## A. ABSTRACT CLASS AND INTERFACE USAGES

### Abstract class:

- It is mainly used to define default behavior for subclasses, it means that all child classes should have performed the same functionality.
- It is a good choice when considering the inheritance concept since it provides a common base class implementation to derived classes.
- It is good for the declaration of non-public members.
- It is a better choice if new methods will be added in future.
- It should be used if multiple versions of components will be created. It provides easy way to version components by updating the base class, hence all inheriting classes are automatically updated.
- It is better used if common implemented functionality among implementations of component will be provided. This is because unlike interface, abstract class allow partial implementation of classes.
- Abstract class is an excellent choice when designing large functional units.

### Interface:

- It is used to define a contract behaviour and it can also act as a contract between two systems to interact.
- If a wide range of disparate objects will have common functionality, an interface is useful in such case.
- Since multiple inheritance is not supported in java, interface is a good choice.
- It is a good choice when designing small, concise bits of functionality.

## B. DIFFERENCES BETWEEN ARRAY AND ARRAYLIST

**Description**

**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.

**ArrayList** is a class of java collections framework. It contains popular classes like Vector, Hashtable and Hashmap.

**Flexibility**

**Array** is static in size.
**ArrayList** is dynamic in size.

**Resizable**

**Array** is a fixed length data structure.

**ArrayList** is a variable-length data structure. It can resize itself when needed.

**Initialization**

It is mandatory to provide the size of an **Array** while initializing it directly or indirectly.
The instance of an **ArrayList** without specifying its size.

**Performance**

**Array** 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 / Generic Type**

**Array** can store both objects and primitive type.

**ArrayList** cannot store primitive type. It automatically converts primitive type to object.

## Iterating Values

**Array** can use **for** loop or **for each** loop to iterate over an  array.

**ArrayList** uses an iterator to iterate over ArrayList.

## Type-Safety

**Array** cannot use generics because it is not a convertible type of array.

**ArrayList**  allows the storage of only generic /type. And that’s why it is type-sefe

## Length

**Array** provide a length variable which denotes the length of an array.

**ArrayList** provides the size() method to determine the size of ArrayList.

## Adding Elements

**Array** adds element in an array by using assignment operator(“=”).

Java provides the **add()** method to add elements in the **ArrayList.**

## Single/Muti-Dimenional

**Array** can be multi-dimensonal.

**ArrayList** is always single-dimensional.