Java Collections

java.util.Arrays; ©

Arrays :- `String[] myStringArr={"A","B"};`

`Chr[] myChrArr={'A','B'};`

`int[] myIntArr = new int[]{1,2,3};`

`float[] myFloat = new float[]{459.432f, 348.43f};`

Syntax

`Data-Type[] variableName;`

and

`Data-Type variableName[];`

How many Ways

forEach loop →

```
for (int myInt : myIntArr){
    s.o.p(myInt);
}
```

```
for (int i=0; i<myInt.length; i++){
    myInt[i];
}
```

Elements starts from 0 to n;

→ Copying of Arrays from one Arry to another Arry

① `int[] a={1,2,3};`
`int[] b = a;`

② using for loop `for(int i=0; i<a.length; a++){ b[i]=a[i]; }`

③ `copyOf(objArr[] a, int length);`
`int[] b = Arrays.copyOf(a, a.length);`

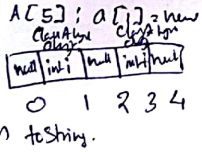
* Using clone:
`int b = a.clone();`

* `arrayCopy()` method of System class;
`System.arrayCopy(a,0,b,0,a.length);`

* Arrays containing Reference to Derived Type of Data:

```
ClassA {
    int i;
}
```

`A[] a; a=new A[5]; a[1]=new A(); a[3]=new A();`



* How to make arrays toString.
`Arrays.toString(a); output [1,2];`

* How to Sort Arrays. by using `Arrays.sort(a);`

* binary Search → Algorithm finds the position of a specified value ("the input key") within a sorted array.
or
half-interval Search
`int[] a={1,2,3,4}` `int Arrays.binarySearch(array, key);`
primitives 0 1 2 3 — return index of the key 3 → 3

* Static `int[] copyOfRange(array, int from, int to);` ②
index index.

* Static void `fill(array, primitive val)`
Assigning the specified primitive val to each element of the array.
`int[] a={1,2,3};` `Arrays.fill(a,5);` → `[5,5,5];`
→ from Index, to Index condition also.

* To Check the Equality of Two Arrays in Java
→ Iterative Method, `Arrays.equal()`, `Arrays.deepEquals()`;

→ `Arrays.deepToString()` method → this is used to print Content of the Array.
Eg:- `String[] twoDim = new Str[]{"one","two"}; Arrays.deepToString(twoDim);`

→ `Arrays.asList(arr)` → this asList (array) return the fixed length of list with specified array type. Eg: `String[] strArr={"one","two"}, List<Str> list = Arrays.asList(strArr);` Size

→ static int hashCode (primitive type Arr) → HashCode based on the Contents of the array.

→ Binary Search – Divided – Conquer – Search.
Half-interval Search Eg: `int[] arr={10,11,20,5,16};`
* step① → before search need to be in sort mode → `Arrays.sort(arr);` `int[] arr={10,11,16,20};`
* `nbs.binarySearch(arr,16);`

logic
```
int, start=0; int end = inputArr.length-1;
while (start<=end){
    int mid = (start+end)/2;
    if(Key == input[mid]){
        return mid;
    }
    if (key<inputArr[mid]){
        end = mid-1; } else { start = mid+1;
```

java.lang.String

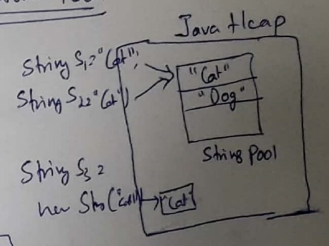
Ternary Operator -> boolean Expression ? expression1 : expression2

```
int y = 10;
int x = (y>5) ? (2*y) : (3*y);
```

Immutable class in java

1) Declare the class is final so it can't be extended.
   public final class String.
2) Make all fields private so that direct access is not allowed.
   private final char value[];
   private int hash;
3) Make all mutable fields final so that it's value can be assigned only once.
   private final char val[];
4) Initialize all the fields via a constructor performing deep copy.
   public String() {
   this.value = new char[0]; }
5) There is no setter methods i.e we have no option to change the value of the instance variable.
6) Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

String Constant Pool



S1 == S2 // true.
S1 == S3 // false.
S1.equals(S3) // true.

METHOD	SUMMARY
char	charAt(int index)
int	codePoint(int index) -> returns unicode at the specified index.
int	codePointBefore(int index)
int	codePointCount(int beginIndex, int endIndex)
int	compareTo(String anotherStr) -> compares two strings lexicographically. (+1, -1, 0)
int	compareToIgnoreCase(String str)
String	concat(String str);
boolean	contains(CharSequence s)
boolean	contentEquals(CharSequence cs)
static String	copyValueOf(char[] data);
static String	copyValueOf(char[] char, int offset, int count); (offset = starting pos., count = end point)
byte[]	getBytes() Encodes this String into a sequence of bytes using the given default charset storing the result into a new byte array.
int	hashCode();
int	indexOf(int ch) -> returns the index of 1st substring. Eg: String s1 = "This is index of example"; int index1 = s1.indexOf("is"); (2, 5, 4, 5, 6)
String	intern() -> a string that has the same content as this string. but is guaranteed to be from a pool of unique strings.
boolean	isEmpty();
int	length();
boolean	matches(String regex);
boolean	regionMatches(int toffset, String other, int ooffset, int len);
String[]	split(String regexp);
String[]	split(String regex, int limit)
String	subString(int beginIndex) (int beginIndex, int endIndex)
char[]	toCharArray() -> returns CharArray of the String.

ignoreCase, toffset - the starting offset of the other, ooffset - the starting offset of the Sub region in the String, len.

Difference Between Shallow Copy Vs Deep Copy in java

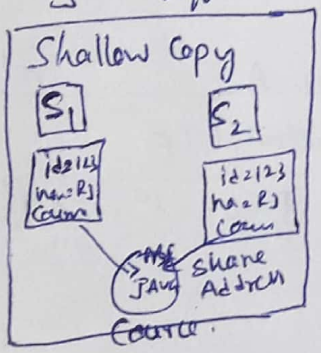
→ cloning is a process of creating an exact copy of an existing object in the memory. java.lang.Object → clone();

→ Note:- Not all the objects in java are eligible for cloning process

? → the objects which implements Cloneable (java.lang.I) interface. (marked interface)

Shallow Copy → the default version of clone() method creates the shallow copy of an object. It is a bit-wise copy of an object.

→ A new object is created that has an exact copy of the values in the original obj. If any of the fields of the object are references to other objects, just reffer addresses are copied. i.e only the memory Address is copied.




```
class Course {
    String cName;
    Course (String cName) {
        this.cName = cName;
    }
}

class D
psvm (String[] args) {
    Student s2 = null;
    Course c1 = new Course();
    Student s1 = new Student();
    s2 = (Student) s1.clone();
}
```

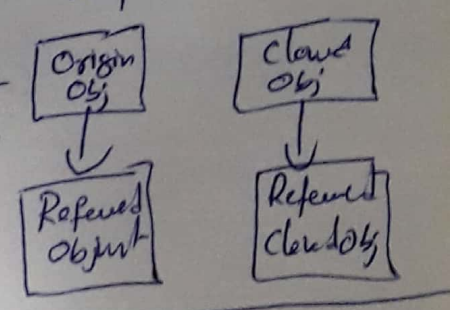
```
class Student implements Cloneable {
    String stName;
    Course c;
    protected Student (String stName, Course c) {
        this.stName = stName;
        this.c = c;
    }
    protected Object clone() {
        return super.clone();
    }
}
```

Deep Copy:-

A Deep Copy copies all fields and makes copies of Dynamically allocated memory pointed by the fields. A deep copy occurs when an Object is copied along with the object to which it refers.

Diff shallow Copy & Deep Copy in java

SC	DC
→ clone obj, orig obj not 100% Disjoint	→ cloned, origin obj 100% Disjoint
→ changes in clone obj will alter the orig obj	→ No reflection b/w clone & orig obj
→ Default version of clone method creates the shallow copy of an obj	→ To create DC of an obj you have to override clone method.
→ it is used if obj has primitive fields	→ it is used if an obj has reference to other objects as field.
→ SC is faster and less expensive	→ DC is slow and very expensive



StringBuilder vs StringBuffer

	Thread Safe	Immutability	Class / Interface	How to Concatinate	How to Create	Default Capacity, Length
java.lang. StringBuffer (C)	✓	× (it is mutable)	Class in java 1.0	append ( ) method	StringBuffer sb = new StringBuffer();	16
java.lang. StringBuilder (C)	×	× (it is mutable)	Class in java 5.0	StringBuilder append()	StringBuilder sb = new StringBuilder();	16

public void trimToSize() → Capacity becomes its length of the string inside buffer

String SubStr (int start)
(int start, int end)

StringBuffer/StringBuilder append (primitiveType name)
char[] str
char[] str, int offset, int len

appendCodePoint (int codePoint) → a unicode code point

int
SB/SB capacity() → gives the allowable number of characters this S.Bobject can still accommodate.

SB/SB ensureCapacity (int minimumCapacity) → Ensures that its capacity is at least equal to its specified minimum.

NOTE if minimum is argument is 20 → (minimum capacity) < (argument capacity) → allocated greater capacity
→ its capacity becomes twice its current capacity plus 2
→ its capacity is now (20*2)+2.

SB/SB delete (int start, int end) →
SB/SB deleteCharAt (int index)
SB/SB reverse()
SB/SB indexOf (String str) and indexOf (String str, int fromIndex)
int length()
void setCharAt (int index, char ch)
void setLength (int newLength)
CharSequence subSequence (int start, int end)

Collection FrameWork :- Group of objects stored in well defined manner.

Dis-Arrays → Earlier Arrays are used for these group of Objects.
→ Arrays are not re-sizable. Size of the arrays are fixed.
→ Size Can't be Changed once they are defined.

java.util → Contains the collections FW, legacy collection classes, event model, date and time facilities, i18n, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array]

Collection → java.util (I)
Interface Collection<E>
extends Iterable<T> (java.lang)

→ The root Interface in the Collection hierarchy.
→ Do not instantiate a Collection directly
→ but rather a subtype of Collection.
→ You may often treat these Subtypes uniformly as a collection.

Collection Subtypes :- following interfaces (collection types) extends the Collection Interface. (extends)

Collection ← List, Set, SortedSet, NavigableSet, Queue, Deque

Ex List extends Collection.

Ex:- class MyColl{
static void doSome(Collection coll){
for(int i : coll){
s.o.p(i);
}

Adding and Removing Collection: Set set = new HashSet(); List list = new ArrayList();
MyColl.doSome(set);
MyColl.doSome(list);

Methods

boolean add(E e)
boolean addAll(Collection<? extends E> c)
void clear()
int size();
boolean isEmpty()
boolean contains(Object o)
Iterator<E> iterator(); (method of Iterable)
Object[] toArray();
T[] toArray(T[] a)

boolean remove(Object o);
boolean containsAll(Collection<?> c);
boolean removeAll(Collection<? extends E> c);
boolean retainAll(Collection<?> c);
boolean equal(Object o);
int hashCode();

static<T> Collections.SynchronizedCollection(Collection) (Map, List, Set) threadsafe

Collections.Shuffle(List);
Collections.fill(List, 0.0);
Collections.min(List)
max(List)
reverse(List)
Collection.copy(destList, sourceList)

java.lang
Interface Iterable<T>

Methods
Iterator<T> iterator() } since 1.5
default void forEach(Consumer<? super T> action)
default spliterator<T> Spliterator(); } 1.8

java.util
Interface Spliterator<T>

int characteristics()
long estimateSize()
default void forEachRemaining(Consumer<? super T>)
default Comparator<? super T> getComparator()
default long getExactSizeIfKnown()
default boolean hasCharacteristics(int Characteristics)
boolean tryAdvance(Consumer<? super T> action)
Spliterator<T> trySplit();

Ex Emp um1 = new Emp("MyName");
List<Emp> Collections.singletonList(obj);
list.add(um2); Exception: UnsupportedOperationException

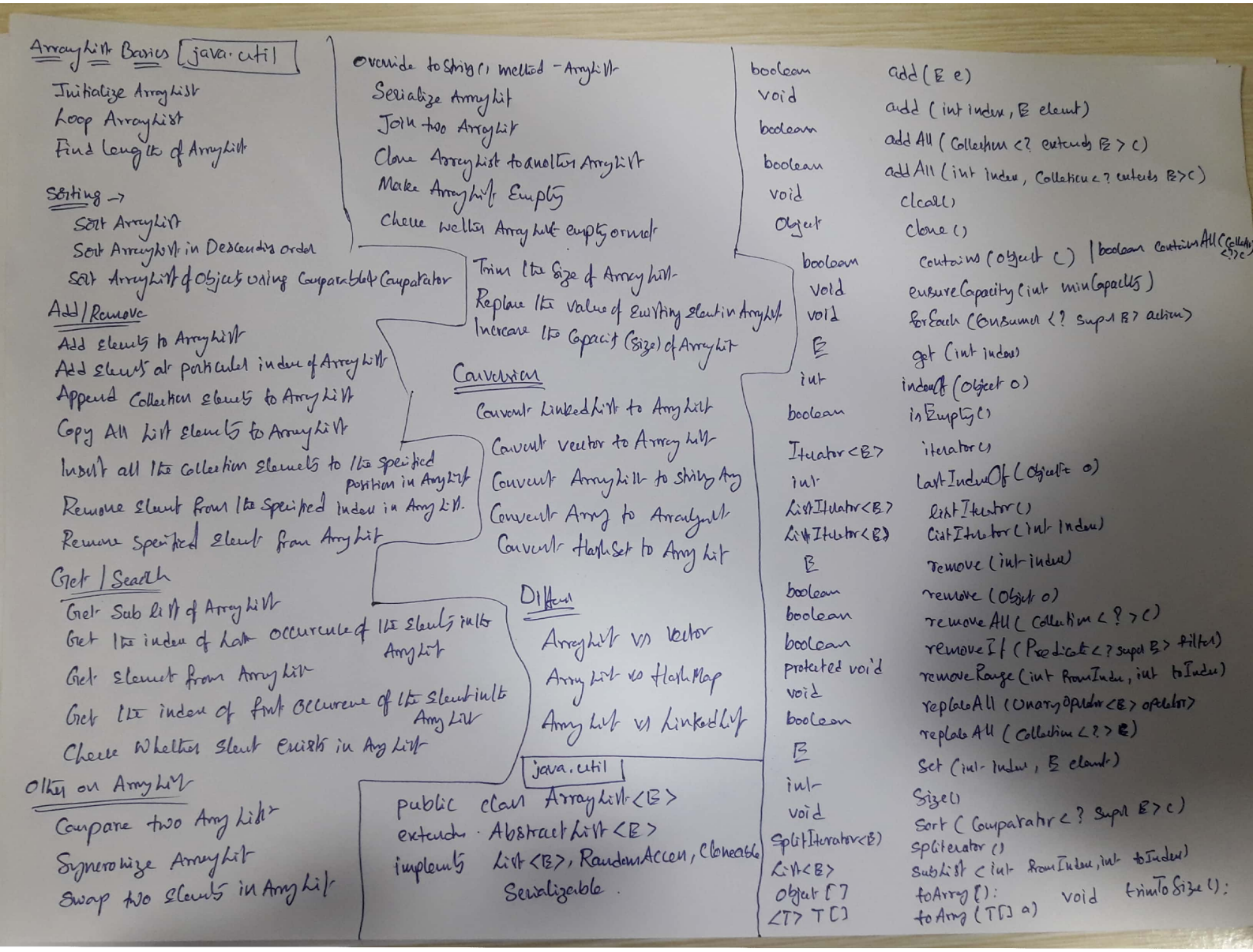
java.util.
Class Collections
extends Object.

→ This class Consists exclusively of static methods that operates on return Collections.
→ It Contains polymorphic algorithms that operate on Collections, "wrappers" which return a new Collection backed by a specified Collection.
→ If Collections class object is null it will throw NullpointerExcp.

static<T> boolean addAll(Collection<? super T>, T...elements)
static<T> Queue<T> asLifoQueue(Deque<T> deque)
static<T> int binarySearch(List<? extends Comparable<? super T>> list, T key)
static<E> Collection<E> checkedCollection(myList, String.class) (wrapper) (List, Set, Map)
static<T> void copy(sourceList, destList)
static boolean disjoint(Collection<?> c1, Collection<?> c2)

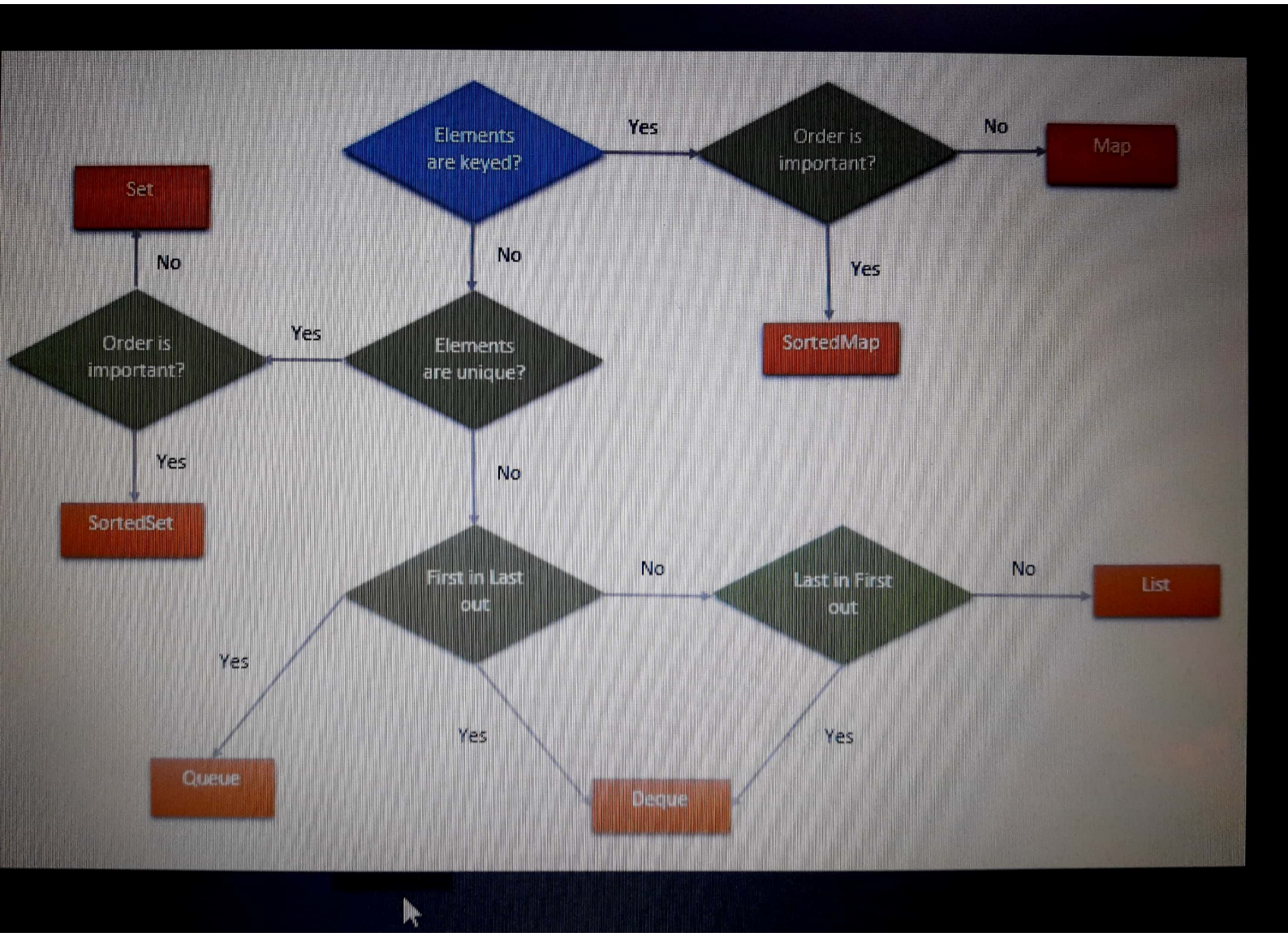
Find common Element count: not Exist → true; Exist → false.

static<T> Enumeration<T> EmptyEnumeration(), emptyIterator(),
static<T> List (Map) unmodifiableMap(Map); (Collection, List, Map, Set) throws UnSupportedOperationException
void swap(List, int i, int j)



	Size	They can hold	Iteration	How to get size?	Generics	Type Safe	Multi-dimensional	How to add elements?
Array	Fixed	Primitives as well as objects	Only through *for* loop or *for-each* loop	*Length* attribute	Doesn't support	No	Yes	Using assignment operator
ArrayList	Re-sizable	Only objects	Iterators or *for* loop or *for-each* loop	*size()* method	supports	Yes	No	Using *add()* method

	ArrayList	LinkedList	Vector	Stack
Duplicates	Allow	Allow	Allow	Allow
Order	Insertion order	Insertion order	Insertion order	Insertion order
Insert / Delete	Slow	Fast	Slow	Slow
Accessibility	Radom and fast	Sequential and slow	Random and fast	Slow
Traverse	Uses Iterator / ListIterator	Uses Iterator / ListIterator	Enumeration	Enumeration
Synchronization	No	No	Yes	Yes
Increment size	50%	No initial size	100%	100%



	ArrayList	LinkedList
Structure	ArrayList is an index based data structure where each element is associated with an index.	Elements in the LinkedList are called as nodes, where each node consists of three things – Reference to previous element, Actual value of the element and Reference to next element.
Insertion And Removal	Insertions and Removals in the middle of the ArrayList are very slow. Because after each insertion and removal, elements need to be shifted.	Insertions and Removals from any position in the LinkedList are faster than the ArrayList. Because there is no need to shift the elements after every insertion and removal. Only references of previous and next elements are to be changed.
Retrieval (Searching or getting an element)	Insertion and removal operations in ArrayList are of order O(n). Retrieval of elements in the ArrayList is faster than the LinkedList . Because all elements in ArrayList are index based.	Insertion and removal in LinkedList are of order O(1). Retrieval of elements in LinkedList is very slow compared to ArrayList. Because to retrieve an element, you have to traverse from beginning or end (Whichever is closer to that element) to reach that element.
Random Access	Retrieval operation in ArrayList is of order of O(1). ArrayList is of type Random Access. i.e elements can be accessed randomly.	Retrieval operation in LinkedList is of order of O(n). LinkedList is not of type Random Access. i.e elements can not be accessed randomly. you have to traverse from beginning or end to reach a particular element.
Usage	ArrayList can not be used as a Stack or Queue.	LinkedList, once defined, can be used as ArrayList, Stack, Queue, Singly Linked List and Doubly Linked List.
Memory Occupation	ArrayList requires less memory compared to LinkedList. Because ArrayList holds only actual data and it's index.	LinkedList requires more memory compared to ArrayList. Because, each node in LinkedList holds data and reference to next and previous elements.
When To Use	If your application does more retrieval than the insertions and deletions, then use ArrayList.	If your application does more insertions and deletions than the retrieval, then use LinkedList.

	Vector	ArrayList
Synchronization and thread-safety	Vector is synchronized means that all the method which structurally modifies Vector e.g. add () or remove () are synchronized which makes it thread-safe and allows it to be used safely in a multi-threaded and concurrent environment.	ArrayList methods are not synchronized thus not suitable for use in multi-threaded environment.
Speed and Performance	Since Vector is synchronized and thread-safe it pays price of synchronization which makes it little slow.	ArrayList is way faster than Vector.ArrayList is not synchronized and fast which makes it obvious choice in a single-threaded access environment. You can also use ArrayList in a multi-threaded environment if multiple threads are only reading values from ArrayList or you can create read only ArrayList as well
Capacity	Vector crossed the threshold specified it increases itself by value specified in capacityIncrement field	increase size of ArrayList by calling ensureCapacity () method.
Enumeration and Iterator	Vector can return enumeration of items it hold by calling elements () method	
Legacy	Vector is one of those classes which comes with JDK 1.0 and initially not part of Collection framework but in later version it's been re-factored to implement List interface	