**Topic Name 🡪Multithreading, Collection Frameworks**

……………………………………………….. Questions & Answers ………………………………………………………

Q1) What is Synchronised ?

* Synchronization means when multiple threads is accessing the one resource at the same time.
* The main purpose of this is we need to ensure that resource will be used by only one thread at a time.
* We can apply the synchronization on method and block only we can’t apply on it on variable and class.

Q2) Diff between Array and Vector ?

|  |  |
| --- | --- |
| **Array** | **Vector** |
| Array is not synchronized. | Vector is synchronized, its object is thread safe. |
| Array stores a fixed-size sequential collection of elements of the same type and it is index based. | Vector is a sequential container to store elements and not index based |
| Arrays have a fixed size. | Vectors have a dynamic size i.e. they can resize themselves. |
| Homogeneous objects are allowed | Heterogeneous object can allow |
| Non-legacy class | Legacy class |

Q3) Which is a default cloneable class in arraylist ?

* Arraylist.clone()

Q4) Your laptop is having only one processor still it works on multiple processes at one time, how ?

* One processor have multiple cores
* Multiple cores are responsible to create multiple threads

Q5) Explain Collection framework ?

* The Java collections framework provides a set of interfaces and classes to implement various data structures and algorithms.
* Collection Framework enables the user to perform various data manipulation operations like storing data, searching, sorting, insertion, deletion, and updating of data on the group of elements.
* It provides high-performance implementations of useful data structures and algorithms that increases the performance.

Q6) What is Garbage collection ?

* The basic principles of garbage collection are to find data objects in a program that cannot be accessed in the future, and to reclaim the resources used by those objects.
* In Java, garbage collection is the process of managing memory, automatically. It finds the unused objects (that are no longer used by the program) and delete or remove them to free up the memory.
* The garbage collector is the best example of the Daemon thread as it is always running in the background.

Q7) Collection interface ?

* The Collection interface is a member of the Java Collections Framework.
* It is a part of java. util package. It is one of the root interfaces of the Collection Hierarchy.
* The Collection interface is not directly implemented by any class
* The core collection interfaces are:

Collection : The root of the collection hierarchy

Set : A collection that cannot contain duplicate elements

List : An ordered collection (sometimes called a sequence)

Queue : A collection used to hold multiple elements prior to processing.

Map : An object that maps keys to values

Q8) Diff Arraylist vs Vector vs Linkedlist, HashSet vs TreeSet ?

|  |  |
| --- | --- |
| Arraylist | Vector |
| 1) ArrayList is **not synchronized.** | Vector is **synchronized.** |
| 2) ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity. | Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity. |
| 3)ArrayList is **not a legacy** class. It is introduced in JDK 1.2. | Vector is a **legacy** class. |
| 4)ArrayList is **fast** because it is non-synchronized. | Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object. |
| 5) ArrayList uses the **Iterator** interface to traverse the elements. | A Vector can use the **Iterator** interface or **Enumeration** interface to traverse the elements. |
|  |  |

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 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. |
| ArrayList is **better for storing and accessing**data. | LinkedList is **better for manipulating** data. |
| The memory location for the elements of an ArrayList is contiguous. | The location for the elements of a linked list is not contagious. |
| 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. |
| an ArrayList is a resizable array. | LinkedList implements the doubly linked list of the list interface. |
|  |  |

|  |  |  |
| --- | --- | --- |
| Parameters | HashSet | TreeSet |
| **Ordering or Sorting** | It does not provide a guarantee to sort the data. | It provides a guarantee to sort the data. The sorting depends on the supplied Comparator. |
| **Null Objects** | In HashSet, **only an element** can be null. | It does not allow null elements. |
| **Comparison** | It is **slower** in comparison to HashSet. | It uses **compare()** or **compareTo()** method for comparison. |
| **Performance** | It is **faster** than TreeSet | It is **slower** in comparison to HashSet. |
| **Data Structure** | HashSet is backed up by a hash table | TreeSet is backed up by a Red-black Tree. |
| **Values Stored** | It allows only heterogeneous value It allows only **heterogeneous** value  It  It allows only **heterogeneous** value  It allows only **heterogeneous** value | It allows only **homogeneous** value. |
| **Implementation** | Internally it uses **HashMap** to store its elements. | Internally it uses **TreeMap** to store its elements. |

Q9) Synchronized keyword ?

Synchronized blocks in Java are marked with the synchronized keyword. A synchronized block in Java is synchronized on some object. All synchronized blocks synchronize on the same object can only have one thread executing inside them at a time.

The synchronized keyword can only be used on method declarations and as synchronized blocks.

Q10) generics in java ?

The Java Generics allows us to create a single class, interface, and method that can be used with different types of data (objects). This helps us to reuse our code.

It is represented by <Employee>

There are mainly 3 advantages of generics. They are as follows:

1. **Type-safety:** We can hold only a single type of objects in generics. It doesn?t allow to store other objects.
2. **Type casting is not required:** There is no need to typecast the object.
3. **Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

**Q11)** Wait and sleep method difference?

|  |  |
| --- | --- |
| **Wait()** | **Sleep()** |
| The Wait() method is related to the Object class. | The Sleep () method is related to the Thread class. |
| It is not a static method. | It is a static method. |
| At the time of the Synchronization, the Wait() method releases obj. | At the time of the Synchronization, the Sleep() method doesn't release the obj, i.e., lock. |
| The Sleep() method has two overloaded methods, which are as follows:   * sleep(long milliseconds, int nanoseconds) * sleep(long milliseconds) | The Sleep() method has three overloaded methods, which are as follows:   * Wait() * wait(long timeout, int nanoseconds) * wait(long timeout) |
| The constructor of the Wait() method is defined in the following way: public final void Wait(long timeout) | The constructor of the Sleep () method in the following way: public static void Sleep (long millis) throws Interrupted\_Execption |

Q12) What is difference between Hashset and 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. |
| **The Number of objects** | Only **one** object is created during the add operation. | There are **two** objects created during put operation, one for **key** and one for **value**. |
|  |  |  |

**Q13)** How we use stream in collection ?

* The whole idea of Java streams is to enable functional-style operations on streams of elements.
* A stream is an abstraction of a non-mutable collection of functions applied in some order to the data.
* A stream is not a collection where you can store elements.
* Collections are used to store and group the data in a particular data structure like List, Set or Map.
* But, streams are used to perform complex data processing operations like filtering, matching, mapping etc on stored data such as arrays, collections or I/O resources.
* That means, collections are mainly about data and streams are mainly about operations on data.

Q14) What is thread scheduler & job of thread scheduler collections ?

* Thread scheduler in Java is the component of JVM that determines the execution order of multiple threads on a single processor (CPU).
* It decides the order in which threads should run. This process is called thread scheduling in Java.
* In Java, a thread is only chosen by a thread scheduler if it is in the runnable state.
* However, if there is more than one thread in the runnable state, it is up to the thread scheduler to pick one of the threads and ignore the other ones.

Q15) Internal working of Hashset/Set?

Each and every element in set is unique. So that there is no duplicate element in set. So In Java, if we want to add element in set then write a code like this.

Example-

**package** com.test;

**import** java.util.HashSet;

**public** **class** HashSetDemo {

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

HashSet hs= **new** HashSet();

hs.add(3);

hs.add("ram");

hs.add("Pune");

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

}

}

Output-

[3, Pune, ram]

Now, let add the duplicate element in the above code as

**package** com.test;

**import** java.util.HashSet;

**public** **class** HashSetDemo {

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

HashSet hashSet = **new** HashSet();

hashSet.add(3);

hashSet.add("ram");

hashSet.add("Pune");

hashSet.add(3);

hashSet.add("Pune");

System.***out***.println(hashSet); }

}

Output-

[3, Pune, ram]

Now what happen internally when you pass duplicate element in set then add () method of set object, it will return false and do not add to hashset as element is already present.

But main problem is arising that how it return false, here is the answer.

When you open hashset implementation of add method () in Java API’s you will find the following code-

Public class HashSet <E> extends AbstractSet<E> implements Set<E>, clonable java.o.serializable

{

Private transient Hashmap<E, Object> Map;

/\* dummy value associate with object in map \*/

Private static final object Present= new object();

Public HashSet(){

Map= new Hashmap<>();

//some code, other method in hashset

}

Public Boolean add (E e) {

return map. Put(e, PRESENT)==null;

}

We are achieving uniqueness in set, internally java through hashmap. Whenever you create the object of hashset it will create the object of hashmap as see in above.

As we know, in hashmap each key is unique. We do in set is that we pass argument in add(Element E) that is E as key in hashmap, now we need to associate some value to key, so what java developer did to pass dummy value that is(new Object()); which is referred by object reference PRESENT.

So actually when you are adding line in hashset like hashset.add(3) what java internally is that it will put that element as E here as 3 key in hashmap and some dummy value that object is passed as value to key.

If you see code of hashmap put(K k, value v) method, you will find something like this,

Public v put (K Key, V value){

//some code

}

The main point is that .put(key,value) will return

1. Null, if key is unique and added to map.
2. Old value of key, if key is duplicate.

So in Hashset add() method, we check return value of map.put(key,value) method will null value i.e.

Public Boolean add(E e){

// code here

}

So If Map.put(key,value) return null, then map.put(e,PRESENT)==null, then map.put(e,PRESENT)==null will return true & element added to hashset.

So If Map.put(key,value) return old value of key, then map.put(e,PRESENT)==null, then map.put(e,PRESENT)==null will return false & element is not added to hashset.

Q16) Why map does not belong to collection interface..?

* Map is not a part of Collection .
* Each Collection stores a single value where as a Map stores key-value pair.
* So methods in Collection interface are incompatible for Map interface.
* its characteristics and behaviors are different than the other collections like List or Set.
* A Map cannot contain duplicate keys and each key can map to at most one value.

Q17) What is array list?

* An ArrayList class is a resizable array, which is present in the java. util package.
* While built-in arrays have a fixed size, ArrayLists can change their size dynamically.
* Elements can be added and removed from an ArrayList
* Below are some useful methods in the ArrayList class:

**Add an item:** The add() method is used to add an item at the start of an ArrayList. The index of this item is 0 and all other indexes are increased by 1.

**Access an item:** The get() method, taking an index as input, is used to access an element in the ArrayList.

**Set an item:** The set() method, taking an index as input, is used to set an element in the ArrayList at the specified index.

**Remove an item:** The remove() method, taking an index as input, is used to remove an element in an ArrayList. Indexes of all the elements in front of the removed element are reduced by 1.

**Remove all items:** The clear() method is used to remove all elements in an ArrayList.

**Size of ArrayList:** The size() method is used to find the number of elements in an ArrayList.

Q18) When you use Array-list and linked list ?

* Array-List provides constant time for search operation, so it is better to use ArrayList if searching is more frequent operation than add and remove operation.
* The LinkedList provides constant time for add and remove operations. So it is better to use LinkedList for manipulation.
* In sort, ArrayList is better to access data wherease LinkedList is better to manipulate data.

Q19) Difference between wait and join, sleep method ?

|  |  |  |
| --- | --- | --- |
| **Key** | **wait()** | **join()** |
| Declaration | wait() method is defined in Object class and hence the wait() method is declared in java.lang package. | join() method, on the other hand, is also defined in java.lang package but in Thread class. |
| Usage | wait() method is primarily used for the inter-thread communication. | On the other hand join() is used for adding sequencing between multiple threads, one thread starts execution after first thread execution finished. |
| Counter method | As wait() method is used to pause the current thread so its counter method is also provided in object class to resume the thread which is notify() and notifyAll(). | On the other hand we can not break the waiting imposed by join() method without unless or interruption the thread on which join is called has execution finished. |
| Context | In order to call wait method we require synchronized block or method as if wait() method is called outside the synchronized context it will throw IllegalMonitorStateException. | On the other no such condition required for calling join() method and we can call join() method with and without synchronized context in Java.. |
| Lock Release | wait() releases the monitor or lock held on the object which wait is invoked on | On the other hand, calling join() method doesn't release any monitor or lock. |

Q20) What is random access ?

* RandomAccess : This interface introduced in Java version 1.4.
* It marks the implementations of list which can be accessed randomly.
* It is present in java.util.RandomAccess.
* Marker interface used by List implementations to indicate that they support fast random access

Q21) How you create a thread ?

Thread is nothing but smallest unit of program.

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

1)By extending Thread class

Thread class provide constructors and methods to create and perform operations on a thread.

Thread class extends Object class and implements Runnable interface.

Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void join():** waits for a thread to die.
4. **public int getPriority():** returns the priority of the thread.
5. **public int setPriority(int priority):** changes the priority of the thread.
6. **public String getName():** returns the name of the thread.

2).By implementing Runnable interface.

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

**public void run():** is used to perform action for a thread.

Q22) Deadlock in Java ?

Deadlock in Java is a condition where two or more threads are blocked forever, waiting for each other

Example: when two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone.

Q23) Life cycle of thread ?

In Java, a thread always exists in any one of the following states. These states are:

**New:** Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.

**Runnable:** A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.

**Running:** When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

**Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.

**Terminated:** A thread reaches the termination state because of the following reasons:

* When a thread has finished its job, then it exists or terminates normally.
* **Abnormal termination:** It occurs when some unusual events such as an unhandled exception or segmentation fault.

Q24) What is Synchronization and when we use ?

* Synchronization is a process of handling resource accessibility by multiple thread requests.
* The main purpose of synchronization is to avoid thread interference.
* At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time.

Q25) Difference between iterator and Listiterator ?

|  |  |
| --- | --- |
| **Iterator** | **ListIterator** |
| The Iterator traverses the elements in the forward direction only. | ListIterator traverses the elements in backward and forward directions both. |
| The Iterator can be used in List, Set, and Queue. | ListIterator can be used in List only. |
| The Iterator can only perform remove operation while traversing the collection. | ListIterator can perform ?add,? ?remove,? and ?set? operation while traversing the collection. |

Q26) Which one is failsafe(iterator) ?

The Fail Safe iterators use a copy of the collection to traverse over the elements.

Unlike the Fail Fast, they require more memory as they cloned the collection.

The examples of Fail Safe iterators are **ConcurrentHashMap, CopyOnWriteArrayList**, etc.

Q27) What is mean by failsafe and failfast?

* Iterators in Java are part of the Java Collection framework.
* They are used to retrieve elements one by one.
* The [Java Collection supports two types of iterators; Fail Fast and Fail Safe.](https://www.javatpoint.com/java-tutorial)
* These iterators are very useful in exception handling.

|  |  |  |
| --- | --- | --- |
| **Base of Comparison** | **Fail Fast Iterator** | **Fail Safe Iterator** |
| **Exception** | It throws a ConcurrentModificationException in modifying the object during the iteration process. | It does not throw Exception. |
| **Clone Object** | No clone object is created during the iteration process. | A copy or clone object is created during the iteration process. |
| **Memory utilization** | It requires low memory during the process. | It requires more memory during the process. |
| **Modification** | It does not allow modification during iteration. | It allows modification during the iteration process. |
| **Performance** | It is fast. | It is slightly slower than Fail Fast |
| **Examples** | HashMap, ArrayList, Vector, HashSet, etc | CopyOnWriteArrayList, ConcurrentHashMap, etc. |

Q28. ArrayList vs Vector

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| 1) ArrayList is **not synchronized**. | Vector is **synchronized**. |
| 2) ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity. | Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity. |
| 3) ArrayList is **not a legacy** class. It is introduced in JDK 1.2. | Vector is a **legacy** class. |
| 4) ArrayList is **fast** because it is non-synchronized. | Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object. |
| 5) ArrayList uses the **Iterator** interface to traverse the elements. | A Vector can use the **Iterator** interface or **Enumeration** interface to traverse the elements. |

Q30. Data structure where insertion order is preserved and duplicates are not allowed?

**Underline data structure** is hashtable duplicate objects not allowed. Insertion order not preserved. It is based on the hashcode of the object

Q31. what is the difference between sleep() & wait()?

|  |  |
| --- | --- |
| **Wait()** | **Sleep()** |
| The Wait() method is related to the Object class. | The Sleep () method is related to the Thread class. |
| It is not a static method. | It is a static method. |
| At the time of the Synchronization, the Wait() method releases obj. | At the time of the Synchronization, the Sleep() method doesn't release the obj, i.e., lock. |
| The Sleep() method has two overloaded methods, which are as follows:   * sleep(long milliseconds, int nanoseconds) * sleep(long milliseconds) | The Sleep() method has three overloaded methods, which are as follows:   * Wait() * wait(long timeout, int nanoseconds) * wait(long timeout) |
| The constructor of the Wait() method is defined in the following way: public final void Wait(long timeout) | The constructor of the Sleep () method in the following way: public static void Sleep (long millis) throws Interrupted\_Execption |

Q32.What is marker interface?

* An [interface](https://www.javatpoint.com/interface-in-java) that does not contain methods, fields, and constants is known as **marker interface.**
* In other words, an empty interface is known as **marker interface** or **tag interface.**
* The **Serializable** and **Cloneable** interfaces are the example of marker interface. In short, it indicates a signal or command to the JVM.
* The declaration of marker interface is the same as interface in Java but the interface must be empty. For example:

**public** **interface** Serializable   {

**}**

Q33. Can you tell mi different types of marker interface?

* In [Java](https://www.javatpoint.com/java-tutorial), built-in marker interfaces are the interfaces that are already present in the [JDK](https://www.javatpoint.com/difference-between-jdk-jre-and-jvm#jdk) and ready to use. There are many built-in marker interfaces some of them are:
* Cloneable Interface
* Serializable Interface
* Remote Interface

Cloneable Interface🡪

* **Cleanable interface** in Java is also a marker interface that belong to **java.lang** package.
* It generates replica (copy) of an object with different name.
* We can implement the interface in the class of which class object to be cloned.
* It indicates the **clone()** method of the Object class.
* If we do not implement the Cloneable interface in the class and invokes the clone() method, it throws the **ClassNotSupportedException.**

Serializable Interface🡪

* It is a marker interface in Java that is defined in the **java.io** package.
* If we want to make the class serializable, we must implement the **Serializable** interface.
* If a class implements the Serializable interface, we can serialize or deserialize the state of an object of that class.
* Serialization (converting an object into byte stream) is a mechanism in which **object state is read from the memory and written into a file or database.**

Remote Interface🡪

* **Remote interface** is a marker interface that belong to **java.rmi** package.
* It marks an object as remote that can be accessed from another machine (host).
* We must implement the Remote interface if we want to make an object as remote.
* It identifies the interfaces whose methods can be invoked from a non-local JVM.
* Any remote object must implement the interface directly or indirectly.

Q34. collection hierarchy.

<<INTERFACE>>

COLLECTION

<<INTERFACE>>

QUEUE

<<INTERFACE>>

LIST

<<INTERFACE>>

SET

PRIORITY

QUEUE

LINKEDLIST

VECTOR

ARRAYLIST

TREESET

<<INTERFACE>>

NEVIGABLESET

<<INTERFACE>>

SORTEDSET

LINKEDHASH

SET

HASHSET

Q35.Advantage of multithreading

* 1. It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.
  2. You **can perform many operations together, so it saves time**.
  3. Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

Q36. Why we dont avoid immutuable class in multithreading.

* Immutable classes are **easier to design, implement, and use than mutable classes**.
* Immutable objects are good Map keys and Set elements, since these typically do not change once created.
* Immutability makes it easier to write

Q37. "We create two objects of class obj1 and obj2. we call obj1.m1() on one thread and obj2.m2() on another thread, at the same time. what will be the behaviour? "

Q38. Thread state

* Life cycle of a Java thread starts with its birth and ends on its death.
* The start() method of the Thread class is used to initiate the execution of a thread and it goes into runnable state and the sleep() and wait() methods of the Thread class sends the thread into non runnable state.

Q39. Data structure

* Data Structures are a specialized means of organizing and storing data in computers in such a way that we can perform operations on the stored data more efficiently.
* Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc

Q40. What is growable array ?

* An array is a fixed size, homogeneous **data structure.**
* The limitation of arrays is that they're fixed in size. It means that we must specify the number of elements while declaring the array
* the concept of **dynamic array** comes into existence.
* dynamic array expends the size of the array dynamically.
* The dynamic array is a **variable size** list data structure.
* dynamic array grows automatically when we try to insert an element if there is no more space left for the new element.
* dynamic array allows us to add and remove elements.
* dynamic array allocates memory at run time using the heap. It can change its size during run time.

Q41. how to create multiple thread / Crete 10 thread at one time(logic use for loop to create multiple thread at one time)

Q42. can we write multiple main thread in single class?

Yes**, you can have as many main methods as you like**. You can have main methods with different signatures from main(String[]) which is called overloading, and the JVM will ignore those main methods. You can have one public static void main(String[] args) method in each class.

Q43. sleep() will release a lock or not ?

Sleep() method does not release the lock on object during Synchronization.

Q44. Why wait method in Object class not thread class?

* Shared objects allow threads to communicate by calling wait() , notify() And notifyAll() Methods, so these methods are in the object class.
* That's all about why wait(), notify() And notifyAll() methods Are in Object Class And Not in Thread Class.
* As every object in java has only one lock(monitor) and wait(),notify(),notifyAll() are **used for monitor sharing** thats why they are part of Object class rather than Thread class.
* Hence, wait() and notify() methods are defined in Object class rather than Thread class. If wait() and notify() were on the Thread instead then each thread would have to know the status of every other thread and there is no way to know thread1 that thread2 was waiting for any resource to access.

Q45. Why we always invoked start method not run method? Reason.

* It's due to the design of multithreading in Java.
* Calling start () will start a new Thread and calling run() method does not start a new Thread.
* start method of thread class is implemented as when it is called a new Thread is created and code inside run() method is executed in that new Thread.
* While if run method is executed directly than no new Thread is created and code inside run() will execute on current Thread and no multi-threading will take place

Q46. What is dead locking.

* A deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource.
* Deadlock occurs when a set of processes are in a wait state, because each process is waiting for a resource that is held by some other waiting process.
* Therefore, all deadlocks involve conflicting resource needs by two or more processes.

Q47. What is array?

* An **array** is a collection of items stored at contiguous memory locations.
* The idea is to store multiple items of the same type together.
* The length of an array is established when the array is created. After creation, its length is fixed.
* It is a data structure where we store similar elements. It holds homogeneous data elements.
* Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.



Advantages

* **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
* **Random access:** We can get any data located at an index position.

Disadvantages

* **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Q48. How u use synchronisation?

* Synchronization in Java is the capability to control the access of multiple threads to any shared resource.
* Java Synchronization is better option where we want to allow only one thread to access the shared resource.
* The synchronization is mainly used to
  + - To prevent thread interference.
    - To prevent consistency problem.

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Q49. How we can make thread safe?

There are four ways to achieve Thread Safety in [Java](https://www.geeksforgeeks.org/java-programming-basics/). These are:

1. Using [Synchronization](https://www.geeksforgeeks.org/synchronized-in-java/).
2. Using [Volatile Keyword](https://www.geeksforgeeks.org/volatile-keyword-in-java/).
3. Using [Atomic Variable](https://www.geeksforgeeks.org/atomic-variables-in-java-with-examples/).
4. Using [Final Keyword](https://www.geeksforgeeks.org/final-keyword-java/).

Q50. Suppose there is 1000 thread then how u can manage resources?

Q51. Why Runnable interface is preferred ?

* Runnable interface is always preferred because, the class implementing it can implement as many interfaces as a developer can, and also extend another class.
* Whereas extending the Thread class, it can not extend another class, as Java supports only single inheritance.

Q52. why list allowed duplicate elements ?

* List allows duplicates while Set doesn't allow duplicate elements . All the elements of a Set should be unique if you try to insert the duplicate element in Set it would replace the existing value.
* List permits any number of null values in its collection while Set permits only one null value in its collection

Q53. what is meant by legacy classes ?

* The classes which are comes from old generation are called legacy classes.
* All the legacy classes are synchronized. The **java.util**package defines the following **legacy** classes:

1. HashTable
2. Stack
3. Dictionary
4. Properties
5. Vector

Q54. comparable &comparator difference ?

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price. | The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc. |
| 2) Comparable **affects the original class**, i.e., the actual class is modified. | Comparator **doesn't affect the original class**, i.e., the actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is present in **java.lang** package. | A Comparator is present in the **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. |

Q55. Why null insertion is not possible in treemap?

TreeMap sorts elements in natural order and doesn't allow null keys because compareTo() method throws NullPointerException if compared with null.

Q56. Context switching in thread ?

* A context switch is the process of storing the state of a process or thread, so that it can be restored and resume execution at a later point.
* This allows multiple processes to share a single central processing unit (CPU), and is an essential feature of a multitasking operating system.

Q57. Difference between library and framework ?

|  |  |
| --- | --- |
| **Library** | **Framework** |
| A library is a group of reusable functions and classes which are used by the developers to perform several common tasks. | A framework is basically a piece of code that dictates the architecture or the structure of our project and also adds in our programs. |
| While using a library in the program, the user has full control when he uses the functions or classes from the Library. | In frameworks, code never called into a framework. Instead, the Framework calls you. |
| The Library helps in linking and the binding process | A Framework provides a standard way to develop and deploy applications |
| It can be incorporated seamlessly into projects that are in the developing phase. | It cannot be incorporated seamlessly into the developing process. But instead of it, we can use it when we start a new project. |
| Example: The jQuery is JavaScript library that simplifies in doom manipulations. | Example: The AngularJS is one of the JavaScript's framework for the dynamic web-applications |

Q58. What is Hashmap ?

* Java **HashMap** class implements the Map interface which allows us to store key and value pair, where keys should be unique.
* if you try to insert the duplicate key, it will replace the element of the corresponding key.
* It is easy to perform operations using the key index like updation, deletion, etc. HashMap class is found in the java.util package.
* HashMap in Java is like the legacy Hashtable class, but it is not synchronized.
* It allows us to store the null elements as well, but there should be only one null key.

Q59. How to create arraylist object ?

* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non [synchronized](https://www.javatpoint.com/synchronization-in-java).
* Java ArrayList allows random access because the array works on an index basis.
* In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
* We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases. For example:

ArrayList<Integer> al = **new** ArrayList<Integer>(); // works fine

Q60. Collection and collections .

Collection🡪

* Collection can simply be described as an Interface**.**
* With the help of Collection, we can easily group various objects into a single unit. Collection forms the root or head of the hierarchy of interfaces.
* The other sub interfaces of this hierarchy are Sets, Lists, Maps, Queue, Deque, etc

Collections🡪

Collectionsis simply a utility classthat is found in java. util package.

The methods coming under the Collections class are compulsorily static methods.

Q61. In which scenario you will go for list and set?

* If you want to store elements and want them to maintain an order on which they are inserted into a collection then go for List again, as List is**an ordered collection and maintain insertion order.**
* If you want to create a collection of unique elements and don't want any duplicate then choosing any Set implementation .

e.g. HashSet , LinkedHashSet or TreeSet .

Q62. Synchronised thread vs volatile keyword ?

|  |  |
| --- | --- |
| **Volatile Keyword** | **Synchronization Keyword** |
| Apply to variable | Apply to method & block |
| Volatile keyword is a field modifier. | Synchronized keyword modifies code blocks and methods. |
| The thread cannot be blocked for waiting in case of volatile. | Threads can be blocked for waiting in case of synchronized. |
| It improves thread performance. | Synchronized methods degrade the thread performance. |
| It synchronizes the value of one variable at a time between thread memory and main memory. | It synchronizes the value of all variables between thread memory and main memory. |
| Volatile fields are not subject to compiler optimization | Synchronize is subject to compiler optimization. |

Q63. Iterator vs collection ?

* Iterator can only move to next() element or remove() an element.   
  However Collection can add(), iterate, remove() or clear() the elements of the collection.
* Iterator provides a better speed than Collections, as the Iterator interface has limited number of operations.
* java.sql.SQLException extends Iterable. Hence it allows the caller to safely iterate over causes of SQLException.

* Iterators are used in Collection framework in Java to retrieve elements one by one.
* A Collection is a group of individual objects represented as a single unit. Java provides Collection Framework which defines several classes and interfaces to represent a group of objects as a single unit.

Q64. Array vs arraylist ?

|  |  |  |
| --- | --- | --- |
|  | **Array** | **ArrayList** |
| **Definition** | An **array** is a dynamically-created object. It serves as a container that holds the constant number of values of the same type. It has a contiguous memory location. | The **ArrayList** is a class of Java **Collections** framework. It contains popular classes like **Vector, HashTable**, and **HashMap**. |
| **Static/ Dynamic** | Array is **static** in size. | ArrayList is **dynamic** in size. |
| **Resizable** | An array is a **fixed-length** data structure. | ArrayList is a **variable-length** data structure. It can be resized itself when needed. |
| **Performance** | It performs **fast** in comparison to ArrayList because of fixed size. | ArrayList is internally backed by the array in Java. The resize operation in ArrayList slows down the performance. |
| **Primitive** | An array can store both **objects** and **primitives** type | We cannot store **primitive** type in ArrayList. It automatically converts primitive type to object. |
| **Length** | Array provides a **length** variable which denotes the length of an array. | ArrayList provides the **size()** method to determine the size of ArrayList. |
| **Single/ Multi-Dimensional** | Array can be **multi-dimensional**. | ArrayList is always **single-dimensional**. |

Q64. Stack Data-Structure ?

* A Stack is a linear data structure that follows the **LIFO (Last-In-First-Out)** principle.
* a stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.
* push()**:** When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
* pop()**:** When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.



* A stack can be implemented by means of Array, Structure, Pointer, and Linked List.
* Stack can either be a fixed size one or it may have a sense of dynamic resizing.
* Here, we are going to implement stack using arrays, which makes it a fixed size stack implementation.

Q65. what u think which gives best performance linked list and arraylist?

1) As explained above the insert and remove operations give good performance (O(1)) in LinkedList compared to ArrayList(O(n)). Hence if there is a requirement of frequent addition and deletion in application then LinkedList is a best choice.

2) Search (get method) operations are fast in Arraylist (O(1)) but not in LinkedList (O(n)) so If there are less add and remove operations and more search operations requirement, ArrayList would be your best bet.

Q66. push () & pop () methods implementation ?

A stack is data structure where we can add the element from the top and also remove the element from the top. It follows “Last in first out (LIFO)” principle.  
  
In this example, the push () method is used to add the element in the list and pop () operation is used to remove the top element from the list. The try and catch block is Import java.util.\*;

Class StackDemo

{

//porforming push operation

static void push(Stack st, int a)

{

st.push(new Integer(a));

System.out.println("Element "+a+" push to Stack");

System.out.println("Stack is: " + st);

}

//porforming pop operation

 static void pop(Stack st)

{

 Integer a = (Integer) st.pop();

System.out.println("Element "+a+" pop to the stack");

System.out.println("Stack is: " + st);

}

 public static void main(String[] args)

{

try  
  {

Stack stack = new Stack();

 System.out.println("Stack: "+stack);

 push(stack, 12);

 push(stack, 15);

 push(stack, 32);

 push(stack, 54);

pop(stack);

  pop(stack);

  pop(stack);

  pop(stack);

  pop(stack);

}

catch (EmptyStackException ex)

{

  System.out.println("Stack is empty");

}}}

Q67. what is default priority, min priority, max priorty of thread ?

* Each thread has a priority. Priorities are represented by a number between 1 and 10.
* In most cases, the thread scheduler schedules the threads according to their priority (known as preemptive scheduling).
* But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.
* Note that not only JVM a Java programmer can also assign the priorities of a thread explicitly in a Java program.

## 3 constants defined in Thread class

1. public static int MIN\_PRIORITY
2. public static int NORM\_PRIORITY
3. public static int MAX\_PRIORITY

* Default priority of a thread is 5 (NORM\_PRIORITY). The value of MIN\_PRIORITY is 1 and the value of MAX\_PRIORITY is 10.
* We know that a thread with high priority will get preference over lower priority threads when it comes to the execution of threads

  Q68. can we call run method by its name in multithreading?

* The **run()** method of thread class is called if the thread was constructed using a separate Runnable object otherwise this method does nothing and returns.
* When the run() method calls, the code specified in the run() method is executed.
* You can call the run() method multiple times.
* The run() method can be called using the start() method or by calling the run() method itself.
* But when you use run() method for calling itself, it creates problems.

Q69. what are ways to create thread & which is the best way.

here are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

### Thread class: 🡪

### Thread class provide constructors and methods to create and perform operations on a thread.

### Thread class extends Object class and implements Runnable interface.

### Runnable interface:🡪

* The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

### Q70. What are domain threads?

### **Daemon thread in Java** is a service provider thread that provides services to the user thread.

### Its life depends on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

### There are many java daemon threads running automatically e.g. gc, finalizer etc.

* it provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
* Its life depends on user threads.
* It is a low priority thread.

Q71. Can I store null key as a hashmap?

**It allows to store the null keys as well, but there should be only one null key object and there can be any number of null values.** This class makes no guarantees as to the order of the map.

### Q72. How do you iterate hashmap?

### [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/) is a part of Java’s collection providing the basic implementation of the Map interface of Java by storing the data in (Key, Value) pairs to access them by an index of another type.

### One object is listed as a key (index) to another object (value).

### If you try to insert the duplicate key, it will replace the element of the corresponding key.

### In order to use this class and its methods,it is necessary to import [java.util.HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/) package or its superclass.

### For each method(), for loop,

### Q73 What is the function of yield() method in multithreading?

* The **yield()** method of thread class causes the currently executing thread object to temporarily pause and allow other threads to execute.

1. Syntax- **public** **static** **void** yield()

* This method does not return any value.

Q74. stack and queue diff?

|  |  |
| --- | --- |
| **Stack** | **Queue** |
| It follows the principle LIFO (Last In- First Out), which implies that the element which is inserted last would be the first one to be deleted. | It follows the principle FIFO (First In -First Out), which implies that the element which is added first would be the first element to be removed from the list. |
| It does not have any types. | It is of three types like priority queue, circular queue and double ended queue. |
| It has a simpler implementation. | It has a comparatively complex implementation than a stack. |
| A Stack is visualized as a vertical collection. | A Queue is visualized as a horizontal collection. |
| It performs two operations, push and pop. The push operation inserts the element in a list while the pop operation removes the element from the list. | It performs mainly two operations, enqueue and dequeue. The enqueue operation performs the insertion of the elements in a queue while the dequeue operation performs the deletion of the elements from the queue. |

Q75. Difference Comparable & Comparator ?

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price. | The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc. |
| 2) Comparable **affects the original class**, i.e., the actual class is modified. | Comparator **doesn't affect the original class**, i.e., the actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is present in **java.lang** package. | A Comparator is present in the **java.util** package. |
| We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. | We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. |

Q76. By using comparator method with practical.

* **Java Comparator interface** is used to order the objects of a user-defined class.
* This interface is found in java.util package and contains 2 methods compare(Object obj1,Object obj2) and equals(Object element).

Example is explain in notes you can refer .

Q77. Array and collection difference?

|  |  |
| --- | --- |
| **Arrays** | **Collection** |
| Arrays due to fast execution consumes more memory and has better performance. | Collections, on the other hand, consume less memory but also have low performance as compared to Arrays. |
| Arrays are fixed in size i.e once the array with the specific size is declared then we can't alter its size afterward. | The collection is dynamic in size i.e based on requirement size could be get altered even after its declaration. |
| Arrays can hold the only the same type of data in its collection i.e only homogeneous data types elements are allowed in case of arrays. | Collection, on the other hand, can hold both homogeneous and heterogeneous elements. |
| Arrays can hold both object and primitive type data. | Arrays can hold both object and primitive type data. |
| Arrays due to its storage and internal implementation better in performance. | Collection on the other hand with respect to performance is not recommended to use. |

Q78. length and length() used for..?

## length

* An **array is an object** that holds a **fixed number of values of the same type.**
* The **length**variable in an array returns the **length of an array** i.e. a **number of elements stored in an array.**
* Once arrays are initialized, its **length cannot be changed**, so the length variable can directly be used to get the length of an array.
* The **length variable** is used only for an **array.**

## length()

* The **length()** method is **a static method of String class.**
* The**length()**returns **the length of a string object** i.e. the number of characters stored in an object.
* **String class** uses this method because the length of a string can be modified using the various operations on an object.
* The **String** class internally uses a **char[] array** that it does not expose to the outside world.

Q79. sizeof used for..?

The size () method of List interface in Java is used to get the number of elements in this list.

That is, this method returns the count of elements present in this list container.

Q80. what is generics ?

The **Java Generics** programming is introduced in J2SE 5 to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time.

There are mainly 3 advantages of generics. They are as follows:

**1)Type-safety:** We can hold only a single type of objects in generics. It doesn?t allow to store other objects.

**2) Type casting is not required:** There is no need to typecast the object.

**3) Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

Q83. what is synchronized block?

* Synchronized block can be used to perform synchronization on any specific resource of the method.
* Suppose we have 50 lines of code in our method, but we want to synchronize only 5 lines, in such cases, we can use synchronized block.
* If we put all the codes of the method in the synchronized block, it will work same as the synchronized method.
* A Java synchronized block doesn't allow more than one JVM, to provide access control to a shared resource.
* The system performance may degrade because of the slower working of synchronized keyword.
* Java synchronized block is more efficient than Java synchronized method.

Syntax:

**synchronized** (object reference expression) {

  //code block

}

Q84. What is mergeSort? explain the merge sort?

* Merge sort is the sorting technique that follows the divide and conquer approach.
* Merge sort is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the elements.
* It is one of the most popular and efficient sorting algorithm.
* It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves.
* We have to define the **merge()** function to perform the merging.

Q85. Concurrent HashMap ?

A hash table supporting full concurrency of retrievals and high expected concurrency for updates.

This class obeys the same functional specification as Hashtable and includes versions of methods corresponding to each method of Hashtable.

However, even though all operations are thread-safe, retrieval operations do not entail locking, and there is not any support for locking the entire table in a way that prevents all access.

This class is fully interoperable with Hashtable in programs that rely on its thread safety but not on its synchronization details..

Class declarartion in hashmap:

**public** **class** ConcurrentHashMap<K,V>

**extends** AbstractMap<K,V>

**implements** ConcurrentMap<K,V>, Serializable

Q86. Hashmap vs Hashtable ?

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| 1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code. | Hashtable is **synchronized**. It is thread-safe and can be shared with many threads. |
| 2) HashMap **allows one null key and multiple null values**. | Hashtable **doesn't allow any null key or value**. |
| 3) HashMap is a **new class introduced in JDK 1.2**. | Hashtable is a **legacy class**. |
| 4) HashMap is **fast**. | Hashtable is **slow**. |
| 5) HashMap is **traversed by Iterator**. | Hashtable is **traversed by Enumerator and Iterator**. |
| 6) Iterator in HashMap is **fail-fast**. | Enumerator in Hashtable is **not fail-fast**. |
| 7) HashMap inherits **AbstractMap** class. | Hashtable inherits **Dictionary** class. |

Q87. What is Difference between Arraylist and Hashset ?

|  |  |
| --- | --- |
| **ArrayList** | **HashSet** |
| ArrayList is the implementation of the list interface. | HashSet on the other hand is the implementation of a set interface. |
| ArrayList internally implements array for its implementation. | HashSet internally uses Hashmap for its implementation. |
| ArrayList maintains the insertion order i.e order of the object in which they are inserted. | HashSet is an unordered collection and doesn't maintain any order. |
| ArrayList allows duplicate values in its collection. | On other hand duplicate elements are not allowed in Hashset. |
| Any number of null value can be inserted in arraylist without any restriction. | On other hand Hashset allows only one null value in its collection,after which no null value is allowed to be added. |

Q88. Map Vs Set

|  |  |
| --- | --- |
| **Set** | **Map** |
| Set is used to construct the mathematical Set in Java. | Map is used to do mapping in the database. |
| It cannot contain repeated values. | It can have the same value for different keys. |
| We can easily iterate the Set elements using the keyset() and the entryset() method of it. | Map elements cannot be iterated. We need to convert Map into Set for iterating the elements. |
| Insertion order is not maintained by the Set interface. However, some of its classes, like LinkedHashSet, maintains the insertion order. | The insertion order is also not maintained by the Map. However, some of the Map classes like TreeMap and LinkedHashMap does the same. |
| Set doesn't allow us to add the same elements in it. Each class that implements the Set interface contains only the unique value. | Map contains unique key and repeated values. In Map, one or more keys can have the same values, but two keys cannot be the same. |

Q89. How to iterate list ?

In Java**, List** is is an interface of the **Collection framework**.

It provides us to maintain the ordered collection of objects.

The implementation classes of List interface are **ArrayList, LinkedList, Stack**, and **Vector**.

The ArrayList and LinkedList are widely used in [Java](https://www.javatpoint.com/java-tutorial).

we will use **ArrayList** to iterate list.

* Iterator🡪

to **iterate** over the Collections, such as List, Map, etc.

It contains two key methods next() and hasNaxt() that allows us to perform an iteration over the List.

**next():** The next() method perform the iteration in forward order.

It returns the next element in the List. It throws **NoSuchElementException** if the iteration does not contain the next element in the List.

This method may be called repeatedly to iterate through the list, or intermixed with calls to previous() to go back and forth.

**hasNext():** The hasNext() method helps us to find the last element of the List.

It checks if the List has the next element or not. If the hasNext() method gets the element during traversing in the forward direction, returns true, else returns false and terminate the execution.

Example🡪

**import** java.util.\*;

**public** **class** IterateListExample3  {

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

{

List<String> city = Arrays.asList("Boston", "San Diego", "Las Vegas", "Houston", "Miami", "Austin");

Iterator<String> cityIterator = city.iterator();

**while**(cityIterator.hasNext())

{

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

}

}

}

* [ListIterator](https://www.javatpoint.com/java-listiterator)🡪

is also an interface that belongs to java.util package. It extends **Iterator<E>** interface.

It allows us to iterate over the List either in forward or backward order.

The forward iteration over the List provides the same mechanism, as used by the Iterator.

We use the next() and hasNext() method of the Iterator interface to iterate over the List.

**import** java.util.\*;

**public** **class** IterateListExample4

{

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

{

List<String> city = Arrays.asList("Boston", "San Diego", "Las Vegas", "Houston", "Miami", "Austin");

ListIterator<String> listIterator = city.listIterator();

**while**(listIterator.hasNext())

{

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

}

}

}

Q90. How to print list using lambda ?

* The lambda expressions are inline code that implements a **functional interface** without creating an **anonymous class.**
* The **forEach()** method defined in an **Iterable**interface and accepts lambda expression as a parameter.

import java.util.\*;

public class ListIterateLambdaTest

{

 public static void main(String[] arg)

{

List<String> countryNames = new ArrayList<String>();

countryNames.add("India");

countryNames.add("England");

countryNames.add("Australia");

countryNames.add("Newzealand");

countryNames.add("South Africa");

// Iterating country names through forEach using Lambda Expression

countryNames.forEach(name -> System.out.println(name));

}

}

Q91. Explain Dequeue ?

A queue is a data structure in which whatever comes first will go out first, and it follows the FIFO (First-In-First-Out) policy.

Insertion in the queue is done from one end known as the **rear end** or the**tail,** whereas the deletion is done from another end known as the **front end** or the **head** of the queue.



Q92. Synchronized vs concurrent collection ?

|  |  |
| --- | --- |
| **Collections** | **Concurrent Collections** |
| Most of the classic classes in Java Collections such as Array List, Linked List, Hash Map etc. are not synchronized and are not thread safe in multi-threading environment. | On other hand Java introduces same classes in Concurrent Collections with implement synchronization in them which not only make these classes as Synchronized but also thread safe in nature. |
| We have some synchronized classes in traditional collections as well such as Vector and Stack but these classes uses lock over whole collection which reduces performance and speed of execution. | On other hand concurrent collections introduces concept of partial locking where it locks only part of collection in case of multi-threading environment which improves the performance and speed of collections in such environment. |
| In case of traditional collections if we try to modify a collection through separate thread during collection iteration then we got Runtime Exception ConcurrentModificationException. | On other hand one would not get such exception if deals with the concurrent collections i.e. concurrent collections allows modification in collection during its iteration. |
| Due to reason mentioned in above points traditional collections are not preferred in multi-threading environment. | On other hand Concurrent collections are primarily preferred in multi-threading environment. |
| Traditional collections are type of legacy collection in Java and are introduced before concurrent collections. | While concurrent collections are introduced in JDK 1.5 i.e. are introduced after traditional collections. |

Q93. Collection scenario related to ClassCasteException ?

* ClassCastException in java is a run time error it occurs when an object can not be casted to another type.
* An object is automatically upcasted to its super class type.
* You need not to mention class type explicitly. But, when an object is supposed to be downcasted to its sub class type, then you have to mention class type explicitly. In such case, there is a possibility of occurring class cast exception.
* In most of time, it occurs when you are trying to downcast an object explicitly to its sub class type.

Try to run below program.

**package** com;

**class** A

{

**int** i = 10;

}

**class** B **extends** A

{

**int** j = 20;

}

**class** C **extends** B

{

**int** k = 30;

}

**public** **class** ClassCastExceptionDemo

{

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

{

A a = **new** B();   //B type is auto up casted to A type

B b = (B) a;     //A type is explicitly down casted to B type.

C c = (C) b;    //Here, you will get class cast exception

System.out.println(c.k);

}

}

Q94. How to insert element in middle of array list ?

Elements can be added in the middle of an ArrayList by using the java.util.ArrayList.add() method.

This method has two parameters i.e. the index at which to insert the element in the ArrayList and the element itself.

If there is an element already present at the index specified by ArrayList.add() then that element and all subsequent elements shift to the right by one.

A program that demonstrates this is given as follows –

import java.util.ArrayList;

import java.util.List;

public class Demo

{

public static void main(String args[]) throws Exception

{

List aList = new ArrayList();

aList.add(0, "Apple");

aList.add(1, "Mango");

aList.add(2, "Banana");

aList.add(1, "Melon");

aList.add(3, "Guava");

System.out.println("The ArrayList elements are: " + aList);

}

}

Output-

The ArrayList elements are: [Apple, Melon, Mango, Guava, Banana]

Q95. Sort the element in array according to some order ?

we need to sort the given array in ascending order such that elements will be arranged from smallest to largest. This can be achieved through two loops. The outer loop will select an element, and inner loop allows us to compare selected element with rest of the elements.

**public** **class** SortAsc

 {

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

 {

**int** [] arr = **new** **int** [] {5, 2, 8, 7, 1};

**int** temp = 0;

 System.out.println("Elements of original array: ");

**for** (**int** i = 0; i < arr.length; i++)

 {

System.out.print(arr[i] + " ");

}

**for** (**int** i = 0; i < arr.length; i++)

 {

**for** (**int** j = i+1; j < arr.length; j++)

 {

**if**(arr[i] > arr[j])

{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

System.out.println();

 System.out.println("Elements of array sorted in ascending order: ");

**for** (**int** i = 0; i < arr.length; i++) {

System.out.print(arr[i] + " ");

}

}

}

Q100. For each method with example ?

* Java provides a new method forEach() to iterate the elements.
* It is defined in Iterable and Stream interface.
* It is a default method defined in the Iterable interface. Collection classes which extends Iterable interface can use forEach loop to iterate elements.
* This method takes a single parameter which is a functional interface. So, you can pass lambda expression as an argument.

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** ForEachExample {

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

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

gamesList.add("Football");

 gamesList.add("Cricket");

gamesList.add("Chess");

gamesList.add("Hocky");

System.out.println("------------Iterating by passing lambda expression--------------");

gamesList.forEach(games -> System.out.println(games));

}

}

Output:

------------Iterating by passing lambda expression------------

--------- Football ---Iterating by passing lambda expression-------------- Cricket

Chess

Hocky

Football

Cricket

Chess

Hocky