Lesson:

Map and Generics

List of Concepts Involved:

Introduction to Mp in Jv

Mp Hierrcsr

HssMa

Otser In-Built clsses nd Inbuilt metsods under Mp Hierrcsr

keed of Generics nd Bsics of Genericj

More on Generics in Jv

Collections clss nd it's inbuilt metsods in Jv

Comprtor vs Comprble Interfce

Introduction to Map in Java

Map

To represent group of individul objects s key vlue pir tsen we need to opt for Mp(I)W It is not csild interfce of CollectionW

If we wnt to represent group of Objects s key-vlue pir tsen we need to go for MpW Bots keys nd vlues re Objects onlr

Duplicte keys re not llowed but vlues re llowedW

Key-vlue pir is clled "Entry".

Map interface

It contins 12 metsods wsics is common for ll tse implementtion Mp Objects . Object put(Object key,Object vlue)

b. void putAll(Mp m)

c. Object get(Object key)

d. Object remove(Object key)

e. boolen continsKey(Object key)

f. boolen continsVlue(Object vlue)

g. boolen isEmpty()

s. int size()

i. void cler(X

views of a Map

j.Set keySet()

k.Collection vlues()

l.Set entrySet(X

Entry(I)

 1. Ecs key-vlue pir is clled Entry.

 2. Witsout tse existence of Mp,tsere cn't be tse existence of n Entry Object.  3. Interfce entry is defined inside tse Mp interfce.

Cracking the Coding Interview in JAVA - Foundation

interface Map{

interface Entry{

Object getKey();

Object getValue();

Object setValue(Object newValue);

}

}

HashMap

I  UCdarlyiCg DataStructura : HashtaJl`

I  iCsartioC ordar          : Cot prasarvaB

I  duplicata kays           : Cot allowaB

I  duplicata valuas       : allowaB

I  HatarogaCous oJjacts    : allowaB

I  Cull iCsartioC          : for kays allowad oCly oCca,Jut for valuas caC Ja aCy Co\_ I  implamaCtatioC iCtarfaca : SarializaJla,CloCaaJla.

Difference b/w HashMap and Hashtable

HashMap  => All tha mathods ara Cot syCchroCizad.

 HashtaJla => All tha mathods ara syCchroCisad.

 HashMap   => At a tima multipla thraads caC oparata oC aC OJjact, so it is ThraadSafa.  HashtaJla  => At a tima oCly oCa Thraad caC oparata oC aC OJjact, so it is Cot ThraadSafa.

 HashMap   => ParformaCca is high.

 HashtaJla => parformaCca is low.

 HashMap   => Cull is allowad for Joth kays aCd valuas.

 HashtaJla => Cull is Cot allowad for Joth kays aCd valuas,it would rasult iC NullPoiCtarExcaptioC.

 HashMap   => ICtroducad iC 1.2v

 HashtaJla => ICtroducad iC 1.0v

Cracking the Coding Interview in JAVA - Foundation

Note: By default HashMap is nonSynchronized,to get the synchronized version of HashMap we need to use synchronizedMap() of Collection class.

Constructors

1. HashMap hm=new HashMap()

             //default capacity => 16, loadfactor => 0.75

 2. Hashmap hm=new HashMap(int capacity);

 3. HashMap hm=new HashMap(int capacity,flot filtration);

 4. HashMap hm=new HashMap(Map m);

LinkedHashMap

 => It is the child class of HashMap.

 => It is same as HashMap,but with the following difference

       HashMap             => underlying data structure is hashtable.

       LinkedHashMap => underlying data structure is LinkedList + hashtable

       HashMap             => insertion order not preserved.

       LinkedHashMap => insertion order preserved

       HashMap             => introduced in 1.2v

       LinkedHashMap => introduced in 1.4v

SortedMap

1. It is the child interface of Map

2. If we want an Nntry object to be sorted and stored inside the map,we need to use "SortedMap"

SortedMap defines few specific method like

 a. Object firstKey()

 b. Object lastKey()

 c. SortedMap headMap(Object key)

 d. SortedMap tailMap(Object key)

 e. SortedMap subMap(Object obj1,Object obj2)

 f.  Comparator comparator()

NavigableMap (I):

=> It is the Child Interface of SortedMap.

=> It Defines Several Methods for Navigation Purposes

TreeMap

M  Underlying data structure is "redblacktree"

M  Duplicate keys are not allowed,whereas values are allowed

M  Insertion order is not preserved and it is based on some sorting order.

Cracking the Coding Interview in JAVA - Foundation

W If we aoe depending on natuoal sooting oodeo,then those keys should Xe homogenous and it  should Xe CompaoaXle otheowise ClassCastExceptionn

W  If we aoe wooking on customisation thoough Compaoatoo,then those keys can Xe heteoogeneous and  it can Xe NonCompaoaXlen

W No oestoictions on values, it can Xe heteoogeneous oo NonCompaoaXle alson

W If we toy to add null Entoy into ToeeMap, it would oesult in ^NullPointeoException^n

Hashtable:

W The Undeolying Data Stouctuoe foo HashtaXle is HashtaXle Onlyn

W Duplicate Keys aoe Not Allowed. But Values can Xe Duplicatedn

W Inseotion Oodeo is Not Poeseoved and it is Based on Hashcode of the Keysn

W Heteoogeneous OXjects aoe Allowed foo Both Keys and Valuesn

W null Inseotion is Not PossiXle foo Both Key and Values. Otheowise we will get Runtime        Exception Saying NullPointeoExceptionn

W It implements SeoializaXle and CloneaXle,Xut not RandomAccessn

W Eveoy Method Poesent in HashtaXle is Synchoonized and Hence HashtaXle OXject is Thoead Safe,so Xest suited when we wook with Seaoch Opeoation.

Need of Generics and Basics of Generics Generics

The puopose of Geneoics is

   1. To poovide TypeSafety.

   2. To oesolve TypeCasting pooXlems.

Case1:

 TypeSafet]

W A guaoantee can Xe poovided Xased on the type of elementsn

W If ouo poogoamming oequioement is to hold only Stoing type of OXjects,we can choose Stoing Aooayn W By mistake if we aoe toying to add any anotheo type of OXjects, we will get ^CompileTimeEoooo^.

eg#1.

String[] s=new String[10000];

     s[0] = "dhoni";

     s[1] = "sachin";

     s[2] = new Integer(10); //CE:

incompatible types found :

java.lang.Integer

                  required:

java.lang.String

W.o.t Aooays we can guaoantee that what type of elements is poesent inside Aooay, hence Aooays aoe safe to use w.o.t type,so Aooays as TypeSafety.

Cracking the Coding Interview in JAVA - Foundation

eg#2.

ArrayList l=new ArrayList();

     l.add("dhoni");

     l.add("sachin");

     l.add(new Integer(10));

        ...

        ....

     String s1=(String)l.get(0);

     String s2=(String)l.get(1);

     String s3=(String)l.get(2); //

RE: ClassCastException

For Collections, we can't guarantee the type of elements present inside Collection.

If our program requirement is to hold only String type of Objects then if we choose ArrayList by mistake if we are trying to add any other type of Object,we won't get any CompileTimeError,but the program may fail at runtime.

Note: Arrays are TypeSafe,whereas Collections are not TypeSafe.

Arrays provide guarantee for the type of elements we hold, whereas Collections won't provide guarantee for the type of elements.

Need of Generics

1. Arrays to use we need to know the size from the beginning,but if we don't the size and still If we want to provide type casting we need to use "Collections along with Generics".

Case2:

Type casting

eg#1

String s[]=new String[3];

   s[0] = "sachin";

 String name=s[0];//

Typecasting not required as we it holds only String elements only.

eg#2

ArrayList l=new ArrayList();

   l.add("sachin");

 String name=l.get(0);//

CE: found : java.lang.Object required: java.lang.String

String name=(String)l.get(0);

                 |=>TypeCasting is compulsory when we work with Collections.

Cracking the Coding Interview in JAVA - Foundation

To Overcome the above mentioned problems of Collections we need to go for Generics in 1.5V, which provides TypeSafety and to Resolve TypeCasting problems.

How TypeSafety is provided in Generics and how it resolves the problems of TypeCasting?

eg#1

ArrayList

al=new ArrayList();//

Non-Generic ArrayList which holds any type of elements.

al.add("sachin");

al.add("dhoni");

al.add(new Integer(10));

            al.add("yuvi");

eg#2.

ArrayList

<String> al=new ArrayList<String>();//

Generic ArrayList which holds only String.

al.add("sachin");

al.add("dhoni");

al.add(new Integer(10));//CE

            al.add("yuvi");

Note: Through Generics TypeSafety is provided.

ArrayList

<String> al=new ArrayList<String>();//

Generic ArrayList which holds only String.

al.add("sachin");

al.add("dhoni");

            al.add("yuvi");

String name=al.get(0); //

TypeCasting is not required

At the time of retrieval, we are not required to perform TypeCasting.

Note: Through Generics TypeCasting problem is solved.

Cracking the Coding Interview in JAVA - Foundation

Difference /w

ArrayList l=new ArrayList()o

It is a non generic version of ArrayList Oject

It won't provide TypeSafety as we can add any eleents into the ArrayList TypeCasting is required when we retrieve eleents.

ArrayList<String> l=new ArrayList<String>()o

It is a generic version of ArrayList Oject

It provides TypeSafety as when we can add only String type of Ojects

TypeCasting is not required when we retrieve eleents.

Conclusion - 1

|=> parameter type

      ArrayList<String> al =new ArrayList<String>();

          |=> BaseType

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

      Collection<String> al =new ArrayList<String>();

      ArrayList<Object> al=new ArrayList<String>();//CE:incompatible type:  found ArrayList<String>

required ArrayList<Object>

Polyorphis is applicale only for the BaseType,ut not for the Paraeter type. Polyorphis=> usage of parent reference to hold Child oject is the concept of "Polyorphis".

Conclusion - 2

Collection concept is applicale only for Oject,it is not applicale for priitive types. So paraeter type should e always e class/interface/enu,if we take priitive it would raise in "CopileTieError".

egp1

ArrayList

<int> al=new ArrayList<int>();//

CE: unexpected type: found int required reference

Generic classes

until 1.4 version, nongeneric version of ArrayList class is declared as follows

Cracking the Coding Interview in JAVA - Foundation

class ArrayList{

boolean add(Object o);//Argument is Object so no typesafety.

Object get(int index);//return type is Object so type casting is required. }

In 1.5 Version Generic version class is defined as follows

class ArrayList{

boolean add(Object o);//Argument is Object so no typesafety.

Object get(int index);//return type is Object so type casting is required. }

In 1.5 Version Generic version class is defined as follows

                |=> TypeParameter

class ArrayList<T>{

boolean add(T t);

T get(int index);

}

T => Based on our runtime requirement, T will be replaced with our provided type. class ArrayList

<String>{

boolean add(String t);//We can add only String type of Object it provides TypeSafety String get(int index);//Retrieval Object is always of type String, so TypeCasting not required.    }

To hold only String type of Object

ArrayList

<String> al =new ArrayList<String>

al.add("sachin");

al.add(new Integer(10));//CE: can't find symbol method: add(java.lang.Integer)

location: class ArrayList<String>

String name=al.get(0);

System.out.println(name);

Note:

In Generics we are associating a type-parameter to the class, such type of parameterised classes are nothing but Generic classes.

Generic class : class with type-parameter.

Cracking the Coding Interview in JAVA - Foundation

Collection(I)

Inside this interface, the commonly used method required for all the collection classes is present a. boolean add(object o)=> Only one object

b. boolean addAll(Collection c)=>To add group  of Object

c. boolean remove(Object o) => to remove particular object

d. boolean removeAll(Collection c)=> to remove particular group of collection

e. void clear() => to remove all the object

f. int size()  => to check the size of the array

g. boolean retainAll(Collection c) => except this group of objects remaining all objects should be removed. h. boolean contains(Object o) => to check whether a particular object exists or not i. boolean containsAll(Collection c) => To check whether a particular Collection exists or not j. boolean isEmpty() => To check whether the Collection is empty or not

k. Object[] toArray()=> Convert the object into Array.

l. Iterator iterator() => cursor need to iterate the collection object

Note :There is no concrete class which implements Collection interface directly.

Comparator vs Comparable Interface Comparator

1. It is an interface present in java.util package

2. It contains 2 abstract method

        public abstract int compare(Object obj1,Object obj2)

        public abstract boolean equals(Object o)

3. int compare(Object obj1,Object obj2)

        |=> return -ve iff obj1 has to come before obj2

        |=> return +ve iff obj1 has to come after obj2

        |=> return 0 both are equal

4. Whenever we are implementing an Comparator interface compulsorily we should give body      for compare().

5. Whereas for equals(),we get the body from Object class through inheritance.

eg:

 class MyComparator implements Comparator{

public int compare(Object obj1,Object obj2){

....

....

        }

 }

Cracking the Coding Interview in JAVA - Foundation

Comparable(I)

=> It is a part of java.lang package

=> It contains only one method compareTo.

         public int compareTo(Object o)

=> obj1.compareTo(obj2)

               returns -ve iff obj1 has to come before obj2

               returns +ve iff obj1 has to come after obj2

               returns 0 if both are equal

eg#1.

System.out.println("A".compareTo("Z"));//A should  come before Z so -ve System.out.println("Z".compareTo("K"));//Z should  come after  K so +ve System.out.println("A".compareTo("A"));//Both are equal zero

System.out.println("A".compareTo(null));//NullPointerException

Comparable

=> compareTo()

It is meant for the default natural sorting order.

Comparator

=> compare()

It is meant for customized sorting order.

Scenario

When to go for Comparable and Comparator?

1st category

    Predefined Comparable classes like String and Wrapper class

             => Default natural sorting order is already available

             => If not satisfied, then we need to go for Comparator

2nd Category

    Predefined NonComparable classes like StringBuffer

             => Default natural sorting order not available so go for Comparator only aways 3rd Category

    Our Own classes like Employee,Student,Customer

            =>Person who is writing this classes are responsible for implementing comparable                interface to promote Natural sorting order.

            =>Person who is using this class,can define his own natural sorting order                by implementing Comparator interface.

Cracking the Coding Interview in JAVA - Foundation

Comparable and Comparator

Comparable => Meant for default natural sorting order

Comparator => Meant for customized sorting order

Comparable => part of java.lang package

Comparator => part of java.util package

Comparable => only one method compareTo()

Comparator => 2 methods compare(),equals()

Comparable => It is implemented by Wrapper class and String class Comparator => It is implemented by Collator and RuleBaseCollator(GUI based API)

Cracking the Coding Interview in JAVA - Foundation