterator it = set.iterator();

while(it.hasNext())

{

System.out.println(it.next());

}

}

}

**Ex: I want to access key and value individually.**

import java.util.\*;

public class Demo1

{

public static void main(String[] args)

{

HashMap hm=new HashMap();

hm.put("sid","kc-102");

hm.put("name","subham");

hm.put("email","subham@gmail.com");

hm.put("phone", "9616754075");

hm.put("add", "btm");

Set set = hm.entrySet();

Iterator it = set.iterator();

while(it.hasNext()){

Map.Entry m=(Map.Entry)it.next();

System.out.println(m.getKey()+" "+m.getValue());

}

}

}

**Q-37: How TreeSet and TreeMap perform sorting .**

**A:** internally both implement comparable interface and use the method

compareTo().

**\*TreeMap->**(1)it uses a tree data structure to decide where to store or search for keys and the position is decided by the sorting order

(2)stores the data sorted order of key.

(3)for comparing the object of sorting purpose compareTo() with the object will be used.

(4)in TreeMap the elements as key should be of similar type.

(5)Null Key can not be added in the TreeMap as Key. If you will add any null key then NullPointerException will come.

(6) in this o/p will be sorted order accoring to the key.

**Ex:**

import java.util.\*;

public class Demo1

{

public static void main(String[] args)

{

TreeMap hm=new TreeMap();

hm.put("sid","kc-102");

hm.put("name","subham");

hm.put("email","subham@gmail.com");

hm.put("phone", "9616754075");

hm.put("add", "btm");

hm.put("weight","76");

hm.put("height", "5.6");

System.out.println(hm);

Set set = hm.entrySet();

Iterator it = set.iterator();

while(it.hasNext())

{

System.out.println(it.next());

}

}

}

O/P->{add=btm, email=subham@gmail.com, height=5.6, name=subham, phone=9616754075, sid=kc-102, weight=76}

**Q-38: What is Weak reference in java.**

**A:** When there are one or more reference to an object it will not be garbage collected in Java. But this rule depends on what type of reference it is. If an object has only weak reference associated with other objects, then it is a valid candidate for garbage collection.

**Q-39: What is Weak Hash Map.**

**A:**  It is exactly same as HashMap except the following difference.

In case of HashMap an Object is not eligible for garbage collection if it is associated with HashMap even though it dosent have any external references.

But in case of WeakHashMap , if an Object is not having any external references then it is always eligible for garabage collectoion even though it is associated with weakHashMap.

**Q-40: What is identity Hash Map.**

**A:** It is exactly same as HashMap except the following difference.

In the HashMap JVM uses equals() method to identify duplicate keys  but in the  case of IdentityHashMap JVM uses == operator for this.

**Q-41: What does synchronized means in Hashtable context?**

**A:** Synchronized means only one thread can update HashTable at one point of time, any thread before any update on HashTable object will have to acquire a lock.

**Q-42: What is HashTable**

**A:** **Hashtable->**(1)A Hashtable is an array of list.Each list is known as a bucket.The position of bucket is identified by calling the hashcode() method.

(2)A Hashtable contains values based on the key. It implements the Map interface

(3)It contains only unique elements.

(4)It do notnot have any null key or value.

(5)It is synchronized.

**Q-43: What is the diff b/w HashMap and HashTable**

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| 2.No method is synchronized and hence HashMap object is not thread safe | 2 .All methods are synchronized and hence it is   thread safe |
| 3.Performance is high | 3.   Performance is low |
| 4. HashMap can contain one null key and multiple null valueslues | 4.   null insertion is not possible for both key and value violation leads to NullPointerException |

**Q-44: What is the difference between Sorting performance of Arrays.sort() vs Collections.sort() ? Which one is faster? Which one to use and when?**

**A:** Collections.sort() has a input as List so it does a translation of List to array and vice versa which is an additional step while sorting.So this should be used when you are trying to sort a list.  
Arrays.sort is for arrays so the sorting is done directly on the array.   
So clearly it should be used when you have a array available with you and you want to sort it.

Arrays.sort() is faster then Collections.sort(),

**Q-45: insert the customize object into HashMap**

**A: import** java.util.\*;

**class** Person {

String name;

**int** age;

**public** Person(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

@Override

**public** **boolean** equals(Object o){

**if** (o **instanceof** Person) {

Person pp = (Person) o;

**return** (pp.name.equals(**this**.name) && pp.age == **this**.age);

} **else** {

**return** **false**;

}

}

@Override

**public** **int** hashCode() {

**int** result = name.hashCode();

result = 5\*result + age; // we can give any number either no

**return** result; // need to give number

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** String toString(){

**return** "Name: "+name+" price: "+age;

}

}

**class** Demo

{

**public** **static** **void** main(String[] args) {

HashMap<Person, Integer> hashMapCustom = **new** HashMap<Person, Integer>();

Person p1 = **new** Person("raghu", 12);

Person p2 = **new** Person("rimmu", 21);

Person p3 = **new** Person("rimmu", 21);

Person p4 = **new** Person("raghu", 12);

hashMapCustom.put(p1, 12); // we can give any number it can be same or

hashMapCustom.put(p2, 345); // different because it is treated as constant

hashMapCustom.put(p3, 151);

hashMapCustom.put(p4, 15);

hashMapCustom.put(p4, 89);

Set<Person> keys = hashMapCustom.keySet();

**for**(Person p:keys){

System.*out*.println(p+"==>"+hashMapCustom.get(p));

}

}

}

**Q-46: Set & List interface extend Collection, so Why doesn't Map interface extend Collection?**

**Q-47:Can we use any custom object as key in HashMap ?**

**A:** yes

**Q-48:What will happen if two different HashMap key objects have same hashcode?**

**A:** They will be stored in same bucket but no next node of linked list. And keys equals () method will be used to identify correct key value pair in HashMap .

**Q-49:How null key is handled in HashMap? Since equals() and hashCode() are used to store and retrieve values, how does it work in case of null key?**

**A:** Null key is handled specially in HashMap, there are two separate method for that putForNullKey(V value) and getForNullKey(). Later is offloaded version of get() to look up null keys. Null keys always map to index 0. This null case is split out into separate methods for the sake of performance in the two most commonly used operations (get and put), but incorporated with conditionals in others. In short, equals() and hashcode() method are not used in case of null keys in HashMap.

**Q-50:Why String is popular HashMap key in Java?**

**A:**Since String is immutable, its hashcode is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for key in a Map and it’s processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.