byte->short->int->long->flloat->double --->implicit type casting

collection framework:

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

Why we need collections?

int x=10;

int x1=20;

-----

----

----

---

int xn=1000000;

quite difficult to store in variables...

too overcome this problem we goto array conecept...

int a[]=new int[];

a[0]=10;

a[1]=20;

---

---

a[n-1]=n value;

but it has some limitations:

1. it is fixed size

2. it hold only homogenious elements

but not hetero genious elements

3. it doesn't support any other methods

to overcome the above limitation we can go for collections...........

1. it is by default growable nature

2. it is stored either homogenious and hetero genious elements

3. it is support a set of ready made methods with standard structure

Example program on Array:

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

package com.Collections;

public class Demo

{

public static void main(String args[])

{

int a[]=new int[5];

a[0]=10;

a[1]=20;

a[2]=30;

a[3]=40;

a[4]=10.3;//error due hetero genious

a[5]=50; //error array index outof bound exception

for(int i=0;i<5;i++)

{

System.out.println(a[i]);

}

}

It is group of object in single entity. it is implemented on basic standard data strcuture

in c++ program concepts which is exist already...........?

collections --(java) ------------>container(c++)

collection framework(java)------------>STL(c++)

===================================================================

Definition collection framework: it is collection of classes and interfaces.

using classes and intefaces to define the objects, it is a set of methods

definition of collection: a group of objects in single entity

Iterator(I):

===================================

All the Collection classes and interface then the root interface is Iterator.

Iterator(I);

contains 1 abstract method and other two are methods, like

a. hasNext( )-->it returns true if the other element is exist in that array

other wise it returns false

b.next( )->it is only to find the cursor point location

Iterator i=object.iterator( );

package com.test;

import java.util.ArrayList;

import java.util.Iterator;

public class Demo

{

public static void main(String[] args) {

ArrayList a=new ArrayList();

a.add(10);

a.add(20);

Iterator i=a.iterator();

while(i.hasNext())//true the iterator has more elements or false

{

System.out.println(i.next());//cursor pointer to the next element

}

}

}

But in Collection framework has mainly

9 key interfaces in collection framework;

=======================================================

1. Collection(I); add(), addAll(),remove(),removeAll( )

2.Set(I)

3.List(I)

4.SoretdSet(I)

5.NavigableSet(I)

6.Queue(I)

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

7.Map(I) ;- it is not a child interface of Collection

8.SoretedMap(I)

9.NavigableMap(I)

Note: The aboove first 6 interfaces represent a group of individual objects

Note: if we represent a group of objects as key pair values then we can go for map interface(rno,name)

List(I):

===========================================================================

ArrayList(C): Insertion order is preserved(In which way to insert the elements that order ------------

only it print),

duplicates values are allowed(repeated Elements are allowed).

Methods are non-synchronized, it means all thread to access single array list ,

thread safty is not available. performance is good than vector.

By default array size is 10, but it is automatically incremented when we insert 11 th element.

it follows the given formula:

Arraylist size= (current capacity)\*3/2+1;

= (10\*3)/2+1

=16 in the same way 25 etc.

package com.Collections;

import java.util.ArrayList;

public class ArrayListEx

{

public static void main(String args[])

{

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

// ((current cap\*3)/2)+1-->16,25

a.add("ajay");

a.add("vijay");

a.add("murali");

a.add(0,"pranay");//here 6 index

a.add("ajay");//duplicates are allowed

//a.add(10); because String is generic in this example

System.out.println(a.size());//by default 10 is size

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

al.addAll(a);

al.add("vinay");

System.out.println(al);

al.removeAll(a); //al.retainall(a) it remove al related objects

System.out.println(al.contains("vinay"));//true

System.out.println(al);

System.out.println(a);

}

}

import java.util.\*;

class Demo {

public static void main(String[] arg)

{

// create default vector

ArrayList v = new ArrayList();

ArrayList v1 = new ArrayList();

v.add(0, 1);

v.add(1, 2);

v.add(2, "it company");

v.add(3, "virtusa");

v.add(4, 3);

v.add(5,"AjayReddy");

v1 = (ArrayList)v.clone( );

// checking vector

System.out.println("Clone of v: " + v1);

}

}

LinkedList: Insertion order is preserved and duplicates values are allowed.

to =====compare the ArrayList insertion and deletion is too simple than arraylist.

because in linked list data resides in nodes,

the diff between ArrayList and Linkedlist is where in ArrayList , if element is added or

delete in any other location then we reshuffle entire the arraylist , but

in linked list we can easily to add or delete any other element.

package com.Collections;

import java.util.LinkedList;

public class LinkedListEx

{

public static void main(String args[])

{

LinkedList l=new LinkedList();

l.add(10);

l.add(40);

l.add(2,50);//value 50 added at the index of 2

l.remove(0);//remove '0' index value

l.add('a');

l.add("AjayReddy");

l.add(10.9);

l.add('a');

l.addFirst('k');

l.addLast('m');

System.out.println(l);

//l.removeAll(l);

l.removeFirst();

l.removeLast();

System.out.println(l);

System.out.println(l.contains(40));

System.out.println(l.containsAll(l));

System.out.println(l.size());

}

}

Vector:

===============

This is a class implements Collection interface. Vector class is a legacy ============

class because all the methods which are having arraylist and linkedlist are common in vector class,

vector methods are synchronized, so thread safty is available, performance too low ,

one thread complete after another thead it execute.

by default 10 then after double it is incremented.ans:20

package com.Collections;

import java.util.\*;

public class Demo\_Vector

{

public static void main(String args[]) throws Exception

{

Vector v=new Vector();

v.add(10);

v.add(20);

v.add(10);

v.add(20);

v.add(10);

v.add(20);

v.add(10);

v.add(20);

v.add(10);

v.add(20);

v.add(10);

v.add(20);

System.out.println(v.capacity());

}

}

stack: stack is sub class , extends from Vector class. it follows LIFO method ,

it ========means last inserted element is deleted first.

Stack is also called as legacy classes because of the methods which are having the same

of the previous list classes.

package com.Collections;

import java.util.\*;

public class Demo\_Stack//legacy classes

{

public static void main(String args[])

{

Stack<String> stack = new Stack<String>();

stack.push("Mahesh");

stack.push("Sujit");

stack.push("Ramesh");

stack.push("Sachin");

stack.push("AjayReddy");

stack.pop();

//System.out.println(stack);

Iterator<String> itr=stack.iterator();

while(itr.hasNext()){

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

}

}

}

Set(I): Insertion order Not preserved and duplicates are Not allowed,

=====================================================================

HashSet(C); HashSet is a class , in this duplicates are not allowed and insertion ---

--------- order is not preserved . HashSet is not thread safe, so that

remaining threads also access the hashSet , Default initial capacity

of HashSet is 16.......

package com.Collections;

import java.util.HashSet;

public class HashSetEx

{

public static void main(String args[])

{

HashSet h=new HashSet();//insertion order not preserved

HashSet h1=new HashSet();

//h.add(10);//duplicates not allowed

h.add(10);

h.add('e');

h.add("AjayReddy");

h.add(null);//allows null values

//h.remove("strong");

//h.remove(10);

//h.clear();

//System.out.println(h.isEmpty());

h1=(HashSet)h.clone();

System.out.println(h.contains('e'));

System.out.println(h);

System.out.println("after cloning"+h1);

}

}

LinkedHashSet: Insertion order is preserved and duplicates are not allowed .

==============

Example program:

package com.Collections;

import java.util.LinkedHashSet;

public class LinkedHashSetEx

{

public static void main(String args[])

{

LinkedHashSet l=new LinkedHashSet();

l.add(10);//duplicates not allowed

l.add(10);

l.add(20);

l.add("ajay");//insertion order is preserved

System.out.println(l);

}

}

SortedSet(i): insertion order Not preserved, without duplicates.

==================

TreeSet(C): TreeSet class is implements from Navigable Interface ,

and navigable interface which is extends

again from SortedSet Interafce.

By default natural sorting order is follows as ascending (Comparable(I)).

\*\*\*\*\*\*Note: hetero genious elements are not allowed in in TreeSet class.

--->null values not allows(Empty TreeSet in that case OK null value accepted)

--->duplictes are not allowed

if user wants to create in descenidng order or customized order to

implements (Comparator(I)) .

package com.Collections;

import java.util.TreeSet;

public class Demo\_Treeset

{

public static void main(String args[])

{

TreeSet t=new TreeSet( );

t.add("A");

t.add("a");

t.add("B");

//t.add(10);//class cast Exception

t.add("L");

//t.add(null);//null pointer Exception

System.out.println(t);

}

}

sorting Order:

=========================================================

1. Comparable(I): default natural sorting order

2. Comparator(I): user to sort or customized sorting order

Comparable interface:

===========================

Java Comparable interface is used to order the objects of the user-defined class.

This interface is found in java.lang package and contains only one method named compareTo(Object).

It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single

data member only. For example, it may be rollno, name, age or anything else.

compareTo(Object obj) method

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

public int compareTo(Object obj): It is used to compare the current

object with the specified object.

-->It returns positive integer, if the current object is greater than the specified object.

-->negative integer, if the current object is less than the specified object.

-->zero, if the current object is equal to the specified object.

Collections class provides static methods for sorting the elements of collections.

If collection elements are of Set or Map, we can use TreeSet or TreeMap.

However,we cannot sort the elements of List. Collections class provides

methods for sorting the elements of List type elements.

Method of Collections class for sorting List elements

public void sort(List list): It is used to sort the elements of List. List elements must

be of the Comparable type.

Note: String class and Wrapper classes implement the Comparable interface by default.

So if you store the objects of string or wrapper classes in a list, set or map,

it will be Comparable by default.

Java Comparable Example:

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

Let's see the example of the Comparable interface that sorts the list elements on the basis of age.

--> Default nature sorting order .

-->java comparable is present in java.lang package

-->it contains only one method compareTo()

Example1:

======================

package com.Collections;

public class Demo\_Comparable2

{

public static void main(String args[])

{

System.out.println("A".compareTo("Z"));//-ve

System.out.println("Z".compareTo("K"));//+ve

System.out.println("A".compareTo("A"));//0

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

}

}

Comparator(I):

=================================================

Comparator for Customized order it means user defined order,

-->customized sorting order it follows

-->comes under java.util package

-->two methods compare( ) and equals( )

It returns positive integer, if the current object is less than the specified object.

negative integer, if the current object is greater than the specified object.

zero, if the current object is equal to the specified object.

package com.Collections;

import java.util.\*;

class TreeSetDemo

{

public static void main(String args[])

{

TreeSet t=new TreeSet(new MyComparator( ));

t.add(10);

t.add(0);

t.add(15);

t.add(5);

t.add(20);

t.add(20);

System.out.println(t);

}

}

class MyComparator implements Comparator

{

public int compare(Object obj1,Object obj2)

{

Integer i1=(Integer)obj1;

Integer i2=(Integer)obj2;

if(i1<i2)

{

return +1;

}

else if(i1>i2)

{

return -1;

}

else

{

return 0;

}

}

}

Queue:(Interaface)

========================

PriorityQueue(C): it is a class implements from Queue interafce.

Same as Queue it follows FIFO(First In First Out), it means first inserted element

only deleted first in the given array;

package com.Collections;

import java.util.\*;

public class Demo\_Queue

{

public static void main(String args[])

{

PriorityQueue queue=new PriorityQueue();

queue.add(10);

queue.add(20);

queue.add(30);

queue.add(40);

queue.add(50);

System.out.println("head:"+queue.element());

System.out.println("head:"+queue.peek());

System.out.println("iterating the queue elements:");

System.out.println(queue);

queue.remove();

queue.poll();

System.out.println("after removing two elements:");

System.out.println(queue);

}

}

Ex: (studentid,100) pair

key , value

not duplicate duplicates

(custid, 101)

Java Map Interface:

===============================

A map contains values on the basis of key, i.e. key and value pair.

Each key and value pair is known as an entry. A Map contains unique keys.

A Map is useful if you have to search, update or delete elements on the basis of a key.

A Map doesn't allow duplicate keys, but you can have duplicate values.

HashMap and LinkedHashMap allow null keys and values,

but TreeMap doesn't allow any null key or value.

Java Map Hierarchy

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

There are two interfaces for implementing Map in java: Map and SortedMap and NavigableSortedMap,

and three classes:

HashMap, LinkedHashMap, and TreeMap. The hierarchy of Java Map is given below:

HashMap(C):

======================

java HashMap class implements the map interface by using a hash table.

Default initial capacity of Hashmap is 16.

Points to remember

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

Java HashMap class contains values based on the key.

Java HashMap class contains only unique keys.

Java HashMap class may have one null key and multiple null values.

Java HashMap class is non synchronized.

Java HashMap class maintains no order.

The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

HashMap class declaration

Let's see the declaration for java.util.HashMap class.

HashMap class Parameters

Let's see the Parameters for java.util.HashMap class.

K: It is the type of keys maintained by this map.

V: It is the type of mapped values.

Example program on HashMap:

=====================================

package com.Collections;

import java.util.Collection;

import java.util.HashMap;

import java.util.Set;

public class demo\_hashmap

{

public static void main(String args[])

{

HashMap m=new HashMap();

m.put("Ramesh",700);

m.put("mahesh",800);

m.put("rajesh",200);

m.put("naresh",500);

System.out.println(m);//insertion based on hashcode

System.out.println(m.put("Ramesh",1000));

Set s=m.keySet();//display the key values

System.out.println(s);

Collection c=m.values();//display values

System.out.println(c);

Set s1=m.entrySet();//display key and values

System.out.println(s1);

}

}

import java.util.\*;

public class MapExample1

{

public static void main(String[] args)

{

Map map=new HashMap();

//Adding elements to map

map.put(1,"Amit");

map.put(5,"Rahul");

map.put(2,"Jai");

map.put(6,"Amit");

for(Map.Entry m:map.entrySet())

{

System.out.println(m.getKey()+" "+m.getValue());

}

}

}

output:

=========

1 Amit

2 Jai

5 Rahul

6 Amit

Example program on HashTable:

==========================

Java Hashtable class implements a hashtable, which maps keys to values.

It inherits Dictionary class and implements the Map interface.

A Hashtable is an array of a list. Each list is known as a bucket.

The position of the bucket is identified by calling the hashcode() method.

A Hashtable contains values based on the key.

-->Java Hashtable class contains unique elements.

-->Java Hashtable class doesn't allow null key or value.

-->Java Hashtable class is synchronized.

-->The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.

Let's see the Parameters for java.util.Hashtable class.

K: It is the type of keys maintained by this map.

V: It is the type of mapped values.

for Ex:

HashTable t=new HashTable();

t.put(1,20);

t.put(10,2);

t.put(15,30);

here hashtable bucket size only 11, from 0->10

but in the above example 15 , so in that case 15/11->4

in that hashcode 4 bucket it stores value 30.

if

t.put(48,100);

in that case 48/11-> 4 again , so it store in the same bucket

hashcode 4 it stores value 100, after 15->30, 48->100

but read that value it follows from top to bottom and right to left

then output follows like this

10->2,48->100,15->30,1->20 is an output.

example:

=====================

package com.test;

import java.util.\*;

class MyThread

{

public static void main(String[] args) {

Hashtable t=new Hashtable();

t.put(1,10);

t.put(4,50);

t.put(15,100);

t.put(10, 100);

//t.put(null,50);

//t.put(3,null);

System.out.println(t);

}

}

output:

{10=100, 15=100, 4=50, 1=10}

if both statement disable comment then we get

===========================================================

Exception in thread "main" java.lang.NullPointerException

at java.util.Hashtable.put(Hashtable.java:460)

at com.test.MyThread.main(MyThread.java:12)

package com.Collections;

import java.util.\*;

class hashTabledemo {

public static void main(String[] arg)

{

// creating a hash table

Hashtable<Integer, String> h =

new Hashtable<Integer, String>();

Hashtable<Integer, String> h1 =

new Hashtable<Integer, String>();

h.put(3, "virtusa");

h.put(2, "IT company");

h.put(1, "DLF");

// create a clone or shallow copy of hash table h

h1 = (Hashtable<Integer, String>)h.clone();

// checking clone h1

System.out.println("values in clone: " + h1);

// clear hash table h

h.clear();

// checking hash table h

System.out.println("after clearing: " + h);

}

}

---->Why we can go for Concurrent collections?

1)collection objects ArrayList,LinkedList,HashSet,TreeSet are not thread safe

because multiple thread are accessing single object

becuase of that it is not thread safe

2) vector, HashTable

synchronizedList()

synchronizedSet() are thread safe

but performance wise it is poor

3) T1 thread accessing one object , other thread

modify that same object,in that case we get

error as "ConcurrentModificationException"

import java.util.\*;

class MyThread extends Thread

{

static ArrayList l=new ArrayList();

public void run()

{

try

{

Thread.sleep(2000);

}

catch(InterruptedException e)

{

}

System.out.println("child thread updating list");

l.add("D");

}

public static void main(String args[]) throws InterrutedException

{

l.add("A");

l.add("B");

l.add("C");

Mythread t=new MyThread();

t.start();

Iterator itr=l.iterator();

while(itr.hasNext())

{

String s1=(String)itr.next();

syso("main thread iterating list and current thread is"+s1);

Thread.sleep(3000);

}

System.out.println(l);

}

}

output:

====================

main thread iterating list and current thread isA

child thread updating list

Exception in thread "main" java.util.ConcurrentModificationException

at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:909)

at java.util.ArrayList$Itr.next(ArrayList.java:859)

at com.test.MyThread.main(MyThread.java:28)

So far that we can go for concurrent collections

Concurrent collections:

======================

1)more thread safe

2)high performance beacuase of different locking mechanisam

3)concurrent modification we can do we are not going to get CMException....

concurrent collection classes:

=====\*\*\*\*\*++++++===============

1) ConcurrentHashMap

2) CopyOnWriteArrayList

3) CopyOnWriteArraySet

java.util.concurrent package:

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

Map(I)-->ConcurrentMap(I)-->ConcurrentHashMap(c)

ConcurrentMap(I) follows the given methods

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

in ConcurrentMap has 3 special methods they are like

\* putIfAbsent()

\* remove()

\* replace()

1)Object putIfAbsent(Object key,Object value),same like put method but if key

is already there it wont add new value.

2)boolean remove(Object key,Object value),we need pass key and value to remove...

3)boolean replace(object key,object oldvalue,object newvalue)

ConurrentHashMap:---------------

hashmap -->not thread safe

hashtable-->thread safe

concurrenthashmap-->threadsafe

default intial size of hasmap,concurrenthashmap is 16..

0 to 15 buckets we are going to have ...

when we are working with hashtable the lock for complete object no other thread

can write or update even read also

bucket level locking,segment level locking:

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

\*In concurrenthashmap multiple thread can read at a time and for

write and update bucket lock is required

so in concurrenthashmap at a time 16 threads can execute....(16 concurrency level)

Example program:

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

import java.util.concurrent.\*;

class Test

{

public static void main(String args[])

{

ConcurrentHashmap m=new ConcurrentHashMap();

m.put(101,"A");

m.put(102,"B");

m.putIfAbsent(103,"C");

m.putIfAbsent(101,"D");

m.remove(101,"D");

m.replace(102,"B","E");

System.out.println(m);

}

}

CopyOnWriteArrayList:

======================

\*it is realted to arraylist...

\*it is concurrent version arraylist(thread safe version of arraylist)

\* it is a child class List(I)...like arraylist

\*\* for every write or update operation on cloned object,then there is no effect on original object.

@certain point two objects will be sync...

Methods of CopyOnWriteArrayList:

================================

1)boolean addIfAbsent(Object o);

2)int addAllIfAbsent(Object o);

CopyOnWriteArraySet:

===================

\*It is a thread safe version of set.

\* child class Set interface.

\*Insertion order is Preserved

\*Duplicate are not allowed.

\*Multiple thread can able read operation simultaneously but for every update operation a separate

cloned copy will created.

creating a sample program using colletions for crud operations:

==============================================================

package com.capgemini;

public class Employee

{

private String empname;

private String empaddr;

private int empsal;

public String getEmpname() {

return empname;

}

public void setEmpname(String empname) {

this.empname = empname;

}

public String getEmpaddr() {

return empaddr;

}

public Employee(String empname, String empaddr, int empsal) {

super();

this.empname = empname;

this.empaddr = empaddr;

this.empsal = empsal;

}

public void setEmpaddr(String empaddr) {

this.empaddr = empaddr;

}

public int getEmpsal() {

return empsal;

}

public void setEmpsal(int empsal) {

this.empsal = empsal;

}

@Override

public String toString() {

return "Employee [empname=" + empname + ", empaddr=" + empaddr + ", empsal=" + empsal + "]";

}

}

package com.capgemini;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map.Entry;

import java.util.Scanner;

import java.util.Set;

public class TestEmployee

{

public static void main(String[] args) {

Employee e1=new Employee("suresh","Hyderabad",5000);

Employee e2=new Employee("Ramesh","secunder",6000);

Employee e3=new Employee("rajesh","venkatapur",7000);

HashMap<Integer,Employee> h=new HashMap<Integer,Employee>();

h.put(1,e1);

h.put(2,e2);

h.put(3,e3);

System.out.println("Employee Application Details");

System.out.println("1. Read Employee Deatails");

System.out.println("2. Delete record details");//delete

System.out.println("3. Fetching all records");//select

System.out.println(" 4.Add employee records");//create

System.out.println("5. update employee salary");//update

Scanner sc=new Scanner(System.in);

System.out.println("select one option");

int option=sc.nextInt();

switch(option)

{

case 1:

System.out.println("read employee details");

int eid=sc.nextInt();

Employee result=h.get(eid);

System.out.println(result);

break;

case 2:

System.out.println("delete empoyee record");

int eid1=sc.nextInt();

h.remove(eid1);

System.out.println("after delete employee record");

System.out.println(h);

break;

case 3:

System.out.println("fetching records from employee");

Set set= h.entrySet();

Iterator itr=set.iterator();

while(itr.hasNext())

{

Entry emp=(Entry) itr.next();

System.out.println(emp.getKey()+" "+emp.getValue());

}

break;

case 4:

System.out.println("adding the records of employee");

int key=sc.nextInt();

System.out.println("enter the employee name");

String name=sc.next();

System.out.println("enter the employee address");

String address=sc.next();

System.out.println("enter the employee salary");

int sal=sc.nextInt();

Employee e=new Employee(name,address,sal);

h.put(key,e);

System.out.println("after adding employee record");

System.out.println(h);

break;

case 5:

System.out.println("Employee update information");

int ei=sc.nextInt();

Object result1=h.get(ei);

Employee emp=(Employee) result1;

System.out.println("employee old salary is"+emp.getEmpsal());

System.out.println("enter salary to update");

emp.setEmpsal(9500);

System.out.println("after update employee salary is"+emp);

break;

default:

System.out.println("exit from the options");

}

}

}