**Consolidated questions**

**Common question**

**1. Self intro**

**2. Discussion on the previous projects and working details**

**Theory questions**

**Oops:**

1. **What is abstraction**
2. **Difference between overriding and overloading, Which one among this is related to Polymorphism**

**Core java:**

1. **Immutable class**

* **once we create the object the content will not change.**
* **If we are trying to the change the content with those change new object will be created.**
* **If no change in the content, existing object will be reused.**
* **Eg: string, wrapper classes**

**Create own immutable class**

1. **Declare the class as final.**
2. **Make all its fields final and private.**
3. **For all mutable fields, make sure that the class creates a copy and only returns the copy to the calling code.**
4. **Create parameterized constructor to initialize the final variable.**
5. **Do not provide any setter methods.**

**Program : wm\_practice - >arrays -> ImmutableClass.java**

1. **Exceptions hierarchy and what are all the exceptions you know**

* **An exception is an unwanted event that interrupt the normal flow of the program**
* **When an exception occurs, program gets abnormally terminated and system generated error messages throws By handling this exception we can provide meaningful error message which user can understand.**

**Exception Heirarchy :**

****

**3.**

**4.Shallow copy vs deep copy**

**Ref: https://www.javatpoint.com/shallow-copy-vs-deep-copy-in-java**

**Shallow copy:**

* **In shallow copy only reference is copied**
* **When we do a copy of some entity to create two or more than two entities such that changes in one entity are reflected in the other entities as well, then we can say we have done a shallow copy. In shallow copy, new memory allocation never happens for the other entities, and the only reference is copied to the other entities. The following example demonstrates the same.**

**Deep Copy:**

* **In deep copy new object is created and changes applied in original obj will not affect copied object**
* **When we do a copy of some entity to create two or more than two entities such that changes in one entity are not reflected in the other entities, then we can say we have done a deep copy. In the deep copy, a new memory allocation happens for the other entities, and reference is not copied to the other entities. Each entity has its own independent reference. The following example demonstrates the same.**

|  |  |
| --- | --- |
| **Shallow Copy** | **Deep Copy** |
| **It is fast as no new memory is allocated.** | **It is slow as new memory is allocated.** |
| **Changes in one entity is reflected in other entity.** | **Changes in one entity are not reflected in changes in another identity.** |
| **The default version of the clone() method supports shallow copy.** | **In order to make the clone() method support the deep copy, one has to override the clone() method.** |
| **A shallow copy is less expensive.** | **Deep copy is highly expensive.** |
| **Cloned object and the original object are not disjoint.** | **Cloned object and the original object are disjoint.** |

**5. how to clone on object:**

* **The object cloning is a way to create exact copy of an object.**
* **The clone() method of Object class is used to clone an object.**
* **The java.lang.Cloneable interface must be implemented by the class whose object clone we want to create.**
* **If we don't implement Cloneable interface, clone() method generates CloneNotSupportedException.**
* **Default implementation of clone() method in Java provides "*shallow copy"* of the object because it creates a copy of Object by creating a new instance and then copying content by assignment, which means if your class contains a mutable field, then both original object and clone will refer to the same internal object.**
* **This can be dangerous because any change made on that mutable field will reflect in both the original and copy object. In order to avoid this, override the clone() method to provide the**[**deep copy of an object**](http://javarevisited.blogspot.sg/2014/03/how-to-clone-collection-in-java-deep-copy-vs-shallow.html)**.**

**6. Static and Singleton design pattern**

**Ref:** [**http://net-informations.com/faq/netfaq/singlestatic.htm**](http://net-informations.com/faq/netfaq/singlestatic.htm)

**https://www.javatpoint.com/why-we-use-static-class-in-java**

* **First difference singleton is Design pattern sataic is key word,**
* **Singleton Objects stored on heap while static class stored in stack.**
* **Singleton Objects can clone but not with static class.**

**7. different ways to create object in java**

**Ref:** [**https://www.geeksforgeeks.org/different-ways-create-objects-java/**](https://www.geeksforgeeks.org/different-ways-create-objects-java/)

1. **Using new keyword – A obj = new A();**
2. **Using new instance –**

**Class cls = Class.forName("GFG");**

**// Creating object of main class**

**// using instance method**

**GFG obj = (GFG)cls.newInstance();**

**3.Using clone() method -             GFG obj2 = (GFG)obj.clone();**

1. **Using deserialization –**

**Whenever we serialize and then deserialize an object, JVM creates a separate object. In deserialization, JVM doesn’t use any constructor to create the object. To deserialize an object we need to implement the Serializable interface in the class.**

1. **Using newInstance() method of Constructor class**

**8.Transient key word**

* **the transient keyword is used during serialization. Fields that are marked as transient can not be part of the serialization and deserialization.**
* **We don't want to save the value of any variable then we use transient keyword with that variable.**
* **The transient keyword is used to exclude variable during serialization . it is initialized with default values**
* **Can not used with static final variable.**

**9.What is Reflection?**

**Ref:** [**https://www.geeksforgeeks.org/reflection-in-java/**](https://www.geeksforgeeks.org/reflection-in-java/)

* **Reflection is used to examine or modify the behavior of methods, class, interfaces at runtime.**
* **The required classes for reflection are provided under java.lang.reflect package**

**10.Solid Principles:**

**Ref:** [**https://www.javatpoint.com/solid-principles-java**](https://www.javatpoint.com/solid-principles-java)

**10. this keyword**

* The this keyword **refers to the current object in a method or constructor**.
* The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

**11.Garbage Collector & its type**

**Ref:** [**https://www.geeksforgeeks.org/garbage-collection-java/**](https://www.geeksforgeeks.org/garbage-collection-java/)

**12. What is multi-threading? ways to create thread? Which one to prefer? Detailed ques. on this.**

**Multi threading:**

* **The process of executing multiple threads simultaneously is known as multithreading.**
* **All threads of a process share the common memory.**
* **It is save cpu time**
* **So, threads are light-weight processes within a process**

**Ways to create:**

**1. By extending thread class**

**Class hi extends Thread{}**

**Hi h = new Hi();**

**h.start();**

**2. By implementing Runnable Interface**

**Class hi implements Runnable{}**

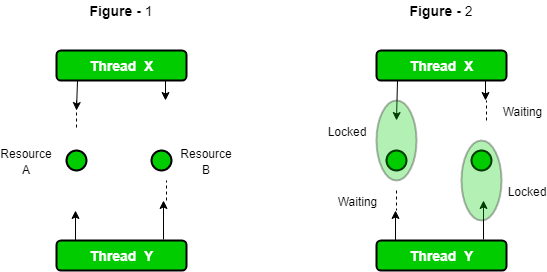
**Runnable h = new Hi();**

**Thread t1 = new Thread(h);**

**T1.start();**

**13. what is deadlock?**

**Deadlock describes a condition in which two or more threads are blocked (hung) forever because they are waiting for each other.**



**Collection / java 8**

1. **Why do we need collections framework?**

* **collection represents group of individual objects as a single entity**
* **It is used to perform various data manipulation operations like storing data, searching, sorting, insertion, deletion, and updating of data on the group of elements**
* **It is used to overcome the limitation of array (fixed size, homogeneous can’t added, built in function not available )**

1. **Difference between Array and Arraylist**

| **Base** | **Array** | **Array List** |
| --- | --- | --- |
| **Dimensionality** | **It can be single-dimensional or multidimensional** | **It can only be single-dimensional** |
|  |  |  |
| **Length** | **length keyword can give the total size of the array.** | **size() method is used to compute the size of ArrayList.** |
| **Size** | **It is static and of fixed length** | **It is dynamic and can be increased or decreased in size when required.** |
| **Speed** | **It is faster as above we see it is fixed size** | **It is relatively slower because of its dynamic nature** |
| **Primitive Datatype Storage** | **Primitive data types can be stored directly unlikely objects** | **Primitive data types are not directly added unlikely arrays, they are added indirectly with help of autoboxing and unboxing** |
|  |  |  |
| **Adding Elements** | **Assignment operator only serves the purpose** | **Here a special method is used known as add() method** |

1. **Difference between ArrayList and LinkedList**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| **1) ArrayList internally uses a dynamic array to store the elements.** | **LinkedList internally uses a doubly linked list to store the elements.** |
| **2) Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory.** | **Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.** |
| **3) An ArrayList class can act as a list only because it implements List only.** | **LinkedList class can act as a list and queue both because it implements List and Deque interfaces.** |
| **4) ArrayList is better for storing and accessing data.** | **LinkedList its better for manipulating data.** |
| **5) The memory location for the elements of an ArrayList is contiguous.** | **The location for the elements of a linked list is not contagious.** |
| **6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList.** | **There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized.** |
| **7) To be precise, an ArrayList is a resizable array.** | **LinkedList implements the doubly linked list of the list interface.** |

1. **Difference between Hashmap vs Concurrent Hashmap**

|  |  |  |
| --- | --- | --- |
| **Parameters** | **HashMap** | **ConcurrentHashMap** |
| **Synchronization** | **Non-synchronized** | **synchronized** |
| **Thread-safety** | **Not thread-safe** | **Thread-safe** |
| **Iterator** | **It is fail-fast and throws an exception during iteration** | **It is fail-safe and performs iteration by multiple threads** |
| **Null Values** | **It allows for storing null keys and values.** | **It does not allow to store null key/values.** |
| **Performance** | **faster** | **Slower than Hashmap** |

1. **What is List**

* **it is child interface of collection interface**
* **if we want to represent a group of individual objects as a single entity where duplicates are allowed and insertion order preserved then we should go for List.**
* **we can differentiate duplicated by using index**
* **we can preserve insertion order by using index**
* **implemented class : Array list , linked list, vector -> stack**

1. **Set Vs List**

|  |  |  |
| --- | --- | --- |
| **S.No** | **List** | **Set** |
| **1.** | **The list implementation allows us to add the same or duplicate elements.** | **The set implementation doesn't allow us to add the same or duplicate elements.** |
| **2.** | **The insertion order is maintained by the List.** | **It doesn't maintain the insertion order of elements.** |
| **3.** | **List allows us to add any number of null values.** | **Set allows us to add at least one null value in it.** |
| **4.** | **The List implementation classes are LinkedList and ArrayList.,Vector,stack** | **The Set implementation classes are TreeSet, HashSet and LinkedHashSet.** |
| **5.** | **We can get the element of a specified index from the list using the get() method.** | **We cannot find the element from the Set based on the index because it doesn't provide any get method.** |
| **6.** | **It is used when we want to frequently access the elements by using the index.** | **It is used when we want to design a collection of distinct elements.** |
| **7.** | **The method of List interface listIterator() is used to iterate the List elements.** | **The iterator is used when we need to iterate the Set elements.** |

1. **What is hashmap**

* **it is implements Map interface**
* **underlying Data structure for HashMap is Hashtable**
* **insertion order not preserved and order is based on hash code of keys**
* **duplicate keys are not allowed but duplicate values are allowed**
* **heterogeneous allowed in both key and values**
* **only one null key accepted , multiple null values accepted(any number of times. If we add duplicate keys the values are override)**
* **Implements serializable, Clonable but not Random Access Interface**
* **hash map is best choice if our frequent operation is search.**

**Constructor:**

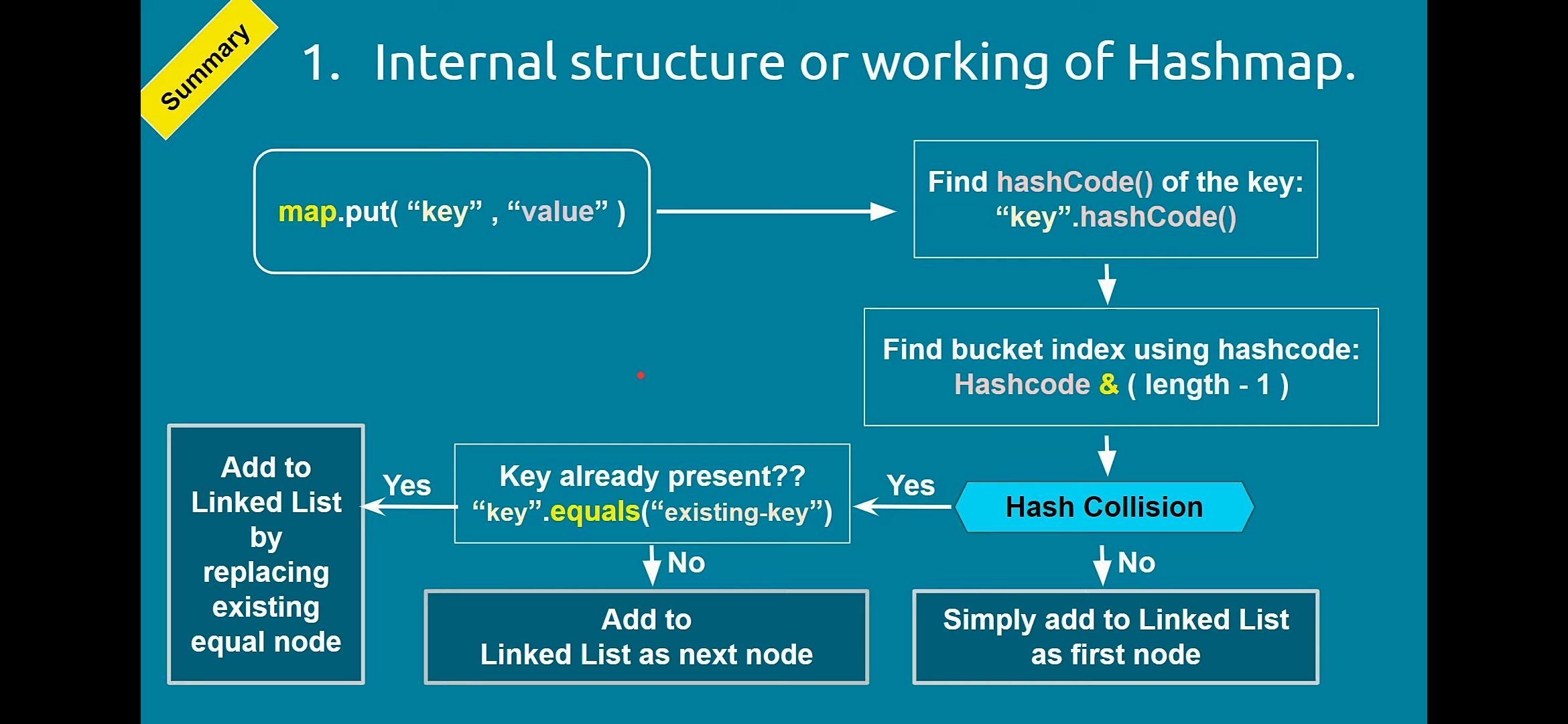
**1. HashMap m = new HashMap(); //creates an empty hash map object with default initial capacity 16, default fill ratio 0.75**

**2. HashMap m = new HashMap(int capacity); //creates an empty hash map object with given initial capacity , default fill ratio 0.75**

**3. HashMap m = new HashMap(int capacity, float ratio); //creates an empty hash map object with given initial capacity, given fill ratio**

**4. HashMap m = new HashMap(Map m);// convert other map object into HashMap**

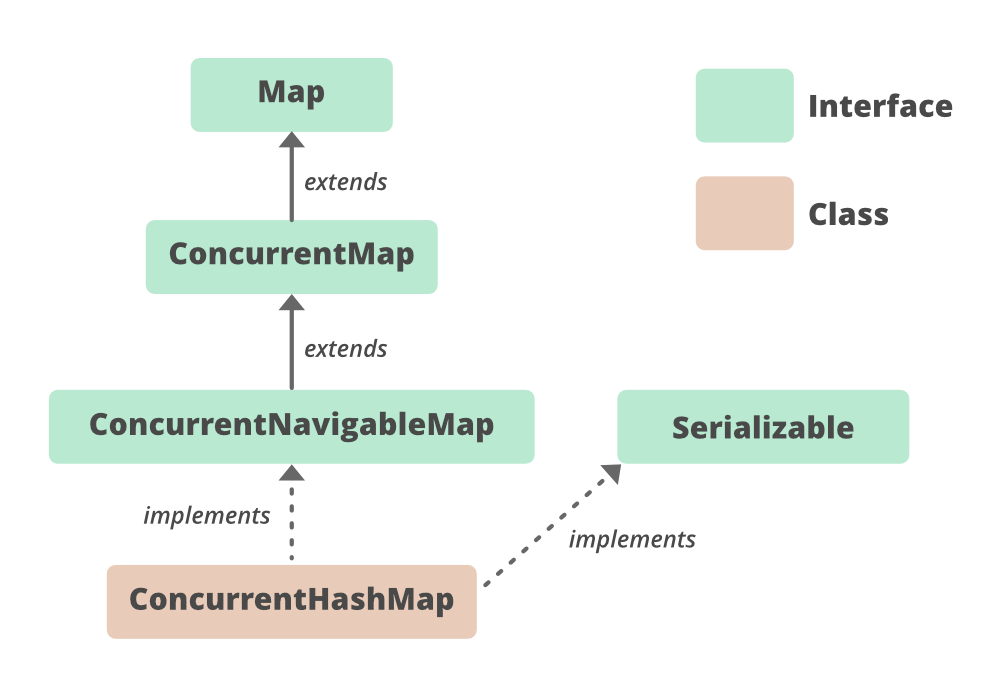
1. **hashmap internall working**

****

* **Internal data structure for hashmap is hashtable**
* **map.put(key,value) method is used to place the key-value pair entry in hashmap**
* **Here hashtable contains the list of buckets, initial bucket size is 16**
* **In each bucket we have a linked list to store more than 1 node**
* **Each node contain, key, value, and address of next node.**
* **When we call the put method, JVM will find the hashcode (integer form of key address/reference) of key**
* **Using hash technique jvm will find the bucket index to store the map entry.**
* **If bucket is empty, hash collision will not occur or else hash collision will occur**
* **If bucket is empty entry will be added to Linked List as first node.**
* **If bucket is not empty, using equals method new key will be compared against existing key value to check the key already exist or not.**
* **This step will be repeated until the linked list next node is null**
* **If key is not exist and linked list next node address is null new key value pair node will be added to linked list**
* **If key is exist, the value will be replaced with new value and returns old value.**
* **It won’t throw any error.**
* **Since it needs to search key in linked list the performance is low for search operation**
* **Hence hashmap performance getting slow for long linked list**
* **To overcome this in JAVA 8 hashmap enhanced to improve performance.**
* **The idea is, linked list will be replaced by binary search tree when it is reached the certain threshold. the threshold is known as “ Treeify threshold “, And the limit is 8.**
* **In binary search tree compareTo() is used to compare the key to check whether the given key is less or greater then existing key.**
* **If we try to add null in hashmap , since hashcode of null is 0, so null key stored in index 0,**
* **If we add again one more null key again it will be added to same index 0 that time the value will be replaced duplication null key not added**

1. **What is concurrent hashmap / internal working?**

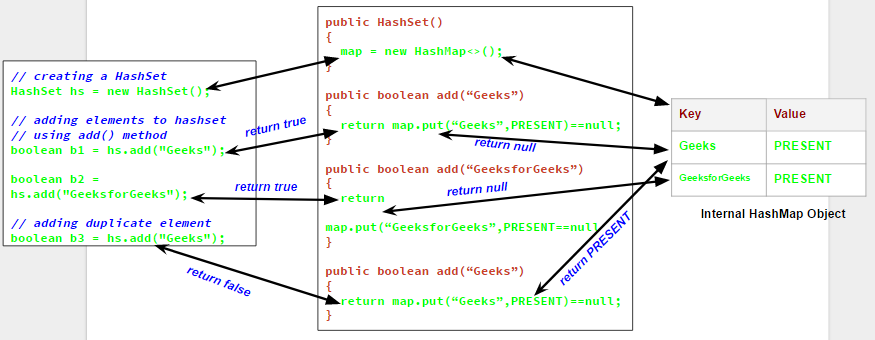
* **ConcurrentHashMap is an enhancement of HashMap which is used in multhreading environment**
* **it is the implementation class of ConcurrentMap**
* **underlying architecture is hashtable**
* **in hashtable, at a time only one thread allowed to operate on collection object. even read operation. hence it is thread safe.**
* **in concurrent hash map, if it is read operation multiple threads allowed to read. no restriction and no waiting time. Hence it is allowed concurrent read operations**
* **if it is update/write operation instead of locking total collection object(in hash table) we are locking bucket level. thread safe write operation allowed**
* **entire hashmap divided into 16 by default which is called as concurrency level. for each segment (part) separate lock will be maintained if thread want to update in one part, that segment only locked,**
* **if capacity of concurrent hash map is 16 then at a time 16 thread can perform update operation. hence performance is increased.**
* **hence for update/write thread locking only the particular part of the map which is bucket level or segment level locking.**
* **concurrent hash map divided into smaller parts which is defined by concurrency level. default concurrency level is 16.**
* **CHM allowed any read operation and 16 update operation allowed**
* **null is not allowed both key/value operation**
* **hence while one thread reading object if other thread modified the collection it won't throw concurrent modification exception**
* **So any write operation(remove/put/clear etc) will work in 3 steps:  
  1. Wait until it gets the lock on that Segment.  
  2. Do the operation.  
  3. Release the lock after it.**
* **Inserting null objects is not possible in ConcurrentHashMap as a key or value.**

****

1. **HashSet Vs HashMap**

|  |  |  |
| --- | --- | --- |
| **Basis** | **HashMap** | **HashSet** |
| **Definition** | **Java HashMap is a hash table based implementation of Map interface.** | **HashSet is a Set. It creates a collection that uses a hash table for storage.** |
| **Implementation** | **HashMap implements Map, Cloneable, and Serializable interface es.** | **HashSet implements Set, Cloneable, Serializable, Iterable and Collection interfaces.** |
| **Stores** | **In HashMap we store a key-value pair. It maintains the mapping of key and value.** | **In HashSet, we store objects.** |
| **Duplicate values** | **It does not allow duplicate keys, but duplicate values are allowed.** | **It does not allow duplicate values.** |
| **Null values** | **It can contain a single null key and multiple null values.** | **It can contain a single null value.** |
| **Method of insertion** | **HashMap uses the put() method to add the elements in the HashMap.** | **HashSet uses the add() method to add elements in the HashSet.** |
| **Performance** | **HashMap is faster/ than HashSet because values are associated with a unique key.** | **HashSet is slower than HashMap because the member object is used for calculating hashcode value, which can be same for two objects.** |
| **The Number of objects** | **There are two objects created during put operation, one for key and one for value.** | **Only one object is created during the add operation.** |
| **Storing Mechanism** | **HashMap internally uses hashing to store objects.** | **HashSet internally uses a HashMap object to store objects.** |
| **Uses** | **Always prefer when we do not maintain the uniqueness.** | **It is used when we need to maintain the uniqueness of data.** |
| **Iterator** | **we cannot iterate a Map directly using**[**iterators**](https://www.geeksforgeeks.org/iterators-in-java/)**, because Map are not**[**Collection.**](https://www.geeksforgeeks.org/collections-in-java-2/) | **We can iterate Set using set.iterator()** |

1. **How hash set working internally**

****

* **Hash set is internally using hashmap to store elements**
* **When we add the value to the set, internally the value will be saved as key in hash map object and value for the corresponding key is “PRESENTT”**
* **It will return null if the key is new**
* **It will return PRESENT/old value if the key already present**
* **While adding the element to the set , set internally check whether the put method returns true or false**
* **when we try to add a duplicate element to a set using add() method, it returns false, and element is not added to hashset, as it is already present**
* **If we try to add null in hashset , since hashcode of null is 0, so null key stored in index 0,**
* **If we add again one more null key again it will not be added to set**

1. **What is hashtable**

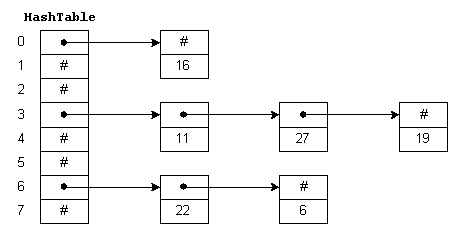
* **underlying DS for hashtable is hash table only.**
* **every method in the class is synchronized hence it is thread safe**
* **insertion order not preserved and it is based on hashcode of keys**
* **duplicate keys not allowed but values can be duplicated**
* **heterogeneous allowed for both key and values**
* **null is not applicable in both key, value**
* **implements Serializabe, Clonable but not Random Access Interface**
* **if our frequent operation is search Hash table is best choice**

**Constructors:**

**1.Hashtable h = new Hashtable(); -- creates empty hash table with default initial capacity 11 and fill ratio 0.75 ( hash set/map initial capacity is 16)**

1. **How hashtable working internally**

* **Hashtable internally contains array of buckets in which it stores the key/value pairs.**
* **The Hashtable uses the key’s**[**hashcode**](https://howtodoinjava.com/java/basics/java-hashcode-equals-methods/)**to determine to which bucket the key/value pair should map.**
* **Using hash function we can calculate the bucket index of the hash table for given key**
* **When we put objects into a hashtable, it is possible that different objects might have the same hashcode. This is called a collision.**
* **To resolve collisions, hashtable uses an linked lists. The pairs mapped to a single bucket (array index) are stored in a list and list reference is stored in array index**

****

1. **Hashmap vs HashTable ?which is faster and why ?**

| **S. No.** | **Hashmap** | **Hashtable** |
| --- | --- | --- |
| **1.** | **No method is synchronized.** | **Every method is synchronized.** |
| **2.** | **Multiple threads can operate simultaneously and hence hashmap’s object is not thread-safe.** | **At a time only one thread is allowed to operate the Hashtable’s object. Hence it is thread-safe.** |
| **3.** | **Threads are not required to wait and hence relatively performance is high.** | **It increases the waiting time of the thread and hence performance is low.** |
| **4.** | **Null is allowed for both key and value.** | **Null is not allowed for both key and value. Otherwise, we will get a null pointer exception.** |

* **Hash map is not thread safe, so multiple threads can access hash map pbject at a time, hence there is no locking and waiting time. And hash map is fast compared to hash table.**

1. **What are Java 8 Features**
2. **Lambda Expression**
3. **Functional Interfaces**
4. **Default methods in Interfaces**
5. **Static methods in Interface**
6. **Streams API**
7. **Predicate (Predefined Functional Interfaces)**
8. **Function (Predefined Functional Interfaces)**
9. **Consumer (Predefined Functional Interfaces)**
10. **Method reference & constructor reference by :: operator.**
11. **Date and Time API (introduced by joda.org, also known as Joda API)**
12. **Optional Class**
13. **JDK enhancement**
14. **What is functional interface**

* **Functional interfaces are included in Java SE 8 with Lambda expressions and Method references in order to make code more readable, clean, and straightforward.**
* **Functional interfaces are interfaces that ensure that they include precisely only one abstract method, any quantity of default and static methods.**
* **Functional interfaces are used and executed by representing the interface with an annotation called *@FunctionalInterface***
* **In Functional interfaces, there is no need to use the abstract keyword as it is optional to use the abstract keyword because, by default, the method defined inside the interface is abstract only.**

**Ref:** [**https://www.geeksforgeeks.org/functional-interfaces-java/#:~:text=A%20functional%20interface%20is%20an,any%20number%20of%20default%20methods**](https://www.geeksforgeeks.org/functional-interfaces-java/#:~:text=A%20functional%20interface%20is%20an,any%20number%20of%20default%20methods)**.**

1. **Lambda expression**

* **A lambda expression is a short block of code which takes in parameters and returns a value.**
* **It is the function without name, without return type, without modifiers**
* **We can also call Lambda expressions as the instance of functional interface.**
* **To call lambda expression we need Functional interface**

**advantage of lambda expression**

* **to enable functional programming in java**
* **To write more readable, maintainable, concise code**
* **To use API’s very easily and effectively**

**18. Which one the more efficient forEach() loop or Stream().forEach()?**

* **Collection.forEach is better & faster ,**
* **Because in stream we need to convert collection into stream which will take extra time and memory space**

**19. what is stream**

* **Stream api is used to process the collection of objects**
* **Stream provide the result without changing the original collection object**
* stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

The features of Java stream are –

* A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.
* Streams don’t change the original data structure, they only provide the result as per the pipelined methods.
* Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

**20.forEach vs stream.forEach**

| foreach() loop | stream().foreach() loop | parallelStream().foreach() loop |
| --- | --- | --- |
| Lambda operators is not used | Lambda operator is used | Lambda operator is used |
| Can be used to access arrays and collections | Can access collections only | Can access collections only |
| The return or control statements work within the loop | The return or control statements don’t work within the loop | The return or control statements don’t work within the loop |
| No multithreading thus slow data is in sequence | No multithreading thuss slow data is in sequence | It is multithreaded thus very fast and sequence is different |

|  |
| --- |
|  |
| **21.How streams working internally?**  **Ref:** **https://theboreddev.com/understanding-java-streams/#:~:text=Since%20JDK%208%2C%20a%20spliterator,whatever%20reason%20you%20need%20it.** |
|  |

**22. Streams Time and space complexity or optimization?**

**Ref:**

**https://jaxenter.com/java-performance-tutorial-how-fast-are-the-java-8-streams-118830.html**

**23. Parallel Streams: How many threads will be created and how does it distribute our collections internally, Can we define our logic for data separation**

**Ref:**

[**https://medium.com/geekculture/pitfalls-of-java-parallel-streams-731fe0c1eb5f**](https://medium.com/geekculture/pitfalls-of-java-parallel-streams-731fe0c1eb5f)

**23. Using HashMap in multiple threads ? What are the consequences.**

**1. hash map is not a thread safe. if we are using hash map in multi thread environment they may be data in consistency and errornous output**

**2. in worst case**  **if one thread tries to put data and requires Rehashing and at same time other thread tries to read data from Hashmap, It will go in infinite loop.**

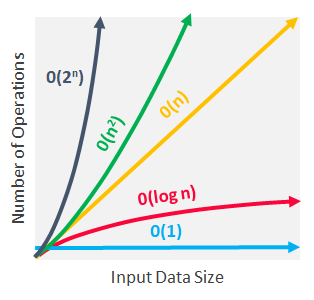
**3. to avoid the above two issue we can use synchronized hash map to provide thread safe.**

**4. In Synchronized HashMap, multiple threads can not access the map concurrently. Hence, the performance is relatively less than the Map, ConcurrentHashMap**

**Scenario:**

1. **How to create logs file for each users from existing logs file. (scenario1)**
2. **How to speed-up the DB processing like – Indexing, Query Optimizing & Stored Procedure (scenario2)**
3. **Have you done anything like profiling & optimization of application?**

**Programs**

****

**Linked List programs:**

**https://www.geeksforgeeks.org/function-to-check-if-a-singly-linked-list-is-palindrome/**

1. **Find First number with least occurrence**

**Sol:**

**Time Complexity: O(n)**

**Space Complexity: O(n)**

**LeastOccurance.java**

1. **Swap elemets in a linkedlist**

**Input 1->2->3->4->5**

**Output 1->3->2->5->4**

**Input 1->2->3->4->5->6**

**Output 2->1->4->3->6->5**

**SwapLinkedListElement.java**

**Ref:**

**Swap any 2 elements in LL**

**Ref:** [**https://www.geeksforgeeks.org/swap-nodes-in-a-linked-list-without-swapping-data/**](https://www.geeksforgeeks.org/swap-nodes-in-a-linked-list-without-swapping-data/)

**Swap adjacent elements in LL**

**Ref:** [**https://www.geeksforgeeks.org/pairwise-swap-elements-of-a-given-linked-list/#:~:text=Given%20a%20singly%20linked%20list,function%20to%20swap%20elements%20pairwise.&text=For%20example%2C%20if%20the%20linked,function%20should%20change%20it%20to**](https://www.geeksforgeeks.org/pairwise-swap-elements-of-a-given-linked-list/#:~:text=Given%20a%20singly%20linked%20list,function%20to%20swap%20elements%20pairwise.&text=For%20example%2C%20if%20the%20linked,function%20should%20change%20it%20to)**.**

**swap only links**

**https://www.geeksforgeeks.org/pairwise-swap-elements-of-a-given-linked-list-by-changing-links/**

1. **Sort an array – need to find diff solution**

**// try to find the solution without back tracking**

**So1 1:**

**SortArray.java**

1. **Find pair of elements in Array with given sum?**

**PairWithSum.java**

1. **Form a custom Linked List which store a value of integer**

**CustomLinkedList.java**

1. **Reverse the string (Efficent- Time Complexity)**

**ReverseString.java**

1. **Find the first duplicate in array**

**FirstDuplicate.java**

1. **Check palindrome with singly linked list characters.**

**consider the linked list r->a->c->e->c->a->, find whether it is palindrome or not with less space complexity and time complexity**

**Ref :** [**https://www.geeksforgeeks.org/function-to-check-if-a-singly-linked-list-is-palindrome/**](https://www.geeksforgeeks.org/function-to-check-if-a-singly-linked-list-is-palindrome/)

**Sol 1: using reverse second half**

**Sol 2 : using stack**

**PalindromeLinkedList.java**

1. **Sort the list by number of occurrence of the element**

**Sol 1: sort using the hashmap**

**sortArrayByFrequency.java**

1. **Sorting of Employee class with id, Name, Salary**

**SortEmployee.java**

1. **Remove duplicate value from an array without using internal function.**

**RemoveDuplicates**

1. **Producer and consumer problem in multi threading**

**Ref:**

[**https://www.geeksforgeeks.org/understanding-threads-on-producer-consumer-problem-java/#:~:text=Multi%2DThreading%20in%20Java%3A%20In,buffer%20used%20as%20a%20queue**](https://www.geeksforgeeks.org/understanding-threads-on-producer-consumer-problem-java/#:~:text=Multi%2DThreading%20in%20Java%3A%20In,buffer%20used%20as%20a%20queue)**.**

**ProduceConsumerProblem.java**

1. **Given an array int[] a={1,2,3,1,1,2,2,1,3,2,1} Move all the 1's to the left, without changing the order of other elements**

**AllOneToLeft.java**

1. **Given a String S="[{(})[]" , find out whether it contains valid paranthesis or not**

**checkParenthesis.java**

1. **Write a sample recursive program**

**Recursion.java**

1. **Program on LRU Cache Implementation**
2. **LRUWithHashSet.java**
3. **LRUCache.java**
4. **Find the third largest element in array.**

**ThirdLargestNumber.java**

1. **Count of strings where adjacent characters are of difference one - try to find better sol, spac comp**

**CountString.java**

1. **Sort a given string in CamelCase input: gEeksfOraEEkS output: aEefgkEkrEOsS**

**camelSort.java**

1. **Check Palindrome or not**

**CheckPalindrome.java**

1. **Find the second repeated elements**

**SecondDuplicate.java**

1. **How to find out employees whose Salary (more than some amount) via java 8 (filter,map,forEach)**

**FilterEmployee.java**

**Concurrent hashing**

**How load balancer works internally using round robin**

**Auto Scaling**

**Circuit breaker design pattern**

**Oracle performance tuning, indexing, Btree indexing what other indexes we can use**

**Saga design pattern, API Gateway,CQRS, service discovery, proxy**

**is it better to use array or any collection classes?**

**Difference between topic and queue**

**Kafka advantages over MQ/rabbit MQ**

**SOLID principles**

**how will you implement logging seperately**

**How security handle in microservices -  JWT token/SAML/OAuth/Authentication and Authorization**

**Streams - Map/Reduce**

**Functional Interfaces - function, predicate,consumer, producer**

**Lambda expression**

**method references**

**write a block of code for filter, map and forEach using Lambda Expression**

**Design patterns - Singleton, factory,builder, bridge, command, adaptor, decorator, chain of responsibilities**

**How Sync/Async calls are handled in your project**

**Doubts:**

1. **Time/space complexity for distinct()/map/filer**