**Collections Framework**

Topics

**1. Introduction to Collections Framework**

> A collections framework is a unified architecture for representing & manipulating collections, enabling collections to be manipulated independently of implementation details. All collections frameworks contain the following:

**a) Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In Object oriented language, interfaces generally form a hierarchy.

**b) Implementations:** These are the concrete implementations of the collections interfaces. In essence, they’re reusable data structures.

**c) Algorithms:** These are the methods that perform useful computations, such as searching & sorting, on objects that implement collection interfaces. The algorithms are said to be polymorphic i.e. same method can be used on many different implementations of the appropriate collection interface.

Apart from the Java collections framework, the best – known examples of collections frameworks are the C++ Standard Template library (STL).

**> Benefits of the Java Collections Framework**

**1.** **Reduces programming effort**: By providing useful data structures & algorithms so you don’t have to write them yourself.

**2. Increases program speed & quality:** This collections framework provides high – performance, high – quality implementations of useful data structures & algorithms. The various implementations of each interface are interchangeable, so programs can be easily tuned by switching collection implementations. Because you’re freed from the drudgery of writing our own data structures, you’ll have more time to devote to improving program’s quality & performance.

**3. Allows interoperability between unrelated**: The collection interfaces are the vernacular by which APIs pass collections back & forth. If my network administration API furnished a collection of node names & if your GUI toolkit expects a collection of column headings, our APIs will interoperated seamlessly, even though they were written independently.

**4. Reduces the effort required to learn & to use new APIs**

**5.** **Reduces the effort required to design & implement:** Designers & Implementers don’t have to reinvent the wheel each time they create an API that relies on collections; instead, they can use standard collection interfaces.

**6. Fosters software reuse:** Fosters software reuse by providing a standard interface for collections & algorithms with which to manipulate them.

**2. Array**

> An array is an indexed collection of fixed no. of homogenous data elements.

> The main advantage of Array is we can represent multiple values by using single variable so that readability of the code will be improved.

> Limitations of Array are

**a)** Fixed in size i.e. once we create an array, there is no chance of increasing/decreasing the size based on our requirement. Due to this, to use Array’s concept, compulsory we should know the size in advance which may not possible always.

**b)** Array can hold only homogenous datatype elements. (Though we can use **Object** type array)

**c)** Array concept is not implemented based on some standard data Structure & hence readymade method support is not available i.e. for every requirement, we have to write the code explicitly which increases complexity of programming.

> To overcome above limitations of Array, we should go for Collection concept.

> Advantages of Collection are

a) Collections are growable in nature i.e. based on our requirement we can increase/decrease the size.

b) Collections can hold both homogenous & heterogeneous elements.

c) Every collection class is implemented based on some standard data Structure hence for every requirement, readymade method support is available.

**> Difference b/w Array & Collection**

|  |  |  |
| --- | --- | --- |
| SNo | **Array** | **Collection** |
| 1. | Fixed in size. | Growable in nature. |
| 2. | With respect to memory, Arrays are not recommended to use. | With respect to memory, Collections are recommended to use. |
| 3. | With respect to performance, Arrays are recommended to use. | With respect to performance, Collections are not recommended to use. |
| 4. | Arrays can hold only homogenous datatype elements. | Collections can hold both homo & heterogeneous datatype elements. |
| 5. | No underlying data structure hence no readymade methods are available. | Every collection class has underlying data structure hence readymade methods are available. |
| 6. | Arrays can hold both primitive & objects. | Collections can hold only object types but not primitive. |

**3. Collection**

> A collection – sometimes called a container – is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate & communicate aggregate data.

> Typically, they represent data items that form a natural group, such as poker hand (a collection of cards), a mail folder (a collection of letters), or a telephone directory (a mapping of names to phone numbers).

> The collection interfaces are divided into 2 groups i.e. **java.util.Collection** & **java.util.Map**

**4. 10 key interfaces of Java Collections Framework**

1. Collection Interface

2. List Interface

3. Set Interface

4. SortedSet Interface

5. NavigableSet Interface

6. Queue Interface

7. Deque Interface

8. Map Interface

9. SortedMap Interface

10. NavigableMap Interface

**1. Collection Interface**

> If we want to represent a group of individual objects as a single entity then we should go for Collection.

> The core collection interfaces encapsulate different types of collections like List interface, Set Interface, Queue Interface, Deque Interface etc.

> These interfaces allow collections to be manipulated independently of the details of their representation.

> Collection interface defines the most common methods which are applicable for any collection object.

> In general, Collection interface is considered as root interface of Collections framework.

> There is no concrete class which implements Collection interface directly.

**Set (I)**

**Deque (I)**

**List (I)**

**Queue (I)**

**Collection (I)**

*all interface extends Collection interface.*

Note:

> There is also a Collections class in java.util package having utility methods for Collection objects (like sorting, searching etc.)

**2. List (Interface) [Order Collection or Sequence]**

> It is child interface of Collection interface.

> If we want to represent a group of individual objects as a single entity where duplicates are allowed & insertion order must be preserved then we should go for list.

**Collection (I) (1.2 V)**

**List (I) (1.2 V)**

implements

***Legacy Classes***

**3. Set (Interface)**

> It is the child interface of Collection interface.

> If we want to represent a group of individual objects as a single entity where duplicates are not allowed & insertion order not required to be preserved then we should go for Set interface.

**4. SortedSet (Interface)**

> It is the child interface of Set interface.

> If we want to represent a group of individual objects as a single entity where duplicates are not allowed & all objects should be inserted according to some sorting order then we should go for SortedSet.

**5. NavigableSet (Interface)**

> It is the child interface of SortedSet interface.

> It contains several methods for Navigation purposes.

Collection (I) (1.2 V)

Set (I) (1.2 V)

SortedSet (I) (1.2 V)

NavigableSet (I) (1.6 V)

**> Difference b/w List & Set**

|  |  |  |
| --- | --- | --- |
| **No.** | **List** | **Set** |
| 1. | Duplicates are allowed. | Duplicates are not allowed. |
| 2. | Insertion order preserved. | Insertion order not preserved. |

**6. Queue**

> It is the child interface of Collection interface.

> If we want to represent a group of individual objects prior to processing then we should go for Queue.