**Java Assignment**

**Topic:- Collections interface and list interface**

**Name:-Taher Fakhruddin Makadam**

**USN:-1MS14MCA50**

**Remember:-**

1. Define collections framework and give a list of top level java.util classes?

Ans.

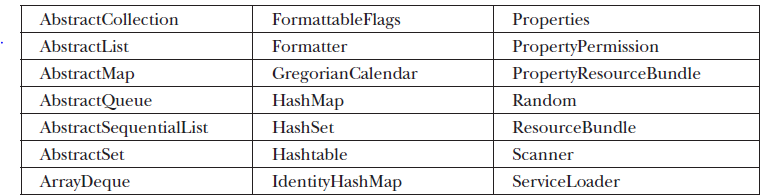
TheCollections Framework is a sophisticated hierarchy of interfaces and classes that provide

state-of-the-art technology for managing groups of objects. It merits close attention by all

programmers.

Because java.util contains a wide array of functionality, it is quite large. Here is a list of

its top-level classes:



**Understand:-**

2. Demonstrate the usage of collections with syntax?

Ans.

The Collection interface is used to pass around collections of objects where maximum generality is desired. For example, by convention all general-purpose collection implementations have a constructor that takes a Collection argument. This constructor, known as a *conversion constructor*, initializes the new collection to contain all of the elements in the specified collection, whatever the given collection's subinterface or implementation type. In other words, it allows you to *convert* the collection's type.

Suppose, for example, that you have a Collection<String> c, which may be a List, a Set, or another kind of Collection. This idiom creates a new ArrayList (an implementation of the List interface), initially containing all the elements in c.

List<String> list = new ArrayList<String>(c);

The Collection interface contains methods that perform basic operations, such as int size(), boolean isEmpty(), boolean contains(Object element), boolean add(E element), boolean remove(Object element), and Iterator<E> iterator().

It also contains methods that operate on entire collections, such as boolean containsAll(Collection<?> c), boolean addAll(Collection<? extends E> c), boolean removeAll(Collection<?> c), boolean retainAll(Collection<?> c), and void clear().

Additional methods for array operations (such as Object[] toArray() and <T> T[] toArray(T[] exist as well.

The Collection interface does about what you'd expect given that a Collection represents a group of objects. It has methods that tell you how many elements are in the collection (size,isEmpty), methods that check whether a given object is in the collection (contains), methods that add and remove an element from the collection (add, remove), and methods that provide an iterator over the collection (iterator).

The add method is defined generally enough so that it makes sense for collections that allow duplicates as well as those that don't. It guarantees that the Collection will contain the specified element after the call completes, and returns true if the Collection changes as a result of the call. Similarly, the remove method is designed to remove a single instance of the specified element from the Collection, assuming that it contains the element to start with, and to return true if the Collection was modified as a result.

**Apply:-**

3. Apply list interface to convert given list of strings to comma seperated values (csv) format.

Ans.

import java.util.ArrayList;

import java.util.List;

public class MyListToCsvString {

public String getListAsCsvString(List<String> list){

StringBuilder sb = new StringBuilder();

for(String str:list){

if(sb.length() != 0){

sb.append(",");

}

sb.append(str);

}

return sb.toString();

}

public static void main(String a[]){

List<String> li1 = new ArrayList<String>(){

{

this.add("animal");

this.add("nuts");

this.add("java");

}

};

MyListToCsvString mtc = new MyListToCsvString();

System.out.println(mtc.getListAsCsvString(li1));

List<String> li2 = new ArrayList<String>(){

{

this.add("java");

this.add("unix");

this.add("c++");

}

};

System.out.println(mtc.getListAsCsvString(li2));

}

}

Output:- animal,nuts,java,java,unix,c++

**Evaluate:-**

4. Programs that allows reading objects from the collection in both the forward and backward directions.

Ans.

import java.util.\*;

public class DemoList {

public static void main(String[] args) {

List ls = new LinkedList();

for(int i=1; i<=5; i++){

ls.add(new StringBuffer("Object " + i));

}

//display how many objects are in the collection

System.out.println("The collection has " + ls.size() + "objects");

//Instantiate a ListIterator

ListIterator li = ls.listIterator();

System.out.println("Forward Reading");

//Forward direction

while(li.hasNext()){

System.out.println(" " + li.next());

}

System.out.println("Backward Reading");

//backword direction

while(li.hasPrevious()){

System.out.println(" " + li.previous());

}

}

}

Output-> The collection has 5objects

Forward Reading

Object 1

Object 2

Object 3

Object 4

Object 5

Backward Reading

Object 5

Object 4

Object 3

Object 2

Object 1

**Analyze:-**

5. Analyze Difference between Generics and Collections with example?

Ans.

->Student Details using Generics:-

List<Student> lstStudents = new List<Student>();

Student objStudent = new Student();

objStudent.Name = "Rajat";

objStudent.RollNo = 1;

lstStudents.Add(objStudent);

objStudent = new Student();

objStudent.Name = "Sam";

objStudent.RollNo = 2;

lstStudents.Add(objStudent);

//Looping through the list of students

foreach (Student currentSt in lstStudents)

{

//no need to type cast since compiler already knows that everything inside

//this list is a Student

Console.WriteLine("Roll # " + currentSt.RollNo + " " + currentSt.Name);

}

-> Student Details using Collections:-

class Student

{

public int RollNo{get; set;}

public string Name{get; set;}

}

//List of students

ArrayList studentList = new ArrayList();

Student objStudent = new Student();

objStudent.Name = "Rajat";

objStudent.RollNo = 1;

studentList.Add(objStudent);

objStudent = new Student();

objStudent.Name = "Sam";

objStudent.RollNo = 2;

studentList.Add(objStudent);

foreach (Object s in studentList)

{

//Type-casting. If s is anything other than a student

Student currentStudent = (Student)s;

Console.WriteLine("Roll # " + currentStudent.RollNo + " " + currentStudent.Name);

}

**Create:-**

6. Write a program that uses Collections.shuffle to generate hands from a normal 52-card deck.

Ans.

import java.util.\*;

public class Deal {

public static void main(String[] args) {

if (args.length < 2) {

System.out.println("Usage: Deal hands cards");

return;

}

int numHands = Integer.parseInt(args[0]);

int cardsPerHand = Integer.parseInt(args[1]);

// Make a normal 52-card deck.

String[] suit = new String[] {

"spades", "hearts",

"diamonds", "clubs"

};

String[] rank = new String[] {

"ace", "2", "3", "4",

"5", "6", "7", "8", "9", "10",

"jack", "queen", "king"

};

List<String> deck = new ArrayList<String>();

for (int i = 0; i < suit.length; i++)

for (int j = 0; j < rank.length; j++)

deck.add(rank[j] + " of " + suit[i]);

// Shuffle the deck.

Collections.shuffle(deck);

if (numHands \* cardsPerHand > deck.size()) {

System.out.println("Not enough cards.");

return;

}

for (int i = 0; i < numHands; i++)

System.out.println(dealHand(deck, cardsPerHand));

}

public static <E> List<E> dealHand(List<E> deck, int n) {

int deckSize = deck.size();

List<E> handView = deck.subList(deckSize - n, deckSize);

List<E> hand = new ArrayList<E>(handView);

handView.clear();

return hand;

}

}