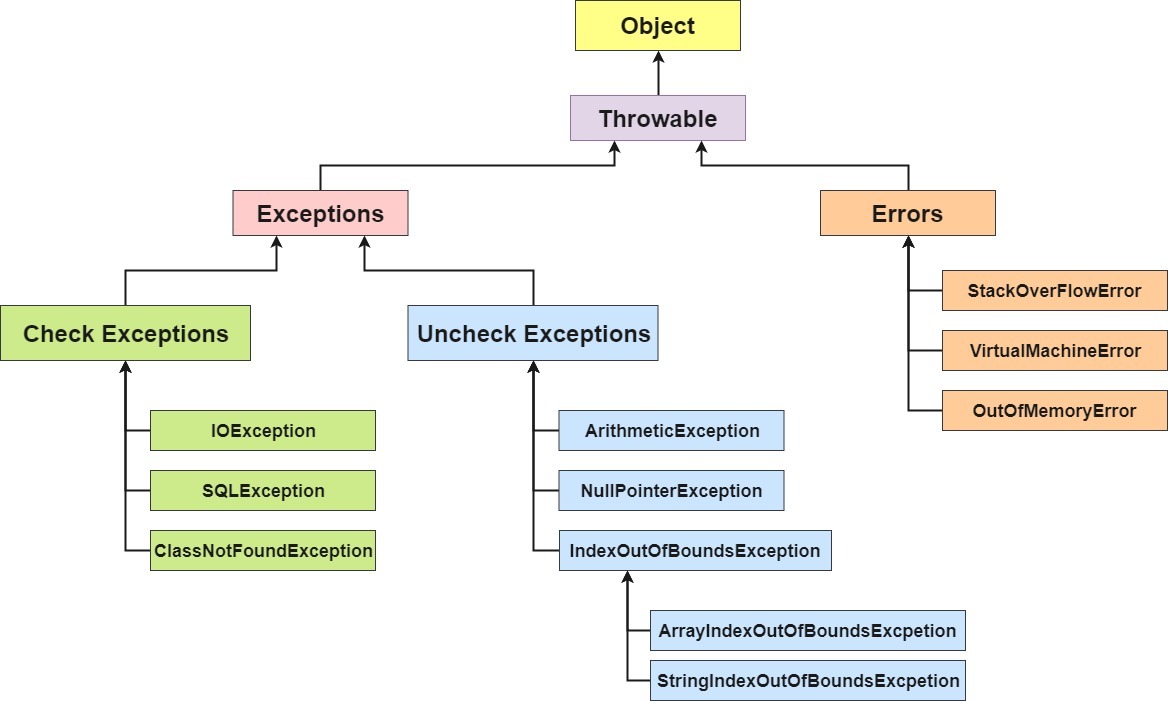
|  |  |  |
| --- | --- | --- |
| **Abstract Class** | | **Interface** |
| Extend only one abstract class. Both abstract & non abstract method. Contains Data Member, constructors. Does not support multiple inheritance. | | Implement multiple interfaces. Only abstract method. Does not contain Data Member, constructors. Support multiple inheritance. |
| **String** | | **String Buffer** |
| Immutable, Thread-safe, string constant pool, synchronization. String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance. | | Mutable, Thread-safe, heap memory, synchronization. String Buffer is fast and consumes less memory when concatenate too many strings. |
| **String Buffer** | **String Builder** | |
| Synchronized i.e., thread safe. It means two threads can’t call the methods of String Buffer simultaneously. | Non-synchronized i.e., not thread safe. It means two threads can call the methods of String Builder simultaneously. | |

|  |  |
| --- | --- |
| **Array** | **ArrayList (CLASS)** |
| Contain both primitives and objects. Not Type Safety i.e. holds same data type, homogeneous. Array does not provides remove an element. Use for loop or for each loop through iterate the Array. | Contains only objects. Type Safety means generics. ArrayList provides remove an element. Use iterate through iterate ArrayList. Maintain insertion order. |
| **ArrayList (CLASS)** | **Linked List (CLASS)** |
| Implements List. ArrayList internally uses a **dynamic array** to store the elements. Manipulation with ArrayList is **slow** because it internally uses an array. ArrayList is **better for storing and accessing** data. | implements List and Deque interfaces. LinkedList internally uses a **doubly linked list** to store the elements. Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list. LinkedList is **better for manipulating** data. |
| **ArrayList (CLASS)** | **Vector (CLASS)** |
| **Not Synchronized.** Faster. ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity. Is not a legacy class introduced in JDK 1.2? | **Synchronized i.e., slow.** Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity. Legacy class. |
| **Hash Map (CLASS)** | **Hash Set (CLASS)** |
| Implementation of Map interface. Store data in form of key value pair. Hash Map is faster because unique key. Put method | Implementation of Set interface. Store only object. Slower. Add method is used for add element. Duplicate value is not allow. Accept Null. |

**HashMap and Hashtable both are used to store data in key and value form. Both are using hashing technique to store unique keys.**

|  |  |
| --- | --- |
| **Hash Map (CLASS)** | **Hash-Table (CLASS)** |
| **Non synchronized**. **Allows one null key and multiple null values**. Fast. Iterator in HashMap is **fail-fast**. Inherits **Abstract Map** class. It maintains order. When one thread is adding or removing the value and at the same time T2 T3 T4 try to access particular value that value is gone from that object….immediately it gives you **concurrent modification error.** | **Synchronized**. **Doesn't allow any null key or value**. Slow. Legacy Class. Enumerator in Hash table is **not fail-fast**. Inherits **Dictionary** class. **Contains only unique values.** |
| **LIST (INTERFACE)** | **ArrayList (CLASS)** |
| **Interface, cannot be instantiated. List interface extends collection frame work.** | **Class, can be instantiated. Extend abstract list class and implement list interface.** |
| **List** maintain **insertion order** while **Set does** not maintain any order like HASH SET TREE SET. **List** allows **duplicates** while **Set** doesn't allow **duplicate** elements. | |
| **int [] a = new int [5]; a[0] = 10;** | |
| **ArrayList<String> aList = new ArrayList<String>(); aList.add("A");** | |
| **List<String> list = new ArrayList<String>(); list.add("Mango");** | |
| **LinkedList<String> link = new LinkedList<String>(); link.size();** | |
| **Map<Integer, String> map = new HashMap<Integer, String>(); map.put(100, "A"); Map has 02 interface and**  **for(Entry m : map.entrySet()) { three classes HashMap, LinkedHashMap, TreeMap**  **System.out.println(m.getKey()+" "+m.getValue());} //HashMap & LinkedHashMap allow null keys & values but TreeMap doesn’t allow null key or value.** | |
| **HashMap<Integer, String> hMap = new HashMap<Integer, String>();** | |
| **Set<String> hSet = new HashSet<String>(); hSet.add("Alpha");** | |
| **Hashtable<Integer, String> h1 = new Hashtable<Integer, String>(); h1.put(1, "A"); Clone possible** | |
| **Deque<Integer> deque = new ArrayDeque<Integer>(); deque.add(1); is an Interface** | |
| **Stack<Integer> stk = new Stack<>(); stk.push(78); is a CLASS; is a linear data structure that is used to store the collection of objects. It is based on LIFO.** | |
| **USER DEFINED CLASS OBJECT ARRAY LIST: declare variable, create a constructor using this keyword, create employee class object with initialize the value, create Array List & value, iterate to traverse the val** | |





|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ACCESS** | **CLASS** | **PACKAGE** | **OP BY SUBCLASS** | **OP** |
| **PRIVATE** | **Y** | N | **N** | **N** |
| **DEFAULT** | **Y** | **Y** | **N** | **N** |
| **PROTECTED** | **Y** | **Y** | **Y** | **N** |
| **PUBLIC** | **Y** | **Y** | **Y** | **Y** |