* Interfaces

Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body).

⇒ Interface is a collection of abstract methods.

To declare an interface, use interface keyword. A class that implement interface must implement all the methods declared in the interface. To implement interface use "implements" keyword.

⇒ Interface have the following properties:

1. An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface.
2. Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
3. Methods in an interface are implicitly public.

Syntax

```
interface interfacename
{
    // declare constant fields
    // declare methods that abstract.
}
```

Java Interface	Abstract Class
1. All the methods are abstract methods.	1. Class can have some concrete method along with abstract method.
2. We use implement keyword. either class → interface or interface → class.	2. We use extends keyword.
3. We don't need to use the abstract keyword because interface are implicitly abstract.	3. We need to declare the abstract keyword.
4. The method need not to be declared with abstract keyword because it's implicitly abstract.	4. The method is declared with abstract keyword without any implementation.
5. We get 100% abstraction	5. We can't achieve because abstract class can contain non-abstract method as well.
6. Multiple inheritance is possible.	6. Not possible.

* Multiple Inheritance

Of a class ~~inherits~~ implements multiple ~~inheritance~~ interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.

Class → Class = Extends

Class → Interface = Implements (vice-versa)

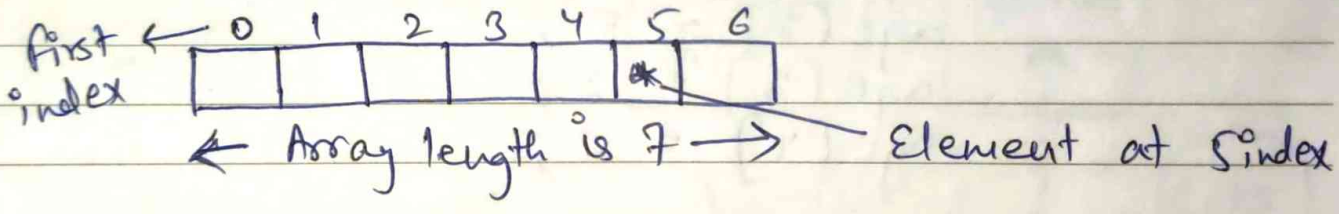
Interface → Interface = Extends

JAVA ARRAYS

An array is a collection of similar types of data.

Java array is an object which contains elements of a similar data type, Additionally, the elements of an array are stored in a contiguous memory ~~location~~ location, We can store only a fixed set of elements in a Java array.

⇒ 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 & so on.



eg. String[] arr = new String(100)

⇒ Here,

the above array cannot store more than 100 words.

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

Syntax : Datatype[] arrayname;

eg. double[] arr;
String[] bh;

Correct way to declare an array.

1. datatype[] arr;
2. or, datatype [] arr;
3. or, datatype arr[];

Memory allocation arr = new int[10];

Declaration & memory allocation

int[] arr = new int[100];

Declaration & initialization

int[] age = {12, 4, 5, 2, 5};

or,

age[0] = 12;
age[1] = 4;
age[2] = 5;
age[3] = 2;
age[4] = 5;

* For-each loop for Java Array

We can also print the Java array using for-each loop. The Java for-each loop prints or accepts the array elements one by one.

Syntax

```
for(datatype variable : array) {
    // body of the loop.
}
```

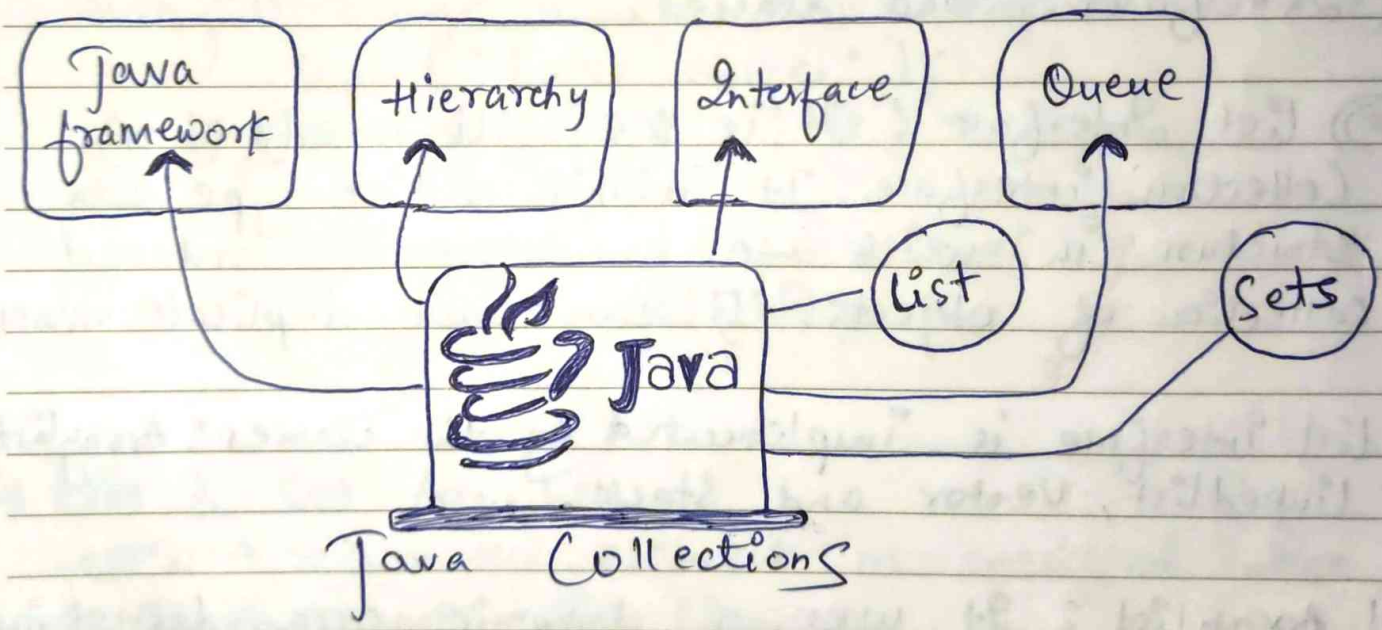
* Print the array from Console.

```
Scanner sc = new Scanner(System.in);
Sopln("Enter the array");

int[] arr = new int[5];

for(int i=0; i<5; i++)
    arr[i] = sc.nextInt();
}

// To print the data
for(int result : arr)
    Sopln(result);
```



It is a framework that provides an architecture to store and manipulate the group of objects.
In Java, a separate framework named the "Collection Framework" has been defined in JDK 1.2 which holds all the collection classes & interface in it.

The collection interface (Java.util.Collection) and Map interface (Java.util.Map) are the two main "root" interface of Java collection classes.

Q. What is a Framework?
A framework is a set of classes and interfaces which provide a ready-made architecture. In order to implement a new feature or a class, there is no need to define a framework.

Q. What is a Java Collection Framework?
It provides an architecture to store & manipulate a group of objects. It includes the following: Interfaces and classes.

⇒ List Interface: It is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List interface is implemented by the classes: Arraylist, Linkedlist, Vector and Stack.

1. Arraylist: It uses a dynamic array for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. Can be found in Java.util. package.

```
Arraylist list = new Arraylist(); // creating non-generic
                                     arraylist.

Arraylist<String> list = new Arraylist<String>();
                          // creating generic arraylist.
```

In a generic collection, we specify the type in angular braces. Now arraylist is forced to have the only specified type of object in it.

Using Console

```
Scanner sc = new Scanner(System.in)
Arraylist<String> name = new Arraylist<String>();
        Sopln("Enter names");
    for(int i=0; i<2; i++)
      {
          String value = sc.nextLine();
            name.add(value);
        }
```

* Get & Set Arraylist.

get() → return the element at specified index.
set() → changes the element.

```
Sopln(name);
Sopln(name.get(1)); // it returns 2nd element
Sopln(name.set(1,"Aadi")); // Change element.
```

Eg. ["Mango", "Apple", "Banana", "Grapes"]

Apple.

["Mango", Aadi, Banana, Grapes]

⇒ Set Interface: It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store atmost one null value in Set. Set is implemented by HashSet, LinkedHashSet and TreeSet class.

```
Set<datatype> sl = new HashSet<datatype>();
```

- HashSet: It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.

⇒ Map Interface: It contains value on the basis of key, i.e., key and value pair. Each key and value pair is known as an entry.
- A Map contains unique keys.
- A Map is useful if you have to search, update, or delete elements on the basis of a key.

HashMap class Parameters.
K: Type of keys maintained by this map.
V: Type of mapped values.

→ No order, non-synchronized.
→ May have one or multiple key null values.
→ Contains values based on key.
→ Unique keys.

```
HashMap<key, value> map = new HashMap
                          <key, value>();
```

eg.

```
HashMap<Integer, String> map = new HashMap
                               <Integer, String>();
    map.put(1, "Mango");
    map.put(2, "Orange");
    map.put(3, "Apple");

    Sopln(map);
    Sopln(map.get(3)); // Apple.
```