1 Introduction

java.util is the built-in package.

 This package contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization (I18N) and other utility classes.

2 Arrays

 You can manage the collection of elements using arrays as follows:

Creating an Array

int arr[]=new int[3];

Initializing an Array

arr[0]=99;

arr[1]=88;

arr[2]=77;

Accessing an Array elements

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

System.out.println(arr[0]);

 Limitations with array:

Array is a homogeneous data structure i.e only one type of elements can be stored.

Array is static in nature i.e size of an array cannot be modified.

Insert and delete operations require more shifting of elements.

etc.

 Java vendor has given some classes and interfaces before Java 2 to solve the limitiations of an array.

 These classes are called as legacy classes.

 These interfaces are called as legacy interfaces. Enumeration is a legacy cursor that is used to access data from legacy classes one by one

 Following are the 5 legacy classes available released with java 1 in 1995:

1. Stack

2. Vector

3. Dictionary

4. Hashtable

5. Properties

 Enumeration is the legacy interface available.It is a cursor or pointer to the elements in the Collection.

Using Enumeration we can access elements of any of the above legacy classes.

Collection of Data can be managed in two forms:

a) Collection of Elements

Ex :

Collection of Emails

Collection of Phones etc.

 Vector and Stack can be used to manage the collection of elements.There is no clear cut distinction between vector and stack.They can be used interchangeably.

Vector v=new Vector();

v.addElement("suresh");

Enumeration e =v.elements();

while(e.hasMoreElements()) {

System.out.println(e.nextElement());

}

b) Collection of key - value pairs

Ex :

Collection of Country and corresponding Capital.

Collection of ID and corresponding Name. etc.

 Properties, Dictionary and Hastable can be used to manage Collection of key - value

Pairs. There is no clear cut distinction between the three.

Limitations with legacy classes:

o There is no specific scenario defined that in which condition which classes can be used.

o Almost all the methods defined in the legacy classes are synchronized. Because of this,

the same object cannot be accessed by multiple threads concurrently.

Collection Framework

Java Vendor has provided some new API (known as Collection Framework API) from Java 2

to solve the problems associated with Legacy classes.

 Collection Framework provides mechanism to store and manipulate the group of elements.

 Operations like searching, sorting, insertion, deletion etc. can be performed easily on the

group of elements using Collection API.

 Collection Framework API consists of following:

o Interfaces

o Implementation Classes

o Algorithms implementations

 inserting, deleting, searching, sorting and shuffling etc

 Following are the top level interfaces available in Collection Framework API:

o Collection

o Map

o Iterator

 Subclasses of Collection interface are used to manage the collection of elements without key

and value format.

 Subclasses of Map interface are used to manage the collection of elements with key and

value format.

 Iterator is used to access the data from the collection.

Exploring Collection Interface

 Collection interface has following 3 sub interfaces:

List(From Java2)-,can have duplicates,indexed,random access possible,null isallowed

Set(From Java2)-elements are unique,null is allowed(except in treeset)

Queue (From Java5)-not unique,null not allowed

The following methods of Collection interface is common to the whole collection framework,

|  |  |
| --- | --- |
| boolean add(Object obj) | Adds objects to the collection. |
| int size() | Returns no. of objects available in collection. |
| boolean isEmpty() | Checks wheather collection is empty or not. |
| boolean contains(Object obj) | Checks wheather collection has specified object or not. |
| Iterator iterator() | Returns Iterator subclass object. |
| Object[] toArray() | Returns an array containing all of the elements of  collection. |
| boolean remove(Object obj) | Removes first occurance of specefied object from  collection. |
| void clear() | Removes all of the elements from collection. |
| boolean addAll(Collection ) | Adds all the data of specified collection in the current  Collection |
| boolean containsAll(Collection) | Returns true if current collection contains all the elements  of the specified collection. |
| boolean removeAll(Collection ) | Removes all the matching elements of specified collection  from current collection. |
| boolean retainAll(Collection ) | Removes all the non matching elements of specified  collection from current collection. |

[Map Interface](https://www.geeksforgeeks.org/map-interface-java-examples/)is present in java.util package, which provides mainly three methods KeySet(),entrySet() and values(). These methods are used to retrieve the keys of the map, key-value pairs of the map, and values of the map respectively. Since these methods are part of Map Interface, so we can use can these methods with all the classes implementing the map interface like TreeMap, HashMap, and LinkedHashMap.

The following example can be tried with Properties also,

Properties hm = **new** Properties();

hm.put("k1", "Suresh");

hm.put("k2", "Ganesh");

hm.put("k3", "Rama");

hm.put("k4", "Sankar");

Enumeration enm=hm.propertyNames();

-------------------

---

--

[**keySet() Method:**](https://www.geeksforgeeks.org/hashmap-keyset-method-in-java/)

The java.util.HashMap.keySet() method in Java is used to create a set out of the key elements contained in the hash map. It basically returns a set view of the keys or we can create a new set and store the key elements in them.

HashMap<Object, Object> hm = **new** HashMap<Object, Object>();

hm.put("k1", "Suresh");

hm.put("k2", "Ganesh");

hm.put("k3", "Rama");

hm.put("k4", "Sankar");

Set s= hm.keySet();

Iterator i = s.iterator();

**while** (i.hasNext()) {

Object key = i.next();

Object value = hm.get(key);

System.***out***.println(value);

}

HashMap<Object, Object> hm = **new** HashMap<Object, Object>();

hm.put("k1", "Suresh");

hm.put("k2", "Ganesh");

hm.put("k3", "Rama");

hm.put("k4", "Sankar");

Iterator i = hm.keySet().iterator();

**while** (i.hasNext()) {

Object key = i.next();

Object value = hm.get(key);

System.***out***.println(value);

}

**for** (Map.Entry<Object, Object> entry : hm.entrySet()) {

String key = (String) entry.getKey();

String value = (String) entry.getValue();

System.***out***.println(key + "=" + value);

}

// iterating the key-value pairs using iterator

Iterator<Map.Entry<Object, Object>> itr = hm.entrySet().iterator();

**while** (itr.hasNext()) {

System.***out***.println(itr.next());

}

// iterating the key-value pairs using Stream.of()

// method

Stream.*of*(hm.entrySet().toArray()).forEach(System.***out***::println);

| keySet() | entrySet() |
| --- | --- |
| This method returns the Set view of all the keys present in the map, ie it returns a set of keys. | This method returns the Set view of all the mappings present in the map, ie it returns a set of key, value pairs. |
| If any changes happen to the map, then they can be observed in the set also,as set is backed up by the map. | For entrySet() method also, If any changes happen to the map, then they can be observed in the set also,as set is backed up by the map. |
| If iterating through all the pairs of maps using keySet(), then the performance of keySet() is poorer as compared to entrySet(), as for each key, we have to access its corresponding value by using get() function. | When Iterating through all the pairs of the map using entrySet(), then the performance of entrySet() is much better as compared to keySet(). |

In the HashTable(and HashMap),the keys are stored in order of the hashcode of the key.

Eg: **class** VectorDemo1 {

**public** **static** **void** main(String[] args) {

Vector v=**new** Vector();

v.addElement(12);

v.addElement("abc");

Iterator<Object> it=v.iterator();

**while**(it.hasNext()) {

System.***out***.println(it.next());

}

Enumeration en=v.elements();

**while**(en.hasMoreElements()) {

System.***out***.println(en.nextElement());

}

}

}

**class** StackDemo2 {

**public** **static** **void** main(String[] args) {

Stack v=**new** Stack();

v.addElement(12);

v.addElement("abc");

Iterator<Object> it=v.iterator();

**while**(it.hasNext()) {

System.***out***.println(it.next());

}

Enumeration en=v.elements();

**while**(en.hasMoreElements()) {

System.***out***.println(en.nextElement());

}

}

}

**class** HashTableDemo1 {

**public** **static** **void** main(String[] args) {

HashMap<Object, Object> hm = **new** HashMap<Object, Object>();

hm.put("k4", "Suresh");

hm.put("k2", "Ganesh");

hm.put("k3", "Rama");

hm.put("k1", "Sankar");

Iterator i = hm.keySet().iterator();

**while** (i.hasNext()) {

Object key = i.next();

Object value = hm.get(key);

System.***out***.println(value);

}

**for** (Map.Entry<Object, Object> entry : hm.entrySet()) {

String key = (String) entry.getKey();

String value = (String) entry.getValue();

System.***out***.println(key + "=" + value);

}

// iterating the key-value pairs using iterator

Iterator<Map.Entry<Object, Object>> itr = hm.entrySet().iterator();

**while** (itr.hasNext()) {

System.***out***.println(itr.next());

}

// iterating the key-value pairs using Stream.of()

// method

Stream.*of*(hm.entrySet().toArray()).forEach(System.***out***::println);

}

}

**class** PropertiesDemo1 {

**public** **static** **void** main(String[] args) {

Properties hm = **new** Properties();

hm.put("k1", "Suresh");

hm.put("k2", "Ganesh");

hm.put("k3", "Rama");

hm.put("k4", "Sankar");

Enumeration enm=hm.propertyNames();

Iterator i = hm.keySet().iterator();

**while** (i.hasNext()) {

Object key = i.next();

Object value = hm.get(key);

System.***out***.println(value);

}

**for** (Map.Entry<Object, Object> entry : hm.entrySet()) {

String key = (String) entry.getKey();

String value = (String) entry.getValue();

System.***out***.println(key + "=" + value);

}

// iterating the key-value pairs using iterator

Iterator<Map.Entry<Object, Object>> itr = hm.entrySet().iterator();

**while** (itr.hasNext()) {

System.***out***.println(itr.next());

}

// iterating the key-value pairs using Stream.of()

// method

Stream.*of*(hm.entrySet().toArray()).forEach(System.***out***::println);

}

}

**class** ArrayListDemo1 {

**public** **static** **void** main(String[] args) {

Collection c=**new** ArrayList();

// List l=new ArrayList();

// ArrayList al=new ArrayList();

System.***out***.println(c.isEmpty());

System.***out***.println("size() tells u how many element are stored,size="+c.size());

System.***out***.println(c.contains("Suresh"));

System.***out***.println(c.remove("Suresh"));

c.add("Suresh");

c.add("Ganesh");

c.add("Rama");

c.add("Sankar");

c.add("Uma");

c.add("List will maintain insertion order");

System.***out***.println("size() tells u how many element are stored,size="+c.size());

System.***out***.println(c.isEmpty());

System.***out***.println(c.contains("Suresh"));

System.***out***.println(c.remove("Suresh"));

System.***out***.println(c);

System.***out***.println("clear() removes all elemenst from the Collection");

c.clear();

System.***out***.println(c);

}

}

**class** ArrayListDemo2 {

**public** **static** **void** main(String[] args) {

Collection c=**new** ArrayList();

// List l=new ArrayList();

// ArrayList al=new ArrayList();

c.add("Suresh");

c.add("Ganesh");

c.add("Rama");

c.add("Sankar");

c.add("Uma");

c.add("Lot of old applications using arrays still.To convert collection to array we use toArray()");

Object[]o=c.toArray();

**for** (Object object : o) {

System.***out***.println(object);

}

Iterator it=c.iterator();

System.***out***.println("Iterator");

**while**(it.hasNext()) {

System.***out***.println(it.next());

}

**for** (Object object : c) {

System.***out***.println(object);

}

//System.out.println(o);

}

}

**class** ArrayListDemo3 {

**public** **static** **void** main(String[] args) {

Collection courseList=**new** ArrayList();

// List l=new ArrayList();

// ArrayList al=new ArrayList();

courseList.add("Java");

courseList.add("Angular");

courseList.add("Reactjs");

courseList.add("Devops");

courseList.add("Spring");

courseList.add("Micro Services");

System.***out***.println(courseList);

Collection numList=**new** ArrayList();

numList.add(11);

numList.add(12);

numList.add(13);

numList.add(14);

numList.add(15);

System.***out***.println(numList);

courseList.addAll(numList);

System.***out***.println("adding");

System.***out***.println(courseList);

**boolean** b=courseList.containsAll(numList);

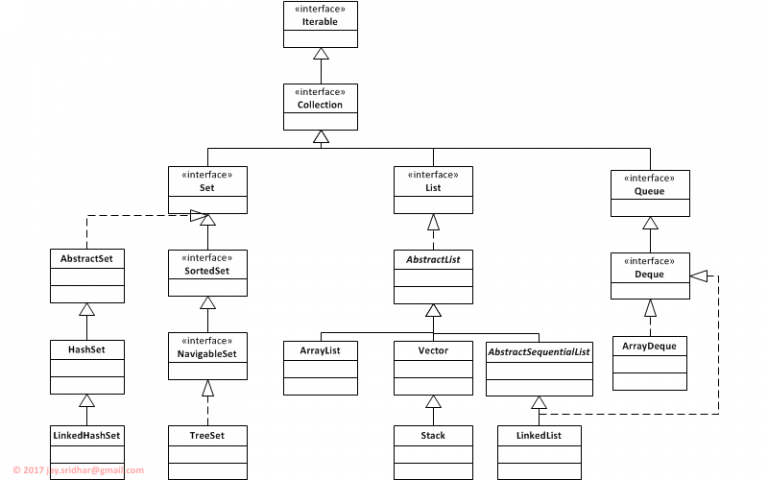
System.***out***.println(b);

System.***out***.println(courseList.retainAll(numList));

System.***out***.println(courseList);

}

}



|  |  |  |  |
| --- | --- | --- | --- |
| Interfaces | Abstract Classes | Concrete Sub classes |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |