COLLECTIONS

Limitations of Arrays

1. Arrays allow storing collection of homogeneous elements. They won't support heterogeneous elements.

2. Arrays are fixed size collections whose size cannot grow/shrink at runtime.

3. Arrays support static memory allocation which causes memory wastage.

4. Arrays don't have any readymade methods to manage the elements inside the arrays.

Note: All the above drawbacks can be overcome using collections.

Collections: Collections allow us to store both homogeneous and heterogeneous elements inside it.

Collections support dynamic memory allocation. Hence the size may grow/shrink at runtime.

Using collections, memory can be utilized efficiently.

Collections provide the readymade methods to manage the elements inside it.

Apex provides the below 3 Collection classes.

1. List

2. Set

3. Map

List Collection: List is an ordered collection, which preserves the Insertion Order, i.e., it will arrange the elements in the same order, in which they were inserted.

List Collection stores both homogeneous and heterogeneous elements.

List collection stores elements of primitive type, SObject type, collection, apex type, and user defined types.

Each element inside the collection will be recognized using an "Index Position", which always starts from Zero.

List Collection supports dynamic memory allocation. Hence the collection size can grow/ shrink at runtime.

List Collection allows storing duplicate elements also.

List Collection provides a set of "Instance Methods", to manage the elements inside the collection.

Syntax: List<DataType> <objectName> = new List<DataType>();

Ex: List<Integer> customerCodes = new List<Integer>();

----> Holds a collection of integer elements.

List<ID> recordIds = new List<Id>();

----> Holds a Collection of record Id's.

List<Object> lstElements = new List<Object>();

----> Holds a collection of heterogeneous elements.

List<Account> lstAccounts = new List<Account>();

----> Holds a collection of account records.

List<Candidate__C> lstCandidates = new List<Candidate__C>();

----> Holds a collection of candidate records.

List<SObject> lstRecords = new List<SObject>();

----> Holds collection of heterogeneous records.

Note: List Collection supports the "Nested List Collections" upto 5 levels.

List<List<List<List<List<DataType>>>>> lstElements

= new List<List<List<List<List<DataType>>>>>();

METHODS:

Ex: List<String> countryNames = new List<String>();

1. Add(<ElementName>): This method will add a new element to the collection. i.e., we can add only one element to the collection at a time.

Ex: countryNames.Add('India');

countryNames.Add('Australia');

countryNames.Add('Japan');

countryNames.Add('Germany');

countryNames.Add('USA');

```
-----------------------------------------------------
| India | Australia | Japan | Germany | USA |
-----------------------------------------------------
  0       1           2         3         4
```

2. Add(<IndexPosition>, <ElementName>): This method will insert the specified element in the collection at the specified index position.

Ex: countryNames.Add(2, 'Middle East');

```
-------------------------------------------------------------------
| India | Australia | Middle East | Japan | Germany | USA |
-------------------------------------------------------------------
  0       1           2             3       4         5
```

3. AddAll(<ArrayName/CollectionName>): This method will add a collection of elements to the List Collection at a time.

Ex: String[] countries = new String[]{'India','UK','Srilanka'};

countryNames.AddAll(countries);

```
-----------------------------------------------------------------------------------------------
| India |Australia|Middle East| Japan | Germany | USA | India | UK | Srilanka|
-----------------------------------------------------------------------------------------------
  0       1           2           3         4         5     6     7       8
```

4. Integer Size(): This method returns an integer value, which indicates the number of elements exist in the collection.

Ex: System.debug('Collection Size is....: '+ countryNames.Size());

5. Boolean IsEmpty(): This method will return TRUE, if the collection is empty. else it returns FALSE.

Ex: if(countryNames.isEmpty())

System.debug('Collection is Empty');

else

System. Debug('Collection is Not Empty.');

6. Get (<IndexPosition>): It returns the Element value based on the specified Index Position.

Ex: System.debug('Element exist at the position 4 is....: '+ countryNames.Get(4));

---> o/p: Germany

7. Integer IndexOf(<ElementName>): This method returns the "Index Position" of the specified element name.

Note: 1. If the element is found, it returns index of the element.

2. If the element is not found in the collection, then it returns a negative Value (i.e. -1).

3. If the element is a duplicate element, then it returns the first occurrence of the Element Index Position.

Ex: System.debug('Index Position of Japan is...: '+ countryNames.IndexOf('Japan'));

8. Boolean Contains(<ElementName>): This method is used to search for the specified element in the collection. It returns TRUE, if the element is found, else it returns FALSE.

Ex: if(countryNames.Contains('India'))

System.debug('Element Found in the Collection.');

else

System.debug('Element Not Found in the Collection.');

9. Remove(<IndexPosition>): It will remove the element from the collection based on the specified index position.

Ex: countryNames.Remove(3);

| India |Australia| Middle East| Germany | USA | India | UK | Srilanka|

---

0 1 2 3 4 5 6 7

10. Set (<IndexPosition>, <ElementValue>): This is used to replace the element value with the new value, based on the specified index position.

Ex: countryNames.Set(4, 'United States');

| India |Australia| Middle East| Germany | United State| India | UK | Srilanka|

---

0 1 2 3 4 5 6 7

11. Sort(): It will arrange the elements in ascending order by default.

Ex: countryNames.Sort();

12. Clear(): It will remove all the elements from the collection.

Ex: countryNames.Clear();

13. Clone(): It creates a Duplicate Copy of the Collection.

Ex: List<String> backupCopy = countryNames.Clone();

14. Boolean Equals(<ListCollectionName>): It returns TRUE, if both the List collections are holding the same elements, else it returns FALSE.

Ex: if (countryNames.Equals(<ListCollectionName>))

System.debug('Both the Collections are Equals.');

else

System.debug('Collections are Different.');

Example:

```
// Defining the List Collection
        List<String> lstElements = new List<String>();
// Print the Collection Size
        System.debug('Collection Size is....: '+ lstElements.Size());
// Adding the Elements to Collection.
        lstElements.Add('India');
        lstElements.Add('Apex');
        lstElements.Add('Bangalore');
        lstElements.Add('Welcome');
        lstElements.Add('Japan');
        System.debug('After Insert, Collection Size is...: '+ lstElements.Size());
// Print the Collection Elements
        System.debug('Collection Elements are....: '+ lstElements);
// Inserting Elements
        lstElements.Add(2, 'United States');
```

```
        System.debug('After Insert, Elements are....: '+ lstElements);
    // Adding Multiple Elements
        String[] countries = new String[]{'Germany','Middle East','UK','India'};
        lstElements.AddAll(countries);
        System.debug('After Adding All, Elements are....: '+ lstElements);
    // Check for Collection is Empty or not
        if(lstElements.isEmpty())
            System.debug('Collection is Empty.');
        else
            System.debug('Collection is Not Empty.');
    // Search for the Element...
        If(lstElements.Contains('Apex'))
            System.debug('Element found in the Collection.');
        Else
            System.debug('Element is not found in the Collection.');
    // Get the Element Index Position
System.debug('Index Position of the Element Welcome is...: '+ lstElements.IndexOf('Welcome'));
// Get the Element Value
    System.debug('Element at the Index 6 is....: '+ lstElements.Get(6));

    // Replace the Element Value.
        lstElements.Set(2, 'USA');
        System.debug('After Replace, Elements are...: '+ lstElements);
    // Sort the Elements.
        lstElements.Sort();
        System.debug('After Sorting, Elements are...: '+ lstElements);
    // Sort the Elements in Descending order
```

```
        for (Integer index = lstElements.Size() - 1; index >= 0 ; index-- )
        {
            System.debug('Element is....: '+ lstElements.Get(index));
        }
    // Removing an Element
        lstElements.Remove(6);
        System.debug('After Removing, Elements are...: '+ lstElements);

    // Create Duplicate Copy.
        List<String> backupCopy = lstElements.clone();
        System.debug('Backup copy elements are...: '+ backupCopy);
    // Remove All the Elements
        lstElements.Clear();
        System.debug('After removing all the elements, Collection Size is....: '+ lstElements.Size());
```

SET COLLECTION:

Set collection allows us to store both homogeneous and heterogeneous elements.

By using set collection, we can store the elements of Primitive type, SObject, Collection, Apex type, and User Defined types.

It is an Un-Ordered Collection, which doesn't preserve the insertion Order. i.e., the elements will not be available in the same order, in which they were inserted.

Set Collection will arrange the elements in ascending order by default.

Set collection won't support duplicate elements. It maintains the uniqueness of the elements by using "Binary Comparison".

It supports the dynamic memory allocation. Hence the collection size can grow/shrink at runtime.

It provides a set of readymade methods, to manage the elements inside it.

Syntax: Set<DataType> <referenceName> = new Set<DataType>();

Ex: Set<Integer> productCodes = new Set<Integer>();

----> Holds a set of Unique Integer Elements.

Set<String> customerNames = new Set<String>();

----> Holds a Set of Unique String elements.

Set<Account> accountsSet = new Set<Account>();

----> Holds a Set of Unique Account Records.

Set<Candidate__C> candidatesSet = new Set<Candidate__C>();

----> Holds a Set of Unique Candidate Records.

Methods:

Ex: Set<String> productNames = new Set<String>();

1. Add(<ElementName>):

Ex: productNames.Add('Laptop');

productNames.Add('Desktop');

productNames.Add('Mobile');

2. AddAll(<ArrayName / CollectionName>):

Ex: String[] products = new String[]{'Washing Machine','Dish Washer', 'AC'};

productNames.AddAll(products);

3. Integer Size():

Ex: productNames.Size();

4. Boolean IsEmpty():

Ex: if(productNames.isEmpty())

System.debug('Collection is Empty.');

else

System.debug('Collection is Not Empty.');

5. Boolean Contains(<ElementName>):

Ex: if(productNames.Contains('Mobile'))

System.debug('Element Found in Collection.');

```
        else
                System.debug('Element Not Found in Collection.');
```

6. Remove(<ElementName>):

```
        Ex: productNames.Remove('Desktop');
```

7. Clear():

```
Ex: productNames.Clear();
```

8. Clone():

```
Ex: Set<String> backupCopy = productNames.Clone();
```

9. Boolean Equals(<SetCollectionName>):

```
Ex:     if(productNames.Equals(<SetCollectionName>))
                        System.debug('Both the Collections are Identical.');
        else
                        System.debug('Both the Collections are Different.');
```

Use Case: To remove the duplicate elements from List Collection.

```
// List Collection with the Elements
List<String> countryNames = new List<String>{'India','Australia','Japan','United States',
                                                'Apex','Japan','China','China','Welcome',
                                                'Germany','India','Apex','Bangladesh',
                                                'Middle East','UK','Germany'};

// Collection Size
   System.debug('Collection Size is....: '+ countryNames.Size());
// Copy the Elements from List to Set
Set<String> uniqueElementsSet = new Set<String>();
uniqueElementsSet.AddAll(countryNames);
System.debug('After Removing Duplicates, Size is....: '+ uniqueElementsSet.Size());
```

MAP COLLECTION

Map is a Key-Value pair collection, where each element contains a "Key" and "Value". Key should be always unique but value can be either unique or duplicate".

Map collection supports to store both homogeneous and heterogeneous elements.

It supports dynamic memory allocation. Hence the collection size can grow/shrink at runtime.

Key and Value can be either "Primitive/SObject/Collection/Apex type/User Defined type of element.

Map Collection class provides a set of readymade methods, to manage the elements inside it.

Note: Map is an Un-Ordered collection which doesn't preserve the insertion Order. The elements will be arranged in ascending order based on the Key Name.

Syntax: Map<KeyDataType, ValueDataType> <objectName> =
new Map<KeyDataType, ValueDataType>();

Ex: Map<String, String> mapCountryCodes = new Map<String, String>();

----> Holds collection of Country Names and Country Codes

Country Name ----> Key

Country Code ----> Value

Map<ID, Account> mapAccounts = new Map<ID, Account>();

----> Holds collection of Account Records

Account Id ----> Key

Complete Account record----> Value.

Map<ID, Position__C> mapPositions = new Map<ID, Position__C>();

----> Holds a Collection of Position Records.

Position Record Id ----> Key

Complete Record ----> Value.

Map<Account, List<Opportunity>> mapAccounts =

new Map<Account, List<Opportunity>>();

----> Holds a Collection of Accounts and their Related Opportunity records.

Account Record ----> Key.

List of Opportunities ---> Value.

Methods:

Ex: Map<String, String> fruitsMap = new Map<String, String>();

1. Put (<KeyName>, <ValueName>): This method is used to add a new element to the collection.

Ex: fruitsMap.Put('Red','Apple');

fruitsMap.Put('Yellow','Banana');

fruitsMap.Put('Green','Grapes');

fruitsMap.Put('Black','Grapes');

2. PutAll(<MapCollectionName>): This method will add all the elements from the source map collection to target map collection.

Ex: fruitsMap.PutAll(<SourceMapCollectionName>);

3. Integer Size():

Ex: fruitsMap.Size(); ---> O/p: 4

4. Boolean IsEmpty():

Ex: If(fruitsMap.IsEmpty())

System.debug('Collection is Empty');

Else

System.debug('Collection is Not Empty');

5. Boolean ContainsKey(<KeyName>): It returns TRUE, if the specified key name is found in the collection, else it returns FALSE.

Ex: If (fruitsMap.ContainsKey('Yellow'))

System.debug('Key Found in the Collection.');

Else

System.debug('Key Not Found. Invalid Key Name.');

6. Get(<KeyName>): It returns the Value for the specified key name.

Ex: System. Debug ('Value for the Key Black is..: '+ fruitsMap.Get('Black'));

7. Remove(<KeyName>):

Ex: fruitsMap.Remove('Green');

8. Set<DataType> KeySet(): This method will returns all the element "Key-Names" in the form of a "Set Collection".

Ex: Set<String> keysCollection = fruitsMap.KeySet();

9. List<DataType> Values(): This method will returns all the element values in the form of "List Collection".

Ex: List<String> valuesCollection = fruitsMap.Values();

10. Clear():

Ex: fruitsMap.Clear();

11. Clone():

Ex: Map<String,String> backupCopyMap = fruitsMap.Clone();

12. Boolean Equals(<MapCollectionName>):

Ex: If (fruitsMap.Equals(<SourceMapCollectionName>))

System.debug('Both the Collections are Identical.');

Else

System.debug('Both the Collections are Different.');

Example:

```
// Defining Map Collection
    Map<String, String> fruitsMap = new Map<String, String>();
// Adding Elements to Collection
    fruitsMap.Put('Red','Apple');
    fruitsMap.Put('Orange','Orange');
    fruitsMap.Put('Yellow','Banana');
    fruitsMap.Put('Green','Grapes');
    fruitsMap.Put('Black', 'Grapes');
// Print the Collection size
    System.debug('Collection Size is.....: '+ fruitsMap.size());
```

```
// Print the Collection Elements
        System.debug('Collection Elements are....: '+ fruitsMap);
// Adding Duplicate Elements
        fruitsMap.Put('Red','Cherry');
        System.debug('After Adding Duplicates, Elements are...: '+ fruitsMap);
// Check for Collection is Empty or not
        If (fruitsMap.IsEmpty())
      System.debug('Collection is Empty.');
        Else
      System.debug('Collection is Not Empty.');
// Search for an Element
        if(fruitsMap.ContainsKey('Black'))
      System.debug('Element Found in the Collection. Value is....: '+ fruitsMap.Get('Black'));
        else
      System.debug('Element Not Found. Invalid Key Name.');
// Get All the Element Key Names
        Set<String> keysCollection = fruitsMap.KeySet();
        System.debug('Element Key Names are....: '+ keysCollection);
// Get All the Element Values
        List<String> valuesCollection = fruitsMap.Values();
        System.debug('Element Values are....: '+ valuesCollection);
// Remove an Element from Map Collection
        fruitsMap.Remove('Green');
        System.debug('After Removing element, Collection is...: '+ fruitsMap);
// Remove all the elements
        fruitsMap.Clear();
        System.debug('After Removing All, Collection Size is....: '+ fruitsMap.Size());
```