***(java.util.)COLLECTIONS***

Collections are like containers that group multiple items in a single unit,

Eg: jar of chocolates

Collections are used in every programming language and when java arrived it also came with few collection classes are 🡪vector,scalar,hashtable,array.

Java 1.2 provided collection framework that is architechture to represent and manipulate collections in java in standard way

* Interfaces
  + Java collections framework interfaces provide the abstract datatype to represent the collections
  + Root interface 🡪 java.util.collection (is on top of cllctn hriechy`)
  + Methods
    - Size()-returns no.of elements in the collection
    - Iterator()
    - Add()
    - Remove()
    - Clear()
* Some of the intefaces
  + Java.util.Map
  + Java.util.Set
  + Java.util.Queue
  + Java.util.List
* Implementation classes

Provides core implementation classes for collections,to create custom collection classes by extending them.

* + - Collection classes
      * Arraylist
      * Linked list
      * Hashmap
      * Treemap
      * Hashset
      * Treeset

🡪map is the only interface that doesn’t inherit collection interface

Benefits of java collections framework:

* Reduced development effort
* Increased quality
* Reusability and interoperability

***Java collection interface:***

JCI are the foundation of JcFramework

* All core collection interfaces are generic
* <E> syntax is for
* Helps in reducing run time error by type-checking the objectsat complie-time.

***COLLECTION INTERFACE****:*

***This is th***e root of the collection hierarchy

Collection represents a group of objects known as elements

* + Size()-returns no.of elements in the collection
  + isEmpty()- returns true if collection is having some elements

//check whether

* + iterator()-
  + boolean add()-ensures that this collection contains the specified element
  + remove()-
    - * BULKOPERATION METHODS
    - addAll()
    - removeAll()
    - containsAll()
    - retainAll()
    - clear

SET INTERFACE:

Collection that contains no duplicate elements, atmost one null element

🡪As implied by name this interface models mathematical set abstraction and is used to represent sets such as the deck of cards

Java platform contains three general purpose set-implementations

* HashSet
* LinkedHashSet
* TreeSet

Set Interface doesn`t allow random access to an element in the collection,

You can use iterator or *foreach* loop to traverse the entire

LIST INTERFACE:

***Ordered collection***

User can access the elements by their integer index, and do search for elements in the list. Remove/replace also

Allows duplicate elements,typically allow pairs of elements (equals()), They allow multiple null elements if they allow null elements at all

QUEUE INTERFACE:

Boolean add(E e)- if the element was added to this queue else its false

Boolean offer(E e)-

throws ClassCast, NullPointer, IllegalArgument Exception

E remove()-

E poll()- retrives and removes the head of this

E element()- used to retrieve but not used to remove the head of the queue

Returns head of the queue

Throws NoSuchElementException if the queue is empty

MapInterface:-

Helps us to maps(create)objects in form of keys and values(pairs)

Doesn’t contain duplicate values

Each key can map at most one value

* HashMap
* TreeMap
* linkedHashMap
  + put()-add an element
  + get()-retrive
  + containsKey()
  + containsValue()
  + size()
  + isEmpty()

***Java Collection classes***

Java collection frame wrok comes with many implementation clases for the interfaces

Most common implementations are arraylist,

HashMAp

,Hashset

***HashSet:-***

We can set the initial capacity and load factor of the collection

LoadFactor is a measure of how full the hashmap is allowed to get before its capacityis automatically increased

Implements cloneable,serializable,

🡪Implements set interface but it is supported by hashmap instance.

🡪it makes no guarantee as to the interation order of the set,in particular it doesnot guarantee the order will remain constant over time

🡪this class permits null elements

***Constructors:***

Public HashSet(){

Map=new HashSet();

}

🡪constructs a new emptyset, backing the hashmap instance has default intial capacity (16) and load Factor (.75)

Public HashSet(int initialCapacity){

Map=new HashSet<>(initialCapacity);

}

🡪

Public HashSet(Collection<? Extends E> c)

{

Map=new HashMap<>(Math.max(()()))

}

🡪

Public HashSet(){

Map=new HashSet();

}

C- create/

U- Read/retrive

R-update/modify

D-delete/drop/rename()/clear()

**TreeSet:-**

**Constructor:-**

**TreeSet(NavigableMap<E,Object> m){**

**This.m=m;**

**}**

**Constructs a new empty treeset,sorted according to the natural ordering of elements. All elements inserted into the set must implement the comparable Interface.**

🡪Constructs a new empty treeset,sorted according to thespecified comparetor. All elements inserted into the set must be mutually comparable.

**compare(e1,e2);**

public TreeSet(Comparator<? super E> comparator)

🡪 public TreeSet([Collection](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<? extends [E](https://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html)> c)

<https://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html>

Sorted-treeset

Unordered-Hashset

**ArrayList:**

Resizeable array of the list implementation.

Implements all list operations and permits all elements including null and

It is unsynchronised

Constructor:

public ArrayList(){

this elementData= ***DEFAULTCAPACITY-EMPTY-ELEMENTDaTA***

}

🡪Constructs an empty list with an initial capacity of ten.

<https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html#ArrayList()>

Methods:

🡪**trimTosize()**

public void trimToSize()

Trims the capacity of this ArrayList instance to be the list's current size. An application can use this operation to minimize the storage of an ArrayList instance.

#### 🡪 ensureCapacity

public void ensureCapacity(int minCapacity)

Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.

* minCapacity - the desired minimum capacity

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Collection | Ordering | KeyValue | Dulpicate | Nullvalue | ThreadSafe | Initial capacity | loadFactor |
| ArrayList | yes | no | Yes | Yes | No |  |  |
| LinkedList | Yes | no | Yes | Yes | No |  |  |
| Hashset | no | no | no | Yes | No |  |  |
| Treeset | Yes | no | no | no | No |  |  |
| Hashmap | no | Yes | no | Yes | No |  |  |
| Treemap | Yes | Yes | no | no | No |  |  |
| Hashtable | no | Yes | no | no | Yes |  |  |
| vector | Yes | no | Yes | Yes | Yes |  |  |

Curd 0p-hashmap

Cla dof{

Private int age;

Private string name;

Get\_\_\_\_\_\_\_;

Set\_-\_\_\_;

}

------------🡪pojo(plain od java object) beans

**Lambda Expressions**

Lambda expression is used to provide implementation of the functional interface. Its saves a lot of code.

Incase of lambda expression,we don’t need to define the method again for providing the implementation. Here we just write the implementation code

Functional Interface:

Lambda provides the implementation of the functional interface.

An interface which has only one abstract method is known as functional interface. Java provides annotation @FuntionalInterface.

===============Thread==================

**A thread is a single sequence of executable code with a larger program.**

All programs so far have used just one thread –the main thread that starts automatically when you run the program-but java lets you create programs that start additional threads to perform specific task.

MULTITHREADING IN JAVA:

It allows concurrent execution of two or more parts of the program for the maximum utilization of cpu. Each part of such program is called a thread. So threads are lightweight processes within a process

Thread creation:

In jaa thread can be created in two ways

1. by extending thread class
2. by implementing runnable interface

**by extending thread class:**

we create a class that extends java.lang.thread class. This class overrides the run() method available in the thread class

**Thread Creation By Implementing The Runnable Interface:**

**We**create a new class whi

**Thread name**

**Thread priority**

**Thread id**

**Race condition:-**

Occursin multi threaded environment when more thaan one thread try to access a shared resource at the same time.Note it is safe if multiple threads are trying to read a shared resource, as long as they are not trying to change it . since multiple threads try to race ech other to finish executing a methodthus the name race condition

**Synchronization:**

In many cases currently running threads share data and two threads try to do operations on the same variables at the same time . this often results in corrupt data as two threads try to operate on the same data.

A popular solution is to provide some kind of lock primitive. Only one thread can acquire a particular lock at any particular time. This can be achieved by using a keyword “synchronized”.

By using the synchronize only one thread can access the method at a time and a second call will be blocked until the first call returns or wait() is called inside the synchronized method.

**Deadlock:**

**Q&A**

Diff b/w run and start method

Diff b/w sleep() and wait()

|  |  |
| --- | --- |
| **Sleep()** | **wait()** |
| Doesn`t releases the lock | Releases the lock |
| It is a method in thread class | It is a method in object class |
| It is a static method | It is a non-static method |
| While in sleep the thread gets started after the specific ms/sec interval | To start thread again from wait(), you have to call notify(), or notifyAll() |
| No need to call sleep() from the synchronized context | Wait should be called only from the synchronized context |
| Sleep() doesn`t releases the lock on an object during synchronization | Wait() releases lock during the synchronization |
| Sleep method execution completes when the thread interruptsor time expired | Wait() method |

## +