Disadvantages of array

The Collection Interface

Collection API framework are set of classes that implement various data structures like lists, sets, maps and queues.

1.We can add objects to the collection.

2. We can remove objects from the collection.

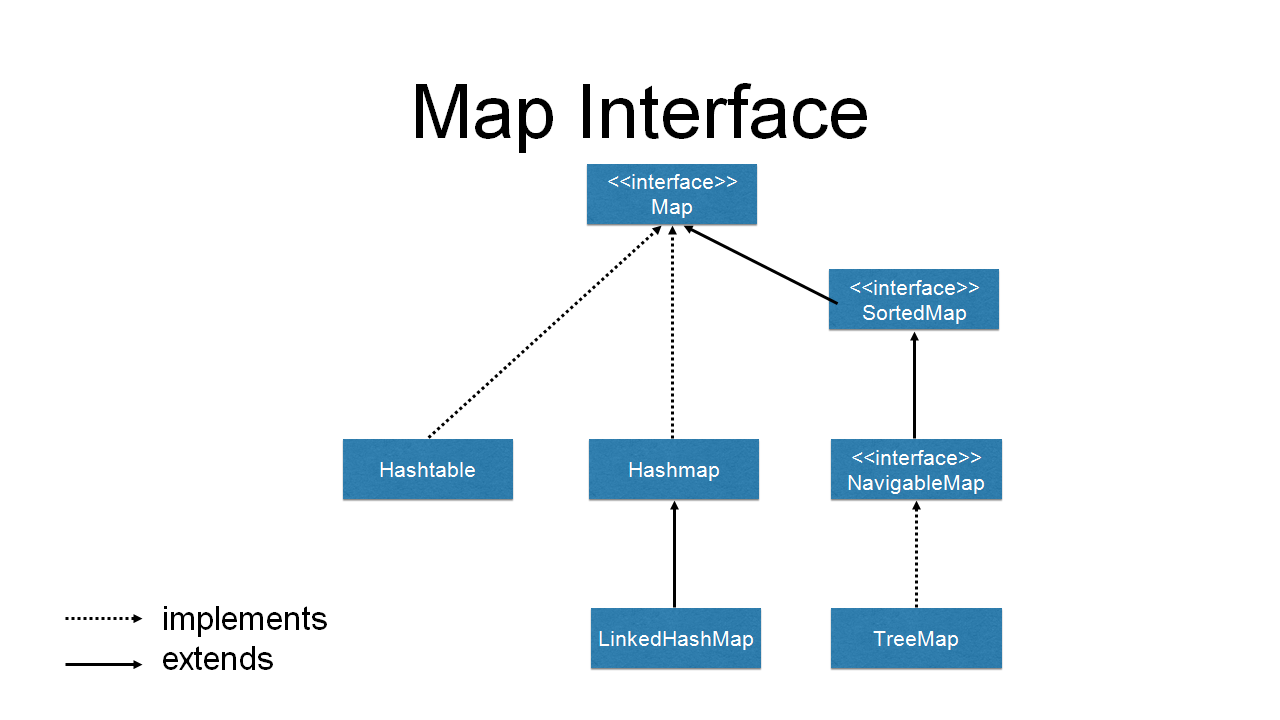
3. Retrieve objects from the collections.

4. Iterate through the collection.

**Note**:

In general collection is an object that contains other object and object present in the collection are called element.

https://www.javatpoint.com/images/collection-hierarchy.png



**Note**:

1. All the Collections are under **util** packages.

2. All the collection classes has to take Object as argument

The List Interface

Lists are used for Storing sequence of elements and it allows duplicates and maintains same order.

**ArrayLists:**

ArrayLists is an ordered (indexed) collection, not sorted and auto-grow able array. It’s internally implemented as resizable array.

This is one of most widely used concrete class. It’s fast to search, but slow to insert or delete and it allows duplicate.

When a element is deleted, the index of remaining elements should be rearranged which is time consuming.thats why we go with linked list where just address of next element shall be updated.

NOTE: Arraylist allows duplicate values

Some of the important methods of arraylist

ADD – add without index, add with index and value, addall

DELETE- remove based on index, remove based on value, remove, clear

Update- add with index and value, set with index and value

Search –contains value, equals

**Collections.copy**- to copy one arraylist elements to other arraylist only if destination arraylist has index more than source arraylist.

**Collections.sort:** To sort arraylist elements

**Collections.shuffle:** To shuffle. arraylist elements

/\*\*

\* array length - array.length

\* String length - string.length()

\* collection length - collection.size()

\*

\* Object - It is the super most class for all the classes

\* If any class is not extending any other class, that class extends Object class by default

\*

\*

\* List(Interface)

\* ArrayList

\* LinkedList

\*

\*/

**public** **class** Run1 {

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

//int arr[] = new int[5];

//arr[6] = 10;

//int arr[] = {10,20.15f,30,40,true};

//ArrayList<Integer> a1 = new ArrayList();//This wil allow only Integer values

LinkedList a1 = **new** LinkedList();//Object

a1.add(10);//0

a1.add(**null**);//1

a1.add(30);//2

a1.add(**null**);//3

a1.add(50);//4

//a1.addAll

//a1.con

//System.out.println(a1);

//String s1 = (String) a1.get(2);

//int size = a1.size();

//System.out.println(size);

/\*for(int i = 0; i < size; i++) {

System.out.println(a1.get(i));

}\*/

/\*for(Object a : a1) {

System.out.println(a);

}\*/

//a1.add(1, 60);

**for**(Object a : a1) {

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

}

/\*a1.remove("Hi");

for(Object a : a1) {

System.out.println(a);

}

System.out.println(a1.isEmpty());\*/

}

}

**LinkedList:**

Linked List is a combination of both list and queue

A Linked list is ordered by index position, like Arraylist, except that the elements are doubly linked to one another , Inserting and deleting is very fast in Linked list. But, accessing elements is slow as we have to traverse

When you want to add or remove elements frequently, then you can use Linked list

INTERVIEW QUESTIONS

Q. What is the difference between Arraylist and Linked list?

Difference between Arraylist and Linked list

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| 1. Arraylist internally uses dynamic array to store the elements | 1.Linkedlist internally uses doubly linked list to store the elements |
| 2. Manipulation with Arraylist is slow because it internally uses array. If any elements is removed from the array, all bits are shifted in memory | 2. Manipulation with Linkedlist is faster than array list because it internally uses doubly linked list, so bit shift is not required in memory |
| 3. Arraylist class can act only as a list ,because it implements list only | 3.Linked list class acts as list and queue both because it implements list and DeQueue interfaces |
| 4. Arraylist is better for storing and accessing the elements | 4. LinkedList is better for manipulating data. |
| 5.arraylist has only one parent List | 5. LinkedList has features from list and DQUEUE |

**Priority Queue:**

Priority Queue class Extends Abstractqueue which implements Queue interface but it does not order the elements in fifo(first in first out) manner.

**Internally autosort**

Random display

The **peek**() method retrieves the value of the first element of the queue without removing it from the queue. If the queue is empty the peek() method returns **null**.

The **element**() method behaves like peek(), so it again retrieves the value of the first element without removing it. Unlike peek ), however, if the list is empty element() throws a **NoSuchElementException**.

The **poll**() method retrieves the value of the first element of the queue by removing it from the queue. If the list is already empty it returns **null** but does not throw any exception.

The **remove**() method behaves as the poll() method, so it removes the first element of the list and if the list is empty it throws a **NoSuchElementException**.

**public** **class** Run4 {

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

LinkedList que = **new** LinkedList();

//que.add("Cherry");

//que.add("Apple");

//que.add("Banana");

//System.out.println(que);

/\*for(Object o:que ) {

System.out.println(o);

}\*/

/\*Object elementVar = que.element();

System.out.println("First Element : " + elementVar);

System.out.println("After element:"+que);\*/

Object peekVar = que.peek();

System.***out***.println("First Element : " + peekVar);

System.***out***.println("After Peek:"+que);

/\*Object rem = que.remove();

System.out.println("Remove Element : " + rem);

System.out.println("After remove:"+que);\*/

//que.poll();

//System.out.println("After poll:"+que);

}

}

The Set Interface

A Set is a Collection that cannot contain **duplicate elements.**

\* It is used to store the unique values. Does not allow the duplicates

\* HashSet

\* All the objects are stored in random order

\*

\* LinkedHashSet

\* Objects are stored in same order as insertion

\*

\* TreeSet

\* Objects will be stored into sorted order(ascending)

\* Same type of data should be present or we will get ClassCastException

**Hashset**:

It uses the Hashcode of an Object being inserted. This class can be used when we want collection **with no duplicates and order is not important** when we iterate.

Since duplicate value is not allowed in set, get method is not available for set, when we try to insert duplicate value, set does store already available data in the set, that’s why get method with index is not available.

**public** **class** Run2 {

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

TreeSet ts = **new** TreeSet();

ts.add("abc");

ts.add("xyz");

//HashSet hs = new HashSet();

//LinkedHashSet hs = new LinkedHashSet();

TreeSet hs = **new** TreeSet();

//Adding the elements/objects/data

hs.add("a");

hs.add("bc");

hs.add("bab");

hs.addAll(ts);

//hs.add(20);

//hs.add("Hi");

//hs.add(100);

//hs.add(true);

**for**(Object h : hs) {

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

}

//hs.remove(10);

/\*System.out.println("----------");

for(Object h : hs) {

if(h.equals(80)) {

System.out.println("Data is present");

}

System.out.println(h);

}\*/

System.***out***.println(hs.contains("bc"));

}

}

**SortedSet**

The SortedSet interface extends Set and declares the behaviour of a set sorted in an ascending order.TreeSet is the class which displays set content in sorted manner.**treeset will accept same type of data because it auto sorts data.**

Several methods throw a **NoSuchElementException** when no items are contained in the invoking set. A **ClassCastException** is thrown when an object is incompatible with the elements in a set.

import java.util.\*;

public class SortedSetTest {

public static void main(String[] args) {

// Create the sorted set

SortedSet set = new TreeSet();

// Add elements to the set

set.add("b");

set.add("c");

set.add("a");

// Iterating over the elements in the set

Iterator it = set.iterator();

while (it.hasNext()) {

// Get element

Object element = it.next();

System.out.println(element.toString());

}

}

}

Output

a

b

c

The Map Interface

The Map interface maps unique keys to values. A key is an object that you use to retrieve a value. Value can be duplicate but not keys.

Given a key and a value, you can store the value in a Map object. After the value is stored, you can retrieve it by using its key.

Since key is uniques, in foreach loop , the keys value is stored in a set.

**Example**

**public** **class** Run3 {

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

//HashMap<Integer, Character> hm = new HashMap();

TreeMap hm = **new** TreeMap();

hm.put(3, 'a');

hm.put(2, 'b');

hm.put(1, 'c');

hm.put(4, 'd');

//System.out.println(hm.get(1));

Set s = hm.keySet();

//s = 1,2,3

//s1 = 3

**for**(Object s1 : s) {

System.***out***.println(hm.get(s1));

}

hm.remove(3);

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

**for**(Object s1 : s) {

System.***out***.println(hm.get(s1));

}

}

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | LIST | SET | MAP | QUEUE |
| INDEX | The List is an indexed sequence | The Set is an non-indexed sequence | Key Based Value |  |
| DUPLICATE VALUES | List allows duplicate elements | Set doesn't allow duplicate elements | Duplicate value is allowed , duplicate keys are not allowed |  |
| ACCESS BY INDEX | Elements by their position can be accessed | Position access to elements is not allowed. | Access using keys |  |
| INSERTION ORDER | The insertion order is maintained by the List. | It doesn't maintain the insertion order of elements. |  |  |
| NULL VALUE | List allows us to add any number of null values. | Set allows us to add at least one null value in it. |  |  |
| WHEN TO USE | It is used when we want to frequently access the elements by using the index. | It is used when we want to design a collection of distinct elements. |  |  |
| ITERATOR | The method of List interface listiterator() is used to iterate the List elements. | The iterator is used when we need to iterate the Set elements. |  |  |

Reading and Writing Files

**Write File-**

Step 1: Use an output stream which selects the directory where file will be save.

Step 2: Use a same output stream to create a file and chain it to previous step.

Step 3: BufferedWriter class to write characters in file and chain it to previous step.

Step 4: Close connection (last stream) imp: because close the resources to release.

Example:

package readingAndWriting;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

public class WriteAFile {

public static void main(String[] args) throws IOException {

//stream connectivity

File f = new File("D:\\FileHandlingBatch56\\text1.html");

FileWriter fw = new FileWriter(f);

BufferedWriter writer = new BufferedWriter(fw);

//writing code

writer.write("Second Line, Mid");

writer.newLine();

writer.write("Third Line, End");

writer.newLine();\*/

//close stream

writer.close();

System.out.println("File is created....");

}

}

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

**Read File-**

Step 1: Use an input stream which selects the directory in which our file is saved.

Step 2: Use a same input stream to use a file and chain it to previous step.

Step 3: Use BufferedReader class to read characters in file and chain it to previous step.

Step 4: Close connection (last stream) imp: because close the resources to release.

Example:

package readingAndWriting;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

public class ReadAFile {

public static void main(String[] args) throws IOException {

//stream connectivity

File f = new File("D:\\FileHandlingBatch56\\text1.txt");

FileReader fr = new FileReader(f);

BufferedReader reader = new BufferedReader(fr);

//Read a file

String line = null;

while((line=reader.readLine())!=null){

System.out.println(line);

}

//close

reader.close();

}

}

**Java Interview Questions**

* **Features of Java?[Fresher]**
* **What do you mean by Constructor?**
* **What is Local Variable and Instance Variable?**
* **What is Class and Object?**
* **What is inheritance with example?**
* **What is encapsulation with example?**
* **What is polymorphism with example?**
* **Difference between Method Overloading and Over-ridding?**
* **What is an Interface? Why we need**
* **Difference between Arrays and Arraylist/Collection?**
* **What are the classes and interfaces are available in collections?**
* **What is Map, List and Set? Usage**
* **Difference between ArrayList and LinkedList?**
* **What is an Exception? Types**
* **Different ways to handle exceptions?**
* **What is final and finally and finalize?**
* **What is the use of super and this keyword?**
* **What us the difference between static member and non static member?**
* **What is string class? List methods from string class**
* **What is constant pool and non-constant pool?**
* **What is Type casting is java? Different way/types**
* **Does java support multiple inheritance?**
* **Does java is 100% object-oriented programming language?**
* **Difference between abstract class and interface?**
* **Difference between == and equals() method?**
* **What is throw, throws and throwable?**
* **What is this keyword and super keyword?**
* **What is the use of final keyword?**
* **What is the difference between | and || and & and &&?**
* **What does variables mean?**
* **What are the members of the class we have it in java?**
* **What ae the control statements we have it in java?**