APEX - COLLECTIONS

Collections are the type of variable which can store multiple number of records. For example, List can store multiple number of Account object's records. Let's have a detailed overview of all collection types.

Lists

List can contain any number of records of primitive, collections, sObjects, user defined and built in Apex type. This is one of the most important type of collection and also, it has some system methods which have been tailored specifically to use with List. List index always starts with 0. This is synonymous to the array in Java. A list should be declared with the keyword 'List'.

Example:

Below is the list which contains the List of a primitive data type *string*, that is the list of cities.

```
List<string> ListOfCities = new List<string>();
System.debug('Value Of ListOfCities'+ListOfCities);
```

Declaring the initial values of list is optional, but we can do that. Below is the example for the same.

```
List<string> ListOfStates = new List<string> {'NY', 'LA', 'LV'}();
System.debug(' Value ListOfStates'+ListOfStates);
```

List of Accounts *sObject*:

```
List<account> AccountToDelete = new List<account> ();//This will be null
System.debug(' Value AccountToDelete'+AccountToDelete);
```

We can declare the nested List as well. It can go up to five level. This is called as Multidimensional list.

This is the list of set of integers.

```
List<List<Set<Integer>>> myNestedList = new List<List<Set<Integer>>>();
System.debug('value myNestedList'+myNestedList);
```

List can contain any number of records, but there is a limitation on heap size to prevent the performance issue and monopolizing the resources.

Methods For Lists

There are methods available for Lists which we can utilize while programming to achieve some functionlity like calculating the size of List, adding an element, etc.

Below are some most frequently used methods.

- size
- add
- get
- clear
- set

The following example demonstrates use of all these methods:

```
//Initialize the List
```

```
List<string> ListOfStatesMethod = new List<string>();

//This statement would give null as output in Debug logs
System.debug('Value of List'+ ListOfStatesMethod);

//Add element to the list using add method
ListOfStatesMethod.add('New York');
ListOfStatesMethod.add('Ohio');

//This statement would give New York and Ohio as output in Debug logs
System.debug('Value of List with new States'+ ListOfStatesMethod);

//Get the element at the index 0
String StateAtFirstPosition = ListOfStatesMethod.get(0);

//This statement would give New York as output in Debug log
System.debug('Value of List at First Position'+ StateAtFirstPosition);

//set the element at 1 position
ListOfStatesMethod.set(0, 'LA');

//This statement would give output in Debug log
System.debug('Value of List with element set at First Position'+ ListOfStatesMethod[0]);

//Remove all the elements in List
ListOfStatesMethod.clear();

//This statement would give output in Debug log
System.debug('Value of List'+ ListOfStatesMethod);
```

You could use array notation as well to declare the List, as given below, but this is not general practice in Apex programming:

```
String [] ListOfStates = new List<string>();
```

Sets

Set is collection type which contains multiple number of unordered unique records. A Set cannot have duplicate records. Like Lists, Sets can be nested.

Example:

We will be defining the set of products which company is selling.

```
Set<string> ProductSet = new Set<string>{'Phenol', 'Benzene', 'H2SO4'};
System.debug('Value of ProductSet'+ProductSet);
```

Methods for Sets

Set does support methods which we can utilize while programming as shown below *weareextendingtheaboveexample*:

```
//Adds an element to the set
//Define set if not defined previously
Set<string> ProductSet = new Set<string>{'Phenol', 'Benzene', 'H2SO4'};
ProductSet.add('HCL');
System.debug('Set with New Value '+ProductSet);

//Removes an element from set
ProductSet.remove('HCL');
System.debug('Set with removed value  '+ProductSet);

//Check whether set contains the particular element or not and returns true or false
ProductSet.contains('HCL');
System.debug('Value of Set with all values '+ProductSet);
```

Maps

It is a key value pair which contains the unique key for each value. Both key and value can be of any data type.

Example:

The following example represents the map of Product Name with Product code.

```
//Initialize the Map
Map<string, string> ProductCodeToProductName = new Map<string, string> {'1000'=>'HCL',
'1001'=>'H2SO4'};

//This statement would give as output as key value pair in Debug log
System.debug('value of ProductCodeToProductName'+ProductCodeToProductName);
```

Methods for Maps

Below are some examples which demostrates the methods which we could use with Map:

```
// Define a new map
Map<string, string> ProductCodeToProductName = new Map<string, string>();

// Insert a new key-value pair in the map where '1002' is key and 'Acetone' is value
ProductCodeToProductName.put('1002', 'Acetone');

// Insert a new key-value pair in the map where '1003' is key and 'Ketone' is value
ProductCodeToProductName.put('1003', 'Ketone');

// Assert that the map contains a specified key and respective value
System.assert(ProductCodeToProductName.containsKey('1002'));
System.debug('If output is true then Map contains the key and output is
:'+ProductCodeToProductName.containsKey('1002'));

// Retrieves a value, given a particular key
String value = ProductCodeToProductName.get('1002');
System.debug('Value at the Specified key using get function: '+value);

// Return a set that contains all of the keys in the map
Set SetOfKeys = ProductCodeToProductName.keySet();
System.debug('Value of Set with Keys '+SetOfKeys);
```

Map values may be unordered and hence we should not rely on the order in which the values are stored and try to access the map always using keys. Map value can be null. Map keys when declared String are case sensitive, for example ABC and abc will be considered as different keys and treated as unique