COLLECTIONS

-> Array is an collection of fixed number of homogeneous data elements

or

-> An array represents a group of elements of same data type.

-> The main advantage of array is we can represent huge number of elements by using single variable. So, the readability of the code is improved.

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

Collection framework: It defines several classes and interfaces to represent a group of Objects as single entity.

Collection is an interface which can be used to represent a group of Objects as a single entity whereas Collections is an utility class present in java.util package to define several utility methods for Collection Objects.

• Collection is an interface

• Collections is a Class

-> All the collection classes are available in “java.util” (utility) package.

-> All the collection interfaces and collection class and together as collection frame work.

-> All the collection classes are classified into three categories

1) List

2) Set

3) Queue

4) Map

List interface

-> It is the child interface of Collection

-> If we want to represent a group of individual objects where duplicates are allowed and insertion order is preserved. Then we should go for List.

-> We can differentiate duplicate Objects and we can maintain insertion order by means of index hence "index plays important role in List"

ArrayList:

-> ArrayList is an implementation class of Collection interface

->The underlying data structure is resizable (Internally it will use Array to store data)

-> Duplicate Objects are allowed

-> Insertion order is preserved

-> Heterogeneous Objects are allowed

LinkedList:

-> LinkedList is one of the implementation classes of Collection interface

-> The underlying data structure is double LinkedList

-> If our frequent operation is insertion or deletion in the middle then LinkedList is the best choice

-> If our frequent operation is retrieval then LinkedList is not best option

-> Duplicate Objects are allowed

-> Insertion order is preserved

-> Heterogeneous Objects are allowed

Stack Class Constructor:

Stack<E> s = new Stack<E>( );

Methods:

-> We can use all collection Methods

-> We can also use legacy methods of Vector class like addElement(), removeElement(), setElementAt(),.....

-> But if we want to follow the LIFO mechanism, we should use Stack methods like follows

1. E push(E obj) : this method will add new element into the Stack

2. E pop() : this method deletes the top element available on Stack

3. E peek() : this method just returns the top element available on Stack

Iterator

-> this cursor is used to access the elements in forward direction only

-> this cursor can be applied Any Collection (List, Set)

-> while accessing the methods we can also delete the elements

-> Iterator is interface and we cannot create an object directly.

-> if we want to create an object for Iterator, we have to use iterator () method

Creation of Iterator:

Iterator it = c.iterator();

here iterator() method internally creates and returns an object of a class which implements Iterator interface.

Methods

1. boolean hasNext()
2. Object next()
3. void remove()

ListIterator

-> This cursor is used to access the elements of Collection in both forward and backward directions

-> This cursor can be applied only for List category Collections

-> While accessing the methods we can also add,set,delete elements

-> ListIterator is interface and we can not create object directly.

-> If we want to create an object for ListIterator we have to use listIterator() method

creation of ListIterator:

ListIterator<E> it = l.listIterator();

here listIterator() method internally creates and returns an object of a class which implements ListIterator interface.

2.Set:

- This category is used to store a group of individual elements. But they elements can't be duplicated.
- Set is an interface whose object cannot be created directly.
- To work with this category, we have to use following implementations class of Set interface

Ex: HashSet,

HashSet

-> HashSet is the implementation class of Set interface which is also used to store group of individual objects but duplicate values are not allowed

-> HashSet internally follows hashtable structure where all the elements are stored using hashing technique which will improve the performance by reducing the waiting time.

Map category

Map interface is not child interface of Collection and hence we cannot apply Collection interface methods.

HashMap

-> HashMap is the implementation class of Map interface which is used to store group of objects in the form of Key-Value pairs where but Keys cannot be duplicated but values can be duplicated

-> HashMap internally follows hashtable data structure

-> HashMap is not a synchronized class

-> HashMap supports only one null value for Key Objects but we can store multiple null values for Value Object

-> HashMap is called unordered Map because it is not guarantee for insertion order of Elements

3. Queue

-> This category is used to hold the elements about to be processed in FIFO(First In First Out) order.

-> It is an ordered list of objects with its use limited to inserting elements at the end of the list and deleting elements from the start of the list, (i.e.), it follows the FIFO or the First-InFirst-Out principle

Ex: PriorityQueue

Java Generics

-> Generics was first introduced in Java 1.5. Now it is one of the most profound features of java programming language.

-> Generic programming enables the programmer to create classes, interfaces and methods in which type of data is specified as a parameter.

-> Generics provide type safety. Type safety means ensuring that an operation is being performed on the right type of data.

Note: Before Generics was introduced, generalized classes, interfaces or methods were created using references of type Object because Object is the super class of all classes in Java, but this way of programming did not ensure type safety.

Ex:-

Class_name <data_type> ref_name = class_name <data_type>();

Generic class

Generic class is a class which can hold any kind of objects, to create Generic class we have to

specify generic type <T> after the class name.

Ex:-

```
class Employee
{
public static void main(String[] args) {

        Employee emp1 = new Employee();


        ArrayList<Employee> al = new ArrayList<Employee> ();
        al.add(emp1);
//      al.add("Hello");


}
}
```