**Collection API**

Collection API means collections of different kind of objects or can be similar kind also or we can say a group collection of different types of object is called Collection API. Here whatever program we will do everything based on different kind of objects. As we studied in Array we can keep only similar kind of objects not different kind of objects. So to overcome this problem Java People introduced this Collection API on jdk 1.2 onwards. This is very flexible way to use different kind of objects.

This entire Collection API is available inside this java.util.\* package.

The entire Collection API is divided into four parts.

1, List

2. Set

3. Map

4. Queue

While Collection API introduced that time its allow only derived data types not primitive. But on jdk 1.5 onwards now we can add primitive data types also. Because when autoboxing is coming to Java on jdk 1.5 onwards after that we can use primitive datatypes for Collection API.

While we are passing any primitive value into Collection API objects then it will automatically converted into that related Wrapper class object types and then we can use them. So this process is called Autoboxing.

All Collection API class override toString(), equals() and hashCode() method from Object class.

**SignificantofListStream:**

1. List stream providing auto indexing process.
2. Its allow duplicate values.
3. It allows null values also.
4. There is no default sorting mechanism is available inside this List Stream. But in order to get this sorting mechanism then we have to use Collections class sort() method.
5. Main class under this List Stream is:  
   5.1 ArrayList
   1. LinkedList
   2. Vector
6. Among all these class vector class is thread safe class and also synchronized class.

In jdk 1.4 if we are adding primitive value inside this ArrayList object then we will get Compile Time Error. Here we are adding primitive value inside this ArrayList object but it required object of the class. So we will get Compile Time Error while running this program in case of jdk 1.4 versions onwards. But now we will not get any Compile Time Error because in these above versions inside jdk **sun microsystem** developer added new thing that Autoboxing which will convert primitive value to Wrapper class value. So we are not getting Compile Time Error while running this program.

As we know that Collection API override toString() method that means ArrayList class also override this toString() method inside this class for which while we are trying to print this class object reference using SOP statement then we are getting value inside object not the memory address location of the object.

If we want to see all elements value inside this ArrayList then we have to use this size() method for iteration or looping. Even get() method also we can call the value of a particular element inside this ArrayList.

**get():**

return type of this method is object types.

**size():**

return type of this method is int type.

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes1

{

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

{

ArrayList list = **new** ArrayList();

list.add(100);

list.add(200);

list.add(200.090);

list.add(**true**);

list.add("abc");

list.add('a');

System.*out*.println("Value of List is: "+list);

}

}

/\*

Here we are trying to check that how primitive value is changing to wrapper class types.

Here is the output of this program:

Value of List is: [100, 200, 200.09, true, abc, a]

If we will run this program on jdk 1.4 then that time it will show Compilation Time Error. Because that time Autoboxing is not added in Java. So it will shows Compilation Error. But while we

are running now we are not getting Compilation Error.

Here in this program we are also getting proper output while we are printing this ArrayList object in SOP statements. That means we understand that toString() method also got override

inside this class. So for this reason we are not getting memory address instead of proper output.

\*/

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes2

{

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

{

ArrayList list = **new** ArrayList();

list.add(100);

list.add(200);

list.add(200.090);

list.add(**true**);

list.add("abc");

list.add('a');

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

System.*out*.println("Using Classical for loop: ");

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

**for** (**int** i = 0; i < list.size(); i++)

{

System.*out*.println(list.get(i));

}

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

System.*out*.println("Using Enhanced for Loop: ");

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

**for** (Object obj : list)

{

System.*out*.println(obj);

}

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

}

}

/\*

Here in this program we are trying to iterate this ArrayList class object to fetch the values inside this class object. So for this reason using for loop we are also using size() method for fetching

the total size of the ArrayList and then we are fetching the value based on this size.

Here is the output of this program:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Using Classical for loop:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

100

200

200.09

true

abc

a

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Using Enhanced for Loop:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

100

200

200.09

true

abc

a

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

How to fetch particular element value from an ArrayList?

Using this get() method we can change this value of an element from and ArrayList.

**Syntax:**int i = (Integer)list.get(0);

How to change value of an element of an ArrayList?

Here using index position of an element of an ArrayList we can change the value of an element.

Here is the syntax for changing the ArrayList element.

list.add(1, 500);

Here we are using first index position we are changing the value of element value. Of an ArrayList.

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes3

{

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

{

ArrayList list = **new** ArrayList();

list.add("Sailendra");

list.add(100);

list.add(200.20);

list.add(10.5f);

System.*out*.println("Value of list before changing any element value: ");

System.*out*.println(list);

list.add(1, 500);

System.*out*.println("Value of list after changing element value: ");

System.*out*.println(list);

}

}

/\*

Here in this program we are first printing the value of list before changing any element of an ArrayList but after changing element of an ArrayList also we are checking the value of list. So

here we are checking after changing the value of list using this below syntax and then we want to check this value.

Syntax:

list.add(1, 500);

Here inside add method two parameters we are passing one is for index position and the new value add into that position.

Here is the output of this program:

Value of list before changing any element value:

[Sailendra, 100, 200.2, 10.5]

Value of list after changing element value:

[Sailendra, 500, 100, 200.2, 10.5]

\*/

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes4

{

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

{

ArrayList list = **new** ArrayList();

System.*out*.println("Value of list before adding elements");

System.*out*.println(list);

list.add("Sailendra");

list.add(100);

list.add(200.00);

list.add(20.30f);

list.add(**true**);

System.*out*.println("Value of list after adding elements");

System.*out*.println(list);

}

}

/\*

Here in this program we are checking before adding elements into this ArrayList and after adding value of this ArrayList.

Here is the output of this program:

Value of list before adding elements

[]

Value of list after adding elements

[Sailendra, 100, 200.0, 20.3, true]

\*/

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes5

{

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

{

ArrayList list = **new** ArrayList();

System.*out*.println("Value of list before adding element into ArrayList");

System.*out*.println(list);

list.add("Sailendra");

list.add(100);

list.add(200.50);

list.add(20.50f);

list.add(**true**);

System.*out*.println("Value of list after adding elements into ArrayList");

System.*out*.println(list);

ArrayList list1 = **new** ArrayList(list);

System.*out*.println("value of list2 after adding list object into ArrayList constructor");

System.*out*.println(list1);

}

}

/\*

Here in this program we are checking before adding elements into an ArrayList and after adding elements into ArrayList and also we are adding this list object into another ArrayList and

checking this another ArrayList object reference value.

Here is the output of this program:

Value of list before adding element into ArrayList

[]

Value of list after adding elements into ArrayList

[Sailendra, 100, 200.5, 20.5, true]

value of list2 after adding list object into ArrayList constructor

[Sailendra, 100, 200.5, 20.5, true]

\*/

One ArrayList container contains another ArrayList container’s value. For getting this value we have to use this syntax:

ArrayList list = new ArrayList(list1);

Here list1 is the reference of one ArrayList where we are adding this list value into another ArrayList list.

How to all elements of one ArrayList value into another ArrayList Value?

Using addAll() method we can add all element of an ArrayList into another ArrayList.

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes6

{

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

{

ArrayList list = **new** ArrayList();

list.add(90);

list.add(0);

list.add(9);

System.*out*.println("Value of list");

System.*out*.println(list);

ArrayList list1 = **new** ArrayList();

list1.add(100);

list1.add(10);

list1.add(0);

list1.add(1, list);

list.addAll(list);

System.*out*.println("Value of list1");

System.*out*.println(list1);

}

}

/\*

Here in this program we have created two ArrayList i.e list and list1. Here we are adding one ArrayList element into another ArrayList with some specific index position.

Here is the output of this program:

Value of list

[90, 0, 9]

Value of list1

[100, [90, 0, 9], 10, 0]

Here is the output of this program using addAll() method

Value of list

[90, 0, 9]

Value of list1

[100, [90, 0, 9, 90, 0, 9], 10, 0]

\*/

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"rawtypes", "unchecked"})

**public** **class** ArrayListNotes7

{

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

{

ArrayList list = **new** ArrayList();

list.add("Sailendra");

list.add(100);

list.add(20.50);

list.add(200.30f);

list.add(**true**);

System.*out*.println("Value of list");

System.*out*.println(list);

ArrayList list1 = **new** ArrayList();

list1.add(50);

list1.add(6);

list1.addAll(list);

list1.add(30);

System.*out*.println("Valus of list1");

System.*out*.println(list1);

}

}

/\*

Here we have used addAll() method in second ArrayList by passing list reference of first ArrayList.

Here is the output of this program:

Value of list

[Sailendra, 100, 20.5, 200.3, true]

Valus of list1

[50, 6, Sailendra, 100, 20.5, 200.3, true, 30]

\*/

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"rawtypes", "unchecked"})

**public** **class** ArrayListNotes8

{

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

{

ArrayList list1 = **new** ArrayList();

list1.add(90);

list1.add(9);

list1.add(0);

System.*out*.println("Value of list1");

System.*out*.println(list1);

ArrayList list2 = **new** ArrayList();

list2.add("Begin");

list2.add("end");

list2.add(1, list1);

System.*out*.println("Value of list2");

System.*out*.println(list2);

}

}

/\*

Here in this program we are trying to add list1 elements into list2 with some specific index position.

Here is the output of this program:

Value of list1

[90, 9, 0]

Value of list2

[Begin, [90, 9, 0], end]

\*/

How many way to read elements of an ArrayList?

There are four way to read elements of an ArrayList.

1. Classical for Loop.
2. Enhanced for Loop.
3. Iterator process.
4. List Iterator process.
5. toString() Process.

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.ListIterator;

@SuppressWarnings({"rawtypes", "unchecked"})

**public** **class** ArrayListNotes9

{

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

{

ArrayList list1 = **new** ArrayList();

list1.add(9000);

list1.add(0);

list1.add(90);

list1.add(9);

System.*out*.println("Using Classical for Loop");

**for** (**int** i = 0; i < list1.size(); i++)

{

System.*out*.println(list1.get(i));

}

System.*out*.println("Using Enhanced for Loop");

**for** (Object obj : list1)

{

System.*out*.println(obj);

}

System.*out*.println("Using toString() Process");

System.*out*.println(list1);

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

Iterator it = list1.iterator();

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

{

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

}

System.*out*.println("Using List Iterator Process");

ListIterator lit = list1.listIterator();

**while** (lit.hasNext())

{

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

}

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

System.*out*.println("Using ListIterator Previous Process");

**while** (lit.hasPrevious())

{

System.*out*.println(lit.previous());

}

}

}

/\*

Here in this program we are trying to print all elements of an ArrayList in different ways.

Using Classical for Loop

9000

0

90

9

Using Enhanced for Loop

9000

0

90

9

Using toString() Process

[9000, 0, 90, 9]

Using Iterator Process

9000

0

90

9

Using List Iterator Process

9000

0

90

9

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Using ListIterator Previous Process

9

90

0

9000

\*/

How to remove elements from an ArrayList?

Using remove() method of ArrayList class we can remove any element from an ArrayList.

We are passing index position as an argument of remove() where JVM will search that index position and then remove that element from an ArrayList as well as re-arrange elements.

Syntax:

List.remove(1);

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes10

{

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

{

ArrayList list = **new** ArrayList();

list.add("abc");

list.add("xyz");

list.add("hello");

list.add("test");

list.add("end");

System.*out*.println("Before removing element form list: ");

System.*out*.println(list);

Object obj = list.remove(1);

System.*out*.println("After removing one element from list:");

System.*out*.println(obj);

**boolean** flag = list.remove("test");

System.*out*.println(flag);

System.*out*.println(list);

}

}

/\*

Here in this program we are removing elements from an ArrayList using remove() method of ArrayList class. Using this method we are passing index position of an element which we wants

to remove from the list.

Here is the output of this program:

Before removing element form list:

[abc, xyz, hello, test, end]

After removing one element from list:

xyz

true

[abc, hello, end]

Here in this program we just first print the list reference of an ArrayList, after that we are removing index position of 1 element and store that element into obj of class Object and then print

that obj. After this again we remove another element name "test" from the list and then check whether that element actually remove from the list or not and pass the status for this in boolean

reference and then print that status whether that element got removed or not. After all remove process doing we want to to check again the list values, so for this reason we again print this

list value.

\*/

How to remove all elements from an ArrayList?

Using removeAll() method of ArrayList class we can remove complete elements from an ArrayList.

Syntax:

list.removeAll(Collection c);

Here we have to pass the reference of that class reference whose element we are going to remove from the list.

This removeAll() method also remove all those elements which are common between two list. If we are trying to remove those elements common in both lists then in that case this method is good for use, which will remove those elements.

Syntax:

list1.removeAll(list2);

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes11

{

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

{

ArrayList list = **new** ArrayList();

list.add("Sailendra");

list.add(100);

list.add(20.30);

list.add(30.50f);

list.add(**true**);

list.add(9599480640l);

System.*out*.println("Elements inside l;ist before remove all elements.");

System.*out*.println(list);

**boolean** status = list.removeAll(list);

System.*out*.println("Elements inside list after remove elements.");

System.*out*.println(list);

System.*out*.println("Remove elements from an ArrayList: "+status);

}

}

/\*

Here we are trying to remove all element from an ArrayList using this removeAll() method. Using this method we are passing ArrayList object reference into this method and also calling

this method using that same object reference. After remove all elements from an ArrayList then we are passing status for this remove into a boolean object reference which will provide the

actual output that whether all elements got remove or not.

Here is the output of this program:

Elements inside l;ist before remove all elements.

[Sailendra, 100, 20.3, 30.5, true, 9599480640]

Elements inside list after remove elements.

[]

Remove elements from an ArrayList: true

\*/

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes12

{

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

{

ArrayList list1 = **new** ArrayList();

list1.add("Sailendra");

list1.add(100);

list1.add(20.30);

list1.add(30.50f);

list1.add(**true**);

list1.add(9599480640l);

System.*out*.println("Elements in list1 before remove elements: "+list1);

ArrayList list2 = **new** ArrayList();

list2.add("Soumya");

list2.add(5);

list2.add(20.30);

list2.add(53.38f);

list2.add(9731728407l);

list2.add(**true**);

System.*out*.println("Elements in list2 before remove elements: "+list2);

list1.removeAll(list2);

System.*out*.println("Elements in list1 after remove elements: "+list1);

System.*out*.println("Elements in list2 after remove elements: "+list2);

}

}

/\*

Here in this program we are checking common elements between two lists and then remove those elements from the list from which list you want to remove these common elements.

Here is the output of this program:

Elements in list1 before remove elements: [Sailendra, 100, 20.3, 30.5, true, 9599480640]

Elements in list2 before remove elements: [Soumya, 5, 20.3, 53.38, 9731728407, true]

Elements in list1 after remove elements: [Sailendra, 100, 30.5, 9599480640]

Elements in list2 after remove elements: [Soumya, 5, 20.3, 53.38, 9731728407, true]

\*/

How to get common elements in both ArrayList?

Using retainAll() method we can get common elements between two ArrayList.

Syntax:

list1.retainAll(list2);

Here we are passing list2 reference in retainAll() method due to that we have to compare list2 all elements with list1 all elements.

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes13

{

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

{

ArrayList list1 = **new** ArrayList();

list1.add("Sailendra");

list1.add(100);

list1.add(20.30);

list1.add(30.50f);

list1.add(**true**);

ArrayList list2 = **new** ArrayList();

list2.add("Soumya");

list2.add(200);

list2.add(20.30);

list2.add(30.50f);

list2.add(**true**);

System.***out***.println("Common elements between two ArrayList is available: ");

**boolean** status = list1.retainAll(list2);

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

}

}

/\*

Here in this program we have to check common elements between two ArrayList object references i.e list1 and list2. Here we are passing list2 reference in retainAll() method which will

check the common elements between these and then reply the status that whether common element is there or not.

Here is the output of this program:

Common elements between two ArrayList is available:

true

\*/

**How many way to remove elements from an ArrayList?**

Here we have already mentioned the way for remove elements from an ArrayList using this below process:

1. Using remove() method.
2. Using retainAll() method.
3. Using reomoveAll() method.
4. Using clear() method.

**How to clear all elements from an ArrayList?**

Using clear() method we can also clear all elements from an ArrayList.

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes14

{

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

{

ArrayList list = **new** ArrayList();

list.add("Sailendra");

list.add(100);

list.add(20.30);

list.add(30.40f);

list.add(9599480640l);

list.add(**true**);

System.*out*.println("Elements before clearing process: ");

System.*out*.println(list);

list.clear();

System.*out*.println("Elements after clearing process: ");

System.*out*.println(list);

}

}

/\*

Here in this program we are trying to clear all elements from an ArrayList. So here we have used clear() method of ArrayList class. Using this method we are going to remove all elements

from the ArrayList.

Here is the output of this program:

Elements before clearing process:

[Sailendra, 100, 20.3, 30.4, 9599480640, true]

Elements after clearing process:

[]

\*/

**How to updates element from an existing ArrayList?**

Here in this program we are going to change the existing element value to another value. Using set() method we can change the value of an existing elements of an ArrayList.

Syntax:

list1.set(2, “new”);

Here in this line in set() method we are passing two arguments one is for indicating index position and another one is the value which will put there instead of old value in an ArrayList.

So for this reason we are using set() method.

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes15

{

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

{

ArrayList list = **new** ArrayList();

list.add("Sailendra");

list.add(100);

list.add(20.30);

list.add(30.50f);

list.add(9599480640l);

list.add(**true**);

System.*out*.println("Elements of list before change element value:");

System.*out*.println(list);

list.set(3, "Soumya");

System.*out*.println("Elements in list after change elements value:");

System.*out*.println(list);

}

}

/\*

Here using set() method of ArrayList class we are changing the third elements value. So we are checking this list elements value before and after changing value using set() method. Inside

set() method is taking two parameters one is for index position of an element and another is for new value which we are going to add into that element position.

Here is the output of this program:

Elements of list before change element value:

[Sailendra, 100, 20.3, 30.5, 9599480640, true]

Elements in list after change elements value:

[Sailendra, 100, 20.3, Soumya, 9599480640, true]

\*/

**How to sorting an elements of an ArrayList?**

In Collection API if we want to sorting an elements then in that case we required Collection class sort() method where we are passing any collection object reference then we are getting this sorted format elements into that reference.

In a collection of an Array we are passing null as an element and also trying to sort that element then we will get NullPointerException. Because we are trying to sort null value into another values so we are getting this exception.

While we are sorting an Collection objects then elements inside this Collection object should be similar kind not different kind. If we will use more than one kind of elements into Collection Objects then we will get Runtime Exception that is ClassCasteException.

Syntax:

**Collectios.sort(list);**

Here list is the reference object of an ArrayList or List or can be any kind of references related to Collection API.

In case of primitive Array we are also using another sort() method which is available inside Arrays class. If we are declaring any primitive array and we want to sort that array elements then we can use predefined method sort() of Arrays class.

Syntax:

**Arrays.sort(Object[] obj);**

Example:

package com.lara.CollectionNotes.ListNotes.ArrayListNotes;

import java.util.ArrayList;

import java.util.Collections;

@SuppressWarnings({"unchecked", "rawtypes"})

public class ArrayListNotes16

{

public static void main(String[] args)

{

ArrayList list1 = new ArrayList();

list1.add(90);

list1.add(0);

list1.add(9);

list1.add(910);

list1.add(100);

list1.add("Sailendra");

list1.add(90);

list1.add(0);

list1.add(9);

list1.add(910);

list1.add(100);

System.out.println("Before Sorting element in list1: ");

System.out.println(list1);

Collections.sort(list1);

System.out.println("After Sorting elements in list1: ");

System.out.println(list1);

}

}

/\*

Here in this program we have used Collections sort() method for sorting an elements. So we are getting sorting ascending format elements.

Here is the output of this program:

Before Sorting element in list1:

[90, 0, 9, 910, 100]

After Sorting elements in list1:

[0, 9, 90, 100, 910]

Here in sorting we can sort only similar kind of elements not different kind elements. If we add different kind of elements into this list1 then we will get ClassCasteException that cannot be

able to convert. So for sorting we have to pass similar kind of elements.

Here is the output of this kind of program:

In this list1 we have added a String value inside this lis1 and trying to sort this list1.

Before Sorting element in list1:

[Sailendra, 50, 20, 80, 30, 5]

Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer

at java.lang.Integer.compareTo(Unknown Source)

at java.util.ComparableTimSort.countRunAndMakeAscending(Unknown Source)

at java.util.ComparableTimSort.sort(Unknown Source)

at java.util.ComparableTimSort.sort(Unknown Source)

at java.util.Arrays.sort(Unknown Source)

at java.util.Collections.sort(Unknown Source)

at com.lara.CollectionNotes.ListNotes.ArrayListNotes.ArrayListNotes16.main(ArrayListNotes16.java:30)

Here we got exception due to that String value can't be converted into int value. So for this reason throwing exception as ClassCasteException.

\*/

Example:

package com.lara.CollectionNotes.ListNotes.ArrayListNotes;

import java.util.ArrayList;

import java.util.Collections;

@SuppressWarnings({"unchecked", "rawtypes"})

public class ArrayListNotes17

{

public static void main(String[] args)

{

ArrayList list = new ArrayList();

list.add("abc");

list.add("ABc");

list.add("Abc");

list.add("ABC");

list.add("123");

list.add("Sailendra");

list.add("Bhagyalaxmi");

System.out.println("Before Sorting an elements of an ArrayList: ");

System.out.println(list);

Collections.sort(list);

System.out.println("After Sorting an elements of an ArrayList: ");

System.out.println(list);

}

}

/\*

Here in this program we have added String types of elements into an ArrayList list and trying to sort this element using sort() method of Collections class.

Here is the output of this program:

Before Sorting an elements of an ArrayList:

[abc, ABc, Abc, ABC, 123, Sailendra, Bhagyalaxmi]

After Sorting an elements of an ArrayList:

[123, ABC, ABc, Abc, Bhagyalaxmi, Sailendra, abc]

Sorting can't be happen more than one app. If we will sort more than one app then it will throws ClassCasteException due to unable to convert.

\*/

Example:

package com.lara.CollectionNotes.ListNotes.ArrayListNotes;

import java.util.ArrayList;

import java.util.Collections;

@SuppressWarnings({"unchecked", "rawtypes"})

public class ArrayListNotes18

{

public static void main(String[] args)

{

ArrayList list = new ArrayList();

list.add(12);

list.add(12.0);

list.add("12");

list.add("12.0");

System.out.println("Before Sorting element into list: ");

System.out.println(list);

Collections.sort(list);

System.out.println("After sorting elements in list: ");

System.out.println(list);

}

}

/\*

Here in this program we have added more than one data types with different kind and also trying to sort these all elements.

Here is the output of this program:

Before Sorting element into list:

[12, 12.0, 12, 12.0]

Exception in thread "main" java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.Double

at java.lang.Double.compareTo(Double.java:49)

at java.util.ComparableTimSort.countRunAndMakeAscending(ComparableTimSort.java:290)

at java.util.ComparableTimSort.sort(ComparableTimSort.java:157)

at java.util.ComparableTimSort.sort(ComparableTimSort.java:146)

at java.util.Arrays.sort(Arrays.java:472)

at java.util.Collections.sort(Collections.java:155)

at com.lara.CollectionNotes.ListNotes.ArrayListNotes.ArrayListNotes18.main(ArrayListNotes18.java:20)

\*/

Example:

package com.lara.CollectionNotes.ListNotes.ArrayListNotes;

import java.util.ArrayList;

import java.util.Collections;

@SuppressWarnings({"unchecked", "rawtypes"})

public class ArrayListNotes19

{

public static void main(String[] args)

{

ArrayList list = new ArrayList();

list.add(**null**);

list.add(10);

list.add(100);

list.add(1);

list.add(200);

System.out.println("Before Sorting elements into an ArrayList list: ");

System.out.println(list);

Collections.sort(list);

System.out.println("After Sorting elements into an ArrayList list: ");

System.out.println(list);

}

}

/\*

Here in this program we have added null value as an element into an ArrayList list and we are trying to sort this list. As we know that here also null value is different kind of value as compare

to int value. So it will throws NullPointerException.

Here is the output of this program:

Before Sorting elements into an ArrayList list:

[null, 10, 100, 1, 200]

Exception in thread "main" java.lang.NullPointerException

at java.lang.Integer.compareTo(Integer.java:1003)

at java.lang.Integer.compareTo(Integer.java:52)

at java.util.ComparableTimSort.countRunAndMakeAscending(ComparableTimSort.java:290)

at java.util.ComparableTimSort.sort(ComparableTimSort.java:157)

at java.util.ComparableTimSort.sort(ComparableTimSort.java:146)

at java.util.Arrays.sort(Arrays.java:472)

at java.util.Collections.sort(Collections.java:155)

at com.lara.CollectionNotes.ListNotes.ArrayListNotes.ArrayListNotes19.main(ArrayListNotes19.java:22)

\*/

Example:

package com.lara.CollectionNotes.ListNotes.ArrayListNotes;

import java.util.ArrayList;

import java.util.Collections;

class A

{

int i;

}

@SuppressWarnings({"unchecked", "rawtypes"})

public class ArrayListNotes20

{

public static void main(String[] args)

{

A a1 = new A();

a1.i = 100;

A a2 = new A();

a2.i = 10;

A a3 = new A();

a3.i = 200;

A a4 = new A();

a4.i = 50;

A a5 = new A();

a5.i = 30;

ArrayList list = new ArrayList();

list.add(a1);

list.add(a2);

list.add(a3);

list.add(a4);

list.add(a5);

System.*out*.println("Before Sorting Elements into an ArrayList list: ");

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

Collections.sort(list);

System.*out*.println("After Sorting Elements into an ArrayList list: ");

System.*out*.println(list);

}

}

/\*

Here in this program we have to added elements as class object references. Using this list we want to sort all list.

Here is the output of this program:

Before Sorting Elements into an ArrayList list:

[com.lara.CollectionNotes.ListNotes.ArrayListNotes.A@41a80e5a, com.lara.CollectionNotes.ListNotes.ArrayListNotes.A@465863,

com.lara.CollectionNotes.ListNotes.ArrayListNotes.A@a54cbb9, com.lara.CollectionNotes.ListNotes.ArrayListNotes.A@450ae3fb,

com.lara.CollectionNotes.ListNotes.ArrayListNotes.A@4dfd9726]

Exception in thread "main" java.lang.ClassCastException: com.lara.CollectionNotes.ListNotes.ArrayListNotes.A cannot be cast to java.lang.Comparable

at java.util.ComparableTimSort.countRunAndMakeAscending(ComparableTimSort.java:290)

at java.util.ComparableTimSort.sort(ComparableTimSort.java:157)

at java.util.ComparableTimSort.sort(ComparableTimSort.java:146)

at java.util.Arrays.sort(Arrays.java:472)

at java.util.Collections.sort(Collections.java:155)

at com.lara.CollectionNotes.ListNotes.ArrayListNotes.ArrayListNotes20.main(ArrayListNotes20.java:42)

Here we are getting ClassCasteException due to class A unable caste into Comparable interface types. So we are getting this exception. This class A is unable to caste into interface

Comparable kind.

\*/

Example:

package com.lara.CollectionNotes.ListNotes.ArrayListNotes;

import java.util.ArrayList;

import java.util.Collections;

@SuppressWarnings("rawtypes")

class B implements Comparable

{

int i;

B(int i)

{

this.i = i;

}

public String toString()

{

return "i: "+i;

}

public int compareTo(Object obj)

{

return this.i - ((B)obj).i;

}

}

@SuppressWarnings({"unchecked", "rawtypes"})

public class ArrayListNotes21

{

public static void main(String[] args)

{

ArrayList list = new ArrayList();

list.add(new B(90));

list.add(new B(0));

list.add(new B(9));

list.add(new B(910));

list.add(new B(220));

System.out.println("Before Sorting elements of an ArrayList list is: ");

System.out.println(list);

Collections.sort(list);

System.out.println("After Sorting elements of an ArrayList list is: ");

System.out.println(list);

}

}

/\*

Here we have implements Comparable interface in class B and also used override method compareTo() for compare elements.

Here is the output of this program:

Before Sorting elements of an ArrayList list is:

[i: 90, i: 0, i: 9, i: 910, i: 220]

After Sorting elements of an ArrayList list is:

[i: 0, i: 9, i: 90, i: 220, i: 910]

From this output we got to know that its calling class B compareTo() for checking which element is small based on that its sorting. So here from this program we are getting proper output.

\*/

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Comparator;

**class** Book

{

**int** pages;

String title;

Double weight;

Book(**int** pages, String title, **double** weight)

{

**this**.pages = pages;

**this**.title = title;

**this**.weight = weight;

}

**public** String toString()

{

**return** pages+", "+title+", "+weight;

}

}

@SuppressWarnings("rawtypes")

**class** SortBasedOnPages **implements** Comparator

{

**public** **int** compare(Object obj1, Object obj2)

{

Book b1 = (Book)obj1;

Book b2 = (Book)obj2;

**return** b1.pages - b2.pages;

}

}

@SuppressWarnings("rawtypes")

**class** SortBasedOnTitle **implements** Comparator

{

**public** **int** compare(Object obj1, Object obj2)

{

Book b1 = (Book)obj1;

Book b2 = (Book)obj2;

**return** b1.title.compareTo(b2.title);

}

}

@SuppressWarnings("rawtypes")

**class** SortBasedOnWeight **implements** Comparator

{

**public** **int** compare(Object obj1, Object obj2)

{

Book b1 = (Book)obj1;

Book b2 = (Book)obj2;

**return** b1.weight.compareTo(b2.weight);

}

}

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes22

{

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

{

Book b1 = **new** Book(120, "Core Java", 200.30);

Book b2 = **new** Book(500, "Spring Frameworks", 300.30);

Book b3 = **new** Book(200, "Servlet Programming", 200.20);

Book b4 = **new** Book(400, "Hibernate", 400.40);

ArrayList list = **new** ArrayList();

list.add(b1);

list.add(b2);

list.add(b3);

list.add(b4);

System.*out*.println("Before Comparing elements in list object: ");

System.*out*.println(list);

System.*out*.println("Sorting Based on Pages of Book class in list:");

Collections.*sort*(list, **new** SortBasedOnPages());

System.*out*.println(list);

System.*out*.println("Sorting Based on Title of Book class in list: ");

Collections.*sort*(list, **new** SortBasedOnTitle());

System.*out*.println(list);

System.*out*.println("Sorting Based on Weight of Book Class in list: ");

Collections.*sort*(list, **new** SortBasedOnWeight());

System.*out*.println(list);

}

}

/\*

Here in this program we are sorting elements of Book class objects using compare() method of Comparator interface inside this class Book by implementing this interface inside this class.

Here we are sorting an elements based on their attributes each not all attributes at a times.

Inside class Book we have declared three attributes i.e pages, title and weight. So we are sorting this attributes based on their values.

Here is the output of this program:

Before Comparing elements in list object:

[120, Core Java, 200.3, 500, Spring Frameworks, 300.3, 200, Servlet Programming, 200.2, 400, Hibernate, 400.4]

Sorting Based on Pages of Book class in list:

[120, Core Java, 200.3, 200, Servlet Programming, 200.2, 400, Hibernate, 400.4, 500, Spring Frameworks, 300.3]

Sorting Based on Title of Book class in list:

[120, Core Java, 200.3, 400, Hibernate, 400.4, 200, Servlet Programming, 200.2, 500, Spring Frameworks, 300.3]

Sorting Based on Weight of Book Class in list:

[200, Servlet Programming, 200.2, 120, Core Java, 200.3, 500, Spring Frameworks, 300.3, 400, Hibernate, 400.4]

\*/

**How to Search any elements inside an ArrayList?**

Using binarySearch() method of Collections class we can search elements in an ArrayList. Here this method will return index position of these search elements.

Syntax:

Collections.binarySearch(“elements\_name”);

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

**import** java.util.Collections;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes23

{

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

{

ArrayList list = **new** ArrayList();

list.add(90);

list.add(100);

list.add(2);

list.add(20);

list.add(1);

list.add(9);

System.*out*.println("Before sorting elements of an ArrayList: ");

System.*out*.println(list);

Collections.*sort*(list);

**int** i = Collections.*binarySearch*(list, 100);

System.*out*.println("Index position of searching elements in an ArrayList: "+i);

}

}

/\*

Here we are searching for an element inside an ArrayList. Using binarySearch() method we are checking first whether that element is there inside this ArrayList, if there then it will return

index position of this elements inside this ArrayList.

Here is the output of this program:

Before sorting elements of an ArrayList:

[90, 100, 2, 20, 1, 9]

Index position of searching elements in an ArrayList: 5

\*/

**How to sorting an elements of an ArrayList with reverse way or sorting an elements in descending order?**

Using Collections.reverse() method we can reverse the order of an elements of an ArrayList. Here in ArrayList we are arranging an elements in descending order.

Syntax:

Collections.reverse(list);

Here list the references of any ArrayList.

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

**import** java.util.Collections;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes24

{

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

{

ArrayList list = **new** ArrayList();

list.add(900);

list.add(90);

list.add(9);

list.add(100);

list.add(1);

System.*out*.println("Before sorting an elements of an ArrayList: ");

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

Collections.*sort*(list, Collections.*reverseOrder*());

System.*out*.println("After sorting an elements in reverse order of an ArrayList: ");

System.*out*.println(list);

}

}

/\*

Here in this program we are trying to sorting elements of an ArrayList in descending order. Using this reverse() method of Collections class.

Here is the output of this program:

Before sorting an elements of an ArrayList:

[900, 90, 9, 100, 1]

After sorting an elements in reverse order of an ArrayList:

[900, 100, 90, 9, 1]

\*/

**How to remove duplicate elements from an ArrayList while adding elements into this ArrayList?**

Using contain() method of ArrayList method we can check whether that element is available inside the list or not.

If we are passing this checkin during element adding then we can remove duplicate element during adding elements into list and remove duplicate elements from list.

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

**import** java.util.Scanner;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes225

{

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

{

ArrayList list = **new** ArrayList();

Scanner sc = **new** Scanner(System.*in*);

String element = **null**;

String decider = **null**;

**do**

{

System.*out*.println("Please enter elements into an ArrayList and check whether is this element available or not:");

element = sc.next();

**if** (!list.contains(element))

{

list.add(element);

System.*out*.println("Element Added Success.");

}

**else**

System.*out*.println("Duplicate elements unable to add");

System.*out*.println("Do you want to continue? Y/N");

decider = sc.next();

} **while** ("Y".equalsIgnoreCase(decider));

System.*out*.println(list);

}

}

/\*

Here in this program we are trying to remove duplicate elements from an ArrayList during adding elements into this list even we are checking duplicate value using contain() method.

Using contain() method we are checking whether that element is available inside this list or not. If this element is available then it will return true. If not available then it will add that

element into this list.

Here is the output of this program:

Please enter elements into an ArrayList and check whether is this element available or not:

5

Element Added Success.

Do you want to continue? Y/N

y

Please enter elements into an ArrayList and check whether is this element available or not:

10

Element Added Success.

Do you want to continue? Y/N

y

Please enter elements into an ArrayList and check whether is this element available or not:

6

Element Added Success.

Do you want to continue? Y/N

y

Please enter elements into an ArrayList and check whether is this element available or not:

10

Duplicate elements unable to add

Do you want to continue? Y/N

y

Please enter elements into an ArrayList and check whether is this element available or not:

5

Duplicate elements unable to add

Do you want to continue? Y/N

y

Please enter elements into an ArrayList and check whether is this element available or not:

6

Duplicate elements unable to add

Do you want to continue? Y/N

y

Please enter elements into an ArrayList and check whether is this element available or not:

8

Element Added Success.

Do you want to continue? Y/N

y

Please enter elements into an ArrayList and check whether is this element available or not:

9

Element Added Success.

Do you want to continue? Y/N

y

Please enter elements into an ArrayList and check whether is this element available or not:

9

Duplicate elements unable to add

Do you want to continue? Y/N

y

Please enter elements into an ArrayList and check whether is this element available or not:

8

Duplicate elements unable to add

Do you want to continue? Y/N

n

[5, 10, 6, 8, 9]

\*/

**How to find min and max value inside an ArrayList?**

Using min(list) and max(list) we can find the min and max value of an ArrayList.

Syntax:

Collections.min(list);

Collections.max(list);

Using these two methods we can find min and max element inside an ArrayList.

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

**import** java.util.Collections;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ArrayListNotes26

{

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

{

ArrayList list = **new** ArrayList();

list.add(100);

list.add(10);

list.add(50);

list.add(1);

list.add(200);

list.add(30);

list.add(5);

list.add(400);

**int** min = (Integer)Collections.*min*(list);

**int** max = (Integer)Collections.*max*(list);

System.*out*.println("Min Value of ArrayList: "+min);

System.*out*.println("Max Value of ArrayList: "+max);

}

}

/\*

In this program we are fetching min and max value from an ArrayList. While we are using min() and max() method of Collections class then in that case we are passing that ArrayList

reference inside this method as an parameter then it will return me corresponding min and max value.

Here is the output of this program:

Min Value of ArrayList: 2

Max Value of ArrayList: 200

\*/

Iterator:

This is an interface in Java API. This is used for looping only Collection objects. This interface is used for looping Collection objects only one time not more than one time like normal for loop or enhanced for loop. Once one Collection object got iterate using this iterator then after that we can’t iterate after that same object.

This interface contains only three methods:

1. public abstract boolean hashNext();
2. public abstract Object next();
3. public abstract void remove();

From this Iterator interface List Iterator is inherited, means List Iterator is a child class of List Iterator.

Through iterator we can iterate Collection objects only one time not more than one time. If we want to iterate this loop more than one times then we have to create iterate object same number of time which we required. After getting these objects we can then iterate this loop as many times. But using one object we can’t iterate more than one time.

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

**import** java.util.Iterator;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** IteratorNotes1

{

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

{

ArrayList list = **new** ArrayList();

list.add(90);

list.add(190);

list.add(910);

list.add(920);

list.add(9);

Iterator it = list.iterator();

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

{

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

}

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

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

{

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

}

}

}

/\*

In this program we are trying to loop list value without usig for loop or advanced for loop. Here we are using Iterator which is an interface, through which we are going to loop this list

objects.

Here is the output of this program:

90

190

910

920

9

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Here we can iterate this list only one times not more than one time like normal for loop or enhanced for loop. Once one list object got iterate means it will not iterate again after that. That’s

why this output is coming like this.

\*/

Before creation of Iteration reference we can add or modify elements inside list objects but after creation of Iteration reference we can’t add or modify elements inside list or in between also we can’t modify or add also.

After completion of Iteration process we can add or modify elements into an ArrayList.

**Uses of remove():**

We can remove element during iteration. While we are running iteration process using while loop then in that case if we mentioned remove() method inside this loop then it will remove elements from an ArrayList. If we want to remove specific elements from an ArrayList then we can also specify this element.

Points about Iterator:

1. Iterator is pointing to ArrayList elements. It is locking the object while it will start iteration process using object lock due to reason that no one can modify anything during this iteration. This pointer is moving using the help of hasNext() method which is moving pointer from left to right direction.
2. This Iterator interface inside this java.util.\* package.
3. Using this iteration process we can do two things one is reading and another is removing elements.
4. Iteration process can’t be move back due to its already reached to the last position.

**List Iterator:**

This is also an interface which is working similar to Iterator interface but only one difference is there is List Iterator can iterate forward as well as backward direction also, but Iterator can only iterate list objects only on forward direction.

List Iterator is inherited from Iterator Interface.

There are so many method is available apart from Iterator methods.

Those methods are:

1. public abstract boolean hasNext();
2. public abstract Object next();
3. public abstract void remove();
4. public abstract void add(Object obj);
5. public abstract void set(Object obj);
6. public abstract boolean hasPrevious();
7. public abstract Object previous();
8. public abstract int nextIndex();
9. public abstract int previousIndex();

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

**import** java.util.ListIterator;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ListIteratorNotes1

{

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

{

ArrayList list = **new** ArrayList();

list.add(100);

list.add(200);

list.add(20);

list.add(0);

ListIterator ls = list.listIterator();

**while** (ls.hasNext())

{

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

}

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

**while** (ls.hasPrevious())

{

System.*out*.println(ls.previous());

}

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

}

}

/\*

In this program we are using ListIterator Interface for iterating collection objects. Here we have declared an ArrayList where we are iterating loop using listIterator() method with the help

of hashNext() method and next() method for forward direction also, with same time after forward direction we are also iterating list object in backward direction with the help of

hasPrevious() method and previous() method.

Here is the output of this program:

100

200

20

0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0

20

200

100

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

**import** java.util.ListIterator;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ListIteratorRemove

{

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

{

ArrayList list = **new** ArrayList();

list.add(100);

list.add(200);

list.add(20);

list.add(0);

System.*out*.println("Before removing element from an ArrayList using ListIterator: ");

System.*out*.println(list);

ListIterator ls = list.listIterator();

**while** (ls.hasNext())

{

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

ls.remove();

}

System.*out*.println("After Removing element from an ArrayList using ListIterator: ");

System.*out*.println(list);

}

}

/\*

In this program we are checking before removing elements from an ArrayList and after removing elements from an ArrayList using ListIterator remove() method.

Here is the output of this program:

Before removing element from an ArrayList using ListIterator:

[100, 200, 20, 0]

100

200

20

0

After Removing element from an ArrayList using ListIterator:

[]

\*/

**How to add elements during iteration process of list?**

Using add() method of ListIterator we can add elements during iteration process. But in case of Iterator interface there is no process such like this. In Iterator interface we can first add elements into ArrayList and then iteration process can happens but during iteration process we can’t add elements into list.

Syntax:

ls.add(Object obj);

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.ArrayList;

**import** java.util.ListIterator;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** ListIteratorAdd

{

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

{

ArrayList list = **new** ArrayList();

list.add(100);

list.add(200);

list.add(300);

System.*out*.println("Before iteration elements inside an ArrayList: ");

System.*out*.println(list);

ListIterator ls = list.listIterator();

**while** (ls.hasNext())

{

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

ls.add(400);

}

System.*out*.println("After iteration process and after adding elements during iteration process elements inside ArrayList: ");

System.*out*.println(list);

}

}

/\*

In this program we are using ListIterator and also we are trying here to add elements inside list during iteration process. As of we know that once iteration process start then it will lock the

list using object lock so that we can't add elements during iteration process in case of Iterator.

Here is the output of this program:

Before iteration elements inside an ArrayList:

[100, 200, 300]

100

200

300

After iteration process and after adding elements during iteration process elements inside ArrayList:

[100, 400, 200, 400, 300, 400]

\*/

**PriorityQueue:**

**poll()**: While we are using this method in case of Collection then its functionality is that it will remove first element from the list.

**pollFirst()**: This method is also working like poll() method. But pollFirst() method is bound to remove first element from the ArrayList.

**pollLast()**: This method is used to remove last element from an ArrayList.

**peek()**: This method is used for getting peek value from a list.

**peekFirst()**: This method will return head of the queue or first element of list.

**peekLast()**: This method will return last element of the list.

**pop()**: This method return the top most element added lastly in the list and also remove that element from the list.

**How we can create Queue using LinkedList?**

Example:

**package** com.lara.CollectionNotes.ListNotes.ArrayListNotes;

**import** java.util.LinkedList;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** LinkedListNotes

{

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

{

LinkedList queue = **new** LinkedList();

queue.add(90);

queue.add(9);

queue.add(0);

queue.add(100);

System.*out*.println(queue);

System.*out*.println("Using poll() method in list: "+queue.poll());

System.*out*.println("Elements in list after poll() method call: "+queue);

System.*out*.println("Using pollFirst() method in list: "+queue.pollFirst());

System.*out*.println("Elements in list after pollFirst() method call: "+queue);

System.*out*.println("Using pollLast() method in List"+queue.pollLast());

System.*out*.println("Elements in list after pollLast() method call: "+queue);

queue.add(90);

queue.add(9);

queue.add(0);

queue.add(100);

System.*out*.println("Using peek() method in list: "+queue.peek());

System.*out*.println("Elements in list after peek() method call: "+queue);

System.*out*.println("Using peekFirst() method in list: "+queue.peekFirst());

System.*out*.println("Elements in list after peekFirst() method call: "+queue);

System.*out*.println("Using peekLast() method in list: "+queue.peekLast());

System.*out*.println("Elements in list after peekLast() method call: "+queue);

queue.add(90);

queue.add(9);

queue.add(0);

queue.add(100);

System.*out*.println("Using pop() method in list: "+queue.pop());

System.*out*.println("Elements in list after pop() method call: "+queue);

}

}

/\*

In this program we are using so many method like poll(), pollFirst(), pollLast(), peek(), peekFirst(), peekLast() and pop() method for different different reason for getting value. Here below

we have explain about this after output of this program:

Here is the output of this program:

[90, 9, 0, 100]

Using poll() method in list: 90

Elements in list after poll() method call: [9, 0, 100]

Using pollFirst() method in list: 9

Elements in list after pollFirst() method call: [0, 100]

Using pollLast() method in List100

Elements in list after pollLast() method call: [0]

Using peek() method in list: 0

Elements in list after peek() method call: [0]

Using peekFirst() method in list: 0

Elements in list after peekFirst() method call: [0]

Using peekLast() method in list: 0

Elements in list after peekLast() method call: [0]

Using pop() method in list: 0

Elements in list after pop() method call: []

poll(): While we are using this method in case of Collection then its functionality is that it will remove first element from the list.

pollFirst(): This method is also working like poll() method. But pollFirst() method is bound to remove first element from the ArrayList.

pollLast(): This method is used to remove last element from an ArrayList.

peek(): This method is used for getting peek value from a list.

peekFirst(): This method will return head of the queue or first element of list.

peekLast(): This method will return last element of the list.

pop(): This method return the top most element added lastly in the list and also remove that element from the list.

\*/

* In Priority Queue automatically sorting order facility is available or we can tell auto sort mechanism is available inside Priority Queue.
* In PriorityQueue we can’t add different types of object into its list, if we are trying to add these objects then we will get Compile Time Error i.e **ClassCastException**.
* PriorityQueue won’t allow null value as elements, if we are trying to add this null value as an element then we will get CTE i.e **NullPointerException**.
* From this point we got to know that poll() method remove elements from the top of the list where peek() method will return the top elements value instead of remove that elements.
* This PriorityQueue class added in JDK 1.5 onwards.

Example:

**package** com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

**import** java.util.PriorityQueue;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** QueueNotes1

{

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

{

PriorityQueue queue = **new** PriorityQueue();

queue.add(110);

queue.add(10);

queue.add(210);

queue.add(90);

queue.add(10);

queue.add(0);

System.*out*.println(queue);

System.*out*.println(queue.poll());

System.*out*.println("After calling first poll() method list is : ");

System.*out*.println(queue);

System.*out*.println(queue.poll());

System.*out*.println("After second poll() method list is: ");

System.*out*.println(queue);

System.*out*.println(queue.poll());

System.*out*.println("After third poll() method list is: ");

System.*out*.println(queue);

}

}

/\*

In Priority Queue automatically sorting order facility is available or we can tell auto sort mechanism is available inside Priority Queue.

In this program we are checking list after calling poll() method three times.

Here is the output of this program:

[0, 10, 10, 110, 90, 210]

0

After calling first poll() method list is :

[10, 90, 10, 110, 210]

10

After second poll() method list is:

[10, 90, 210, 110]

10

After third poll() method list is:

[90, 110, 210]

\*/

Example:

**package** com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

**import** java.util.PriorityQueue;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** QueueNotes2

{

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

{

PriorityQueue queue = **new** PriorityQueue();

queue.add("abc");

queue.add("pqr");

queue.add("ijk");

queue.add("lmn");

queue.add("adf");

queue.add("cdf");

queue.add("def");

queue.add("rst");

queue.add("uvw");

System.*out*.println("Before calling poll() method list is:");

System.*out*.println(queue);

System.*out*.println(queue.poll());

System.*out*.println("After calling poll() method list element is: ");

System.*out*.println(queue);

System.*out*.println(queue.poll());

System.*out*.println("After calling poll() method list element is: ");

System.*out*.println(queue);

System.*out*.println(queue.poll());

System.*out*.println("After calling poll() method list element is: ");

System.*out*.println(queue);

}

}

/\*

In this program we are trying to remove first elements from the list of PriorityQueue object using poll() method. So before calling poll() method we are first printing the value inside queue

references and after calling poll() method we are printing list inside queue reference.

Here is the output of this program:

Before calling poll() method list is:

[abc, adf, cdf, pqr, lmn, ijk, def, rst, uvw]

abc

After calling poll() method list element is:

[adf, lmn, cdf, pqr, uvw, ijk, def, rst]

adf

After calling poll() method list element is:

[cdf, lmn, def, pqr, uvw, ijk, rst]

cdf

After calling poll() method list element is:

[def, lmn, ijk, pqr, uvw, rst]

\*/

Example:

**package** com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

**import** java.util.PriorityQueue;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** QueueNotes3

{

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

{

PriorityQueue list = **new** PriorityQueue();

list.add(100);

list.add(20.35);

list.add("Sailendra");

list.add(120.35f);

list.add(**true**);

list.add(9599480640l);

System.*out*.println(list);

}

}

/\*

In this program we are created an object of PriorityQueue and we are trying to add some different kind or types of value inside this Queue list object using add() method.

Here is the output of this program:

Exception in thread "main" java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.Double

at java.lang.Double.compareTo(Double.java:49)

at java.util.PriorityQueue.siftUpComparable(PriorityQueue.java:637)

at java.util.PriorityQueue.siftUp(PriorityQueue.java:629)

at java.util.PriorityQueue.offer(PriorityQueue.java:329)

at java.util.PriorityQueue.add(PriorityQueue.java:306)

at com.lara.CollectionNotes.ListNotes.ArrayListNotes.QueueNotes3.main(QueueNotes3.java:13)

From this output we got to know that PriorityQueue is not allowing different kind of data to be store inside its list. So for this reason its throwing ClassCasteException.

\*/

Example:

**package** com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

**import** java.util.PriorityQueue;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** QueueNotes4

{

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

{

PriorityQueue list = **new** PriorityQueue();

list.add(**null**);

list.add(200);

System.*out*.println(list);

}

}

/\*

In this program we are trying to check whether this PriorityQueue is adding null value or not.

Here is the output of this program:

Exception in thread "main" java.lang.NullPointerException

at java.util.PriorityQueue.offer(PriorityQueue.java:320)

at java.util.PriorityQueue.add(PriorityQueue.java:306)

at com.lara.CollectionNotes.ListNotes.ArrayListNotes.QueueNotes4.main(QueueNotes4.java:12)

From this output we got to know that PriorityQueue won't allow null value as an elements.

\*/

Example:

**package** com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

**import** java.util.PriorityQueue;

**class** A1

{

**int** i;

A1(**int** i)

{

}

}

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** QueueNotes5

{

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

{

PriorityQueue que = **new** PriorityQueue();

que.add(**new** A1(90));

que.add(**new** A1(0));

que.add(**new** A1(9));

que.add(**new** A1(910));

que.add(**new** A1(190));

System.*out*.println(que);

}

}

/\*

In this program we are trying to check the list added inside this que of PriorityQueue object reference.

Here is the output of this program:

Exception in thread "main" java.lang.ClassCastException: com.lara.CollectionNotes.ListNotes.ArrayListNotes.A1 cannot be cast to java.lang.Comparable

at java.util.PriorityQueue.siftUpComparable(PriorityQueue.java:633)

at java.util.PriorityQueue.siftUp(PriorityQueue.java:629)

at java.util.PriorityQueue.offer(PriorityQueue.java:329)

at java.util.PriorityQueue.add(PriorityQueue.java:306)

at com.lara.CollectionNotes.ListNotes.ArrayListNotes.QueueNotes5.main(QueueNotes5.java:22)

From this output we got to know that class A1 is not types of Comparable class types or not implements this interface. So for this reason its throwing ClassCastException.

\*/

Example:

**package** com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

**import** java.util.PriorityQueue;

**class** A2 **implements** Comparable

{

**int** i;

A2(**int** i)

{

**this**.i = i;

}

**public** String toString()

{

**return** "i: "+i;

}

**public** **int** compareTo(Object obj)

{

**return** **this**.i - ((A2)obj).i;

}

}

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** QueueNotes6

{

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

{

PriorityQueue que = **new** PriorityQueue();

que.add(**new** A2(90));

que.add(**new** A2(9));

que.add(**new** A2(0));

que.add(**new** A2(190));

System.*out*.println(que);

System.*out*.println("First poll() method calling: "+que.poll());

System.*out*.println("Second poll() method calling: "+que.poll());

System.*out*.println("Third poll() method calling: "+que.poll());

System.*out*.println("Forth poll() method calling: "+que.poll());

System.*out*.println(que);

}

}

/\*

In this program we are going to check whether its calling class A2 compareTo() method or not. Because this class implements Comparable interface to compare between class A2 object

references.

After this we are calling poll() method using this PriorityQueue object reference.

Here is the output of this class:

[i: 0, i: 90, i: 9, i: 190]

First poll() method calling: i: 0

Second poll() method calling: i: 9

Third poll() method calling: i: 90

Forth poll() method calling: i: 190

[]

As we know that poll() method is using for removing elements from head side of the list. How many times we will call this poll() method that time it will remove that elements from that list.

So here we are getting same result as an output.

\*/

Example:

package com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

import java.util.Comparator;

import java.util.PriorityQueue;

class Person1

{

int age;

String firstName;

String lastName;

long mobNo;

Person1(int age, String firstName, String lastName, long mobNo)

{

this.age = age;

this.firstName = firstName;

this.lastName = lastName;

this.mobNo = mobNo;

}

public String toString()

{

return "Age: "+age+", First Name: "+firstName+", Last Name: "+lastName+", Mobile Number: "+mobNo;

}

}

@SuppressWarnings("rawtypes")

class SortBasedOnAge implements Comparator

{

public int compare(Object obj1, Object obj2)

{

Person1 per1 = (Person1)obj1;

Person1 per2 = (Person1)obj2;

return per1.age - per2.age;

}

}

@SuppressWarnings("rawtypes")

class SortBasedOnFirstName1 implements Comparator

{

public int compare(Object obj1, Object obj2)

{

Person1 per1 = (Person1)obj1;

Person1 per2 =(Person1)obj2;

return per1.firstName.compareTo(per2.firstName);

}

}

@SuppressWarnings("rawtypes")

class SortBasedOnLastName1 implements Comparator

{

public int compare(Object obj1, Object obj2)

{

Person1 per1 =(Person1)obj1;

Person1 per2 = (Person1)obj2;

return per1.lastName.compareTo(per2.lastName);

}

}

@SuppressWarnings("rawtypes")

class SortBasedOnMobNo1 implements Comparator

{

public int compare(Object obj1, Object obj2)

{

Person1 per1 = (Person1)obj1;

Person1 per2 = (Person1)obj2;

return (int) (per1.mobNo - per2.mobNo);

}

}

@SuppressWarnings({"unchecked", "rawtypes"})

public class QueueNotes7

{

public static void main(String[] args)

{

PriorityQueue que = new PriorityQueue();

que.add(new Person1(32, "Sailendra", "Jena", 9599480640l));

que.add(new Person1(28, "Bhagyalaxmi", "Jena", 9971359955l));

que.add(new Person1(5, "Samir", "Jena", 9731728407l));

que.add(new Person1(35, "Siba", "Jena", 9437202484l));

System.out.println("Before calling poll() method list is:");

// System.out.println(que);

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

System.out.println("First poll() method is calling: "+que.poll());

System.out.println("Second poll() method is calling: "+que.poll());

System.out.println("Third poll() method is calling: "+que.poll());

System.out.println("Forth poll() method is calling: "+que.poll());

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

System.out.println("After calling poll() method list is: ");

// System.out.println(que);

}

}

Example:

**package** com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

**import** java.util.PriorityQueue;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** QueueNotes8

{

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

{

PriorityQueue que = **new** PriorityQueue();

que.add(90);

que.add(190);

que.add(920);

que.add(490);

que.add(290);

System.*out*.println("Before calling poll() method and peek() method list is: ");

System.*out*.println(que);

System.*out*.println("After calling poll() method: "+que.poll());

System.*out*.println("After calling peek() method: "+que.peek());

System.*out*.println("After calling poll() method: "+que.poll());

System.*out*.println("After calling peek() method: "+que.peek());

System.*out*.println("After calling poll() and peek() method list is: ");

System.*out*.println(que);

}

}

/\*

In this program we are first creating object of PriorityQueue class and we have added some int value into this and after that we are checking value after calling poll() method and peek()

method.

Here is the output of this program:

Before calling poll() method and peek() method list is:

[90, 190, 920, 490, 290]

After calling poll() method: 90

After calling peek() method: 190

After calling poll() method: 190

After calling peek() method: 290

After calling poll() and peek() method list is:

[290, 490, 920]

From this point we got to know that poll() method remove elements from the top of the list where peek() method will return the top elements value instead of remove that elements.

\*/

Example:

**package** com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

**public** **class** Person

{

**int** age;

String firstName;

String lastName;

**long** mobNo;

Person(**int** age, String firstName, String lastName, **long** mobNo)

{

**this**.age = age;

**this**.firstName = firstName;

**this**.lastName = lastName;

**this**.mobNo = mobNo;

}

**public** String toString()

{

**return** "Age: "+age+", First Name: "+firstName+", Last Name: "+lastName+", Mobile Number: "+mobNo;

}

}

**package** com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

**import** java.util.Comparator;

@SuppressWarnings("rawtypes")

**public** **class** SortBasedOnFirstName **implements** Comparator

{

**public** **int** compare(Object obj1, Object obj2)

{

Person per1 = (Person)obj1;

Person per2 = (Person)obj2;

**return** per1.firstName.compareTo(per2.firstName);

}

}

**package** com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

**import** java.util.Comparator;

@SuppressWarnings("rawtypes")

**public** **class** SortBasedOnLastName **implements** Comparator

{

**public** **int** compare(Object obj1, Object obj2)

{

Person per1 = (Person)obj1;

Person per2 = (Person)obj2;

**return** per1.lastName.compareTo(per2.lastName);

}

}

**package** com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

**import** java.util.Comparator;

@SuppressWarnings("rawtypes")

**public** **class** SortBasedOnMobNo **implements** Comparator

{

**public** **int** compare(Object obj1, Object obj2)

{

Person per1 = (Person)obj1;

Person per2 = (Person)obj2;

**return** (**int**) (per1.mobNo - per2.mobNo);

}

}

**package** com.lara.CollectionNotes.QueueNotes.PriorityQueueNotes;

**import** java.util.Comparator;

@SuppressWarnings("rawtypes")

**public** **class** SortOnAge **implements** Comparator

{

**public** **int** compare(Object obj1, Object obj2)

{

Person per1 = (Person)obj1;

Person per2 = (Person)obj2;

**return** per1.age - per2.age;

}

}

**Significance of Set:**

HashSet is the subclass of Set interface. Here Set is an Interface where we can’t create the object of this interface but we can use this interface reference and creating the object of HashSet class.

Here there are some few points about Set, HashSet and TreeSet class.

* Set, HashSet and TreeSet won’t allow duplicate value inside its list. If we will try to add this duplicate elements the also it will show one single elements not duplicate elements.
* add() method is returning boolean value or literal.
* We can’t read specific element from the list like ArrayList using index position.
* Set, TreeSet and HashSet doesn’t have indexing.

Example:

**package** com.lara.CollectionNotes.SetNotes.HashSetNotes;

**import** java.util.HashSet;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashSetNotes1

{

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

{

HashSet set = **new** HashSet();

set.add(120);

set.add(120);

set.add(120);

set.add(120);

set.add(120);

set.add(120);

System.*out*.println("Size of the Set: "+set.size());

System.*out*.println("Value of Set: "+set);

}

}

/\*

In this program we are trying to add duplicate value inside this HashSet object.

Here is the output of this program:

Size of the Set: 1

Value of Set: [120]

\*/

* Set, HashSet and TreeSet don’t have indexing. Means if we want to see any elements then we have to iterate this HashSet object elements or loop all these elements inside this HashSet.

Example:

**package** com.lara.CollectionNotes.SetNotes.HashSetNotes;

**import** java.util.HashSet;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashSetNotes2

{

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

{

HashSet set = **new** HashSet();

System.*out*.println(set.add(100));

System.*out*.println(set.add(100));

System.*out*.println(set.add(200));

System.*out*.println(set.add(200));

System.*out*.println("Size of the HashSet: "+set.size());

System.*out*.println("Value inside HashSet Object list: "+set);

}

}

/\*

In this program also we are trying to add different value with one time repetition of that value and we are checking whether those repeated values are adding into this HashSet object list or

not. As we know that in Set, HashSet and TreeSet won't allow duplicate value. So we are checking this by adding duplicate value into this.

Here is the output of this program:

true

false

true

false

Size of the HashSet: 2

Value inside HashSet Object list: [100, 200]

\*/

Example:

**package** com.lara.CollectionNotes.SetNotes.HashSetNotes;

**import** java.util.HashSet;

**import** java.util.Iterator;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashSetNotes3

{

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

{

HashSet set = **new** HashSet();

set.add(110);

set.add(120);

set.add(10);

set.add(130);

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

System.*out*.println("Using Iteration Process");

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

Iterator it = set.iterator();

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

{

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

}

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

System.*out*.println("Using Enhanced for loop");

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

**for** (Object obj : set)

{

System.*out*.println(obj);

}

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

}

}

/\*

In this program we have created HashSet object where we are added some elements into this object. After adding elements into this we are checking which way we can fetch the elements

and print those elements. Set can only iterate using Iteration process and Enhanced for loop process.

Here is the output of this program:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Using Iteration Process

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

110

10

130

120

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Using Enhanced for loop

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

110

10

130

120

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

From this output we got to know that HashSet value can only be able to fetch using Iteration process and Enhanced for loop process. Due to reason that Set, HashSet and TreeSet don't have

indexing process.

\*/

* HashSet allows null values but not allow duplicates.

Example:

**package** com.lara.CollectionNotes.SetNotes.HashSetNotes;

**import** java.util.HashSet;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashSetNotes4

{

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

{

HashSet set = **new** HashSet();

set.add("abc");

set.add("abc");

set.add("null");

set.add("xyz");

set.add(**null**);

System.*out*.println("Elements inside HashSet Objects list: ");

System.*out*.println(set);

}

}

/\*

In this program we are trying to check that in HashSet class is allow null value or not or it will throws NullPointerException.

Here is the output of this program:

Elements inside HashSet Objects list:

[null, abc, null, xyz]

From this output we got to know that HashSet allow null value inside its list.

\*/

Example:

**package** com.lara.CollectionNotes.SetNotes.HashSetNotes;

**import** java.util.HashSet;

**class** Z

{

**int** i;

Z(**int** i)

{

**this**.i = i;

}

**public** String toString()

{

**return** "i: "+i;

}

}

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashSetNotes5

{

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

{

HashSet set = **new** HashSet();

set.add("abc");

set.add("abc");

set.add(120);

set.add(120);

set.add(**new** Z(10));

set.add(**new** Z(10));

System.*out*.println("Value inside HashSet object list: "+set);

}

}

/\*

In this program we are trying different way to pass duplicate value inside this HashSet object reference and checking that whether its passing into this list or throwing exception.

Here we are also passing duplicate value using object of class Z constructor.

Here is the output of this program:

Value inside HashSet object list: [abc, i: 10, i: 10, 120]

From this output we got to know that using class Z object references we can add duplicate value.

Here in class Z hashCode() and equals() method not got override so while adding this elements then JVM understand both objects are different its not checking content inside this objects.

For this reason we can easily added elements into it.

\*/

Example:

**package** com.lara.CollectionNotes.SetNotes.HashSetNotes;

**import** java.util.HashSet;

**class** Y

{

**int** i;

Y(**int** i)

{

**this**.i = i;

}

**public** String toString()

{

**return** "i: "+i;

}

**public** **int** hashCode()

{

**return** Integer.*toString*(i).hashCode();

}

**public** **boolean** equals(Object obj)

{

Y y = (Y)obj;

**return** **this**.i == y.i;

}

}

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashSetNotes6

{

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

{

HashSet set = **new** HashSet();

set.add(**new** Y(90));

set.add(**new** Y(90));

set.add(**new** Y(90));

set.add(**new** Y(90));

set.add(**new** Y(90));

System.*out*.println("Size of HashSet object reference: "+set.size());

System.*out*.println("List inside this HashSet reference is : ");

System.*out*.println(set);

}

}

/\*

In this program we have override toString(), equals() and hashCode() method from Object class to class Y and we are passing duplicate value using class Y constructor and checking

whether JVM allow this duplicate value or not.

Here is the output of this program:

Size of HashSet object reference: 1

List inside this HashSet reference is :

[i: 90]

\*/

* In HashSet equals(), hashCode() and toString() method not got override automatically.
* So here we are override all these methods into this class Y and then checking for equality of object values.
* To add unique value into HashSet using another class object then we have to override these equals() and hashCode() method.

Example:

**package** com.lara.CollectionNotes.SetNotes.HashSetNotes;

**import** java.util.HashSet;

**class** Person

{

**int** age;

**double** weight;

Person(**int** age, **double** weight)

{

**this**.age = age;

**this**.weight = weight;

}

**public** String toString()

{

**return** "Age: "+age+", Weight: "+weight;

}

**public** **int** hashCode()

{

**int** hash = Integer.*toString*(age).hashCode();

hash+= Double.*toString*(weight).hashCode();

**return** hash;

}

**public** **boolean** equals(Object obj)

{

Person p = (Person)obj;

**return** (**this**.age == p.age) && (**this**.weight == p.weight);

}

}

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashSetNotes7

{

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

{

HashSet set = **new** HashSet();

set.add(100);

set.add(100.0);

set.add("abc");

set.add(**new** Person(20, 123.090));

set.add(**new** Person(20, 123.090));

set.add("abc");

set.add(100);

set.add(100.0);

System.*out*.println("Value inside HashSet is : ");

System.*out*.println(set);

}

}

/\*

In this program we are trying to add again duplicate value using Person class where this Person class override all these three equals(), hashCode() and toString() method. So we are checking

whether duplicate value is taking while we are adding all these elements into HashSet object reference or not.

Here is the output of this program:

Value inside HashSet is :

[abc, 100, Age: 20, Weight: 123.09, 100.0]

\*/

**TreeSet:**

**How to Sort HashSet or Set elements?**

Using TreeSet we can sort HashSet or Set elements. There is no default sorting mechanism available in HashSet or Set. Only one is process is there where we can sort these class elements.

Using TreeSet class sort( ) method is there through which we can sort elements of HashSet or Set.

Multiple datatypes cannot be able to sort it will throws ClassCasteException. It will sort between similar kinds of datatypes.

Example:

package com.lara.CollectionNotes.SetNotes.TreeSetNotes;

import java.util.HashSet;

import java.util.TreeSet;

@SuppressWarnings({"unchecked", "rawtypes"})

public class TreeSetNotes1

{

public static void main(String[] args)

{

HashSet set = new HashSet();

set.add(900);

set.add(90);

set.add(0);

set.add(100);

set.add(50);

System.out.println("Before sorting element of HashSet: ");

System.out.println(set);

System.out.println("After sorting elements of HashSet: ");

TreeSet treeset = new TreeSet(set);

System.out.println(treeset);

}

}

/\*

In this program we are sorting elements of HashSet using TreeSet class object constructor.

Here is the output of this program:

Before sorting element of HashSet:

[0, 50, 100, 900, 90]

After sorting elements of HashSet:

[0, 50, 90, 100, 900]

\*/

Example:

**package** com.lara.CollectionNotes.SetNotes.TreeSetNotes;

**import** java.util.ArrayList;

**import** java.util.HashSet;

**import** java.util.TreeSet;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** TreeSetNotes2

{

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

{

ArrayList list = **new** ArrayList();

list.add(900);

list.add(90);

list.add(0);

list.add(100);

list.add(20);

HashSet set = **new** HashSet(list);

System.*out*.println("While Testing on HashSet: "+set);

TreeSet treeset = **new** TreeSet(list);

System.*out*.println("While Testing on TreeSet: "+treeset);

}

}

/\*

In this program we are trying to pass ArrayList object reference inside this HashSet class constructor during creation of HashSet class object.

Here is the output of this program:

While we are passing this list object reference into TreeSet class constructor for sorting list elements. But while we are passing this list object reference into HashSet then its not sorting

elements inside list.

Here is the output of this both scenarios:

While Testing on HashSet: [0, 100, 20, 900, 90]

While Testing on TreeSet: [0, 20, 90, 100, 900]

\*/

Example:

**package** com.lara.CollectionNotes.SetNotes.TreeSetNotes;

**import** java.util.TreeSet;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** TreeSetNotes3

{

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

{

TreeSet set = **new** TreeSet();

set.add(90);

set.add(901);

set.add(90);

set.add(902);

set.add(9);

System.*out*.println("Elements in TreeSet list: ");

System.*out*.println(set);

}

}

/\*

In this program we are trying to check when we are creating object of TreeSet class then in that case we are getting sorting ordered elements or not.

Here is the output of this program:

Elements in TreeSet list:

[9, 90, 901, 902]

From this output we got to know that we are getting sorted ordered elements.

\*/

Map Stream:

HashMap:

HashMap is also part of Collection API. This class is the sub class of Map Interface. This class object is taking two values as parameter one is key and another is value. That’s the reason we are calling as Map is taking key and value pair elements. Key is the memory location of the value where it got stored in the object areas or heap memory. Here is the example of this sub class HashMap.

Using put(key, value) method we are inserting elements into HashMap object.

Syntax:

HashMap map = new HashMap();

map.put(“abc”, “abc”);

We can’t directly call and print then elements of HashMap we have to iterate HashMap key based on that we can get the values.

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.HashMap;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes1

{

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

{

HashMap map = **new** HashMap();

map.put("abc", 100);

map.put(200, 300);

map.put("xyz", "hello");

System.*out*.println("Elements inside map is: ");

System.*out*.println(map);

}

}

/\*

In this program we are creating a simple HashMap class object and adding some values into this.

Here is the output of this program:

Elements inside map is:

{abc=100, 200=300, xyz=hello}

Here its showing all key and values pair. As we discuss that if we want only values then we have to iterate key then it will fetch the value related to key.

\*/

* Inside HashMap put() is a method through which we are passing elements. Where in put() method 1st option is Key and 2nd option is element. i.e put(key, element).

**Updating Value of Key in HashMap:**

In HashMap we can update key values it will not allow duplicate value inside this list.

Here below is the example where we have shown this features.

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.HashMap;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes2

{

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

{

HashMap map = **new** HashMap();

map.put("abc", 1000);

System.*out*.println("Before update value of Key abc: ");

System.*out*.println(map);

map.put("abc", 2000);

System.*out*.println("After update value of key abc: ");

System.*out*.println(map);

map.put("abc", 3000);

System.*out*.println("After another update value of key abc: ");

System.*out*.println(map);

}

}

/\*

In this program we are trying to check after updating map key valye three time that whether that old value is still there or its changing and updating value without duplication.

Here is the output of this program:

Before update value of Key abc:

{abc=1000}

After update value of key abc:

{abc=2000}

After another update value of key abc:

{abc=3000}

Its changing values and also not duplication happening with this key.

\*/

* In HashMap there is no Duplication of value or element will allows. It will just update that values.

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.HashMap;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes3

{

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

{

HashMap map = **new** HashMap();

map.put("key1", "abc");

map.put("key1", 1000);

map.put("key1", 5000);

map.put("key1", "hello");

map.put("key1", "test");

map.put("key1", "Sailendra");

map.put("key1", 50);

map.put("key1", **true**);

System.*out*.println("Value in HashMap object reference: ");

System.*out*.println(map);

}

}

/\*

In this program also we are checking whether this HashMap object reference is allowing duplicate value or not.

Here is the output of this program:

Value in HashMap object reference:

{key1=true}

From this output we got to know that its won't allow duplicate elements its only update the latest value.

\*/

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.HashMap;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes4

{

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

{

HashMap map = **new** HashMap();

map.put(**null**, "abc");

map.put("key", "xyz");

map.put(**null**, "test");

System.*out*.println("Elements inside map list: ");

System.*out*.println(map);

}

}

/\*

In this program we are trying to check whether HashMap key is allow null value or not. In this program we are passing two time null value inside key and we are checking that this program

is running properly or not and also we are checking that this null value is taking two times or taking the latest value only like value cases or element cases.

Here is the output of this program:

Elements inside map list:

{null=test, key=xyz}

From this output we got to know that only latest value is taking also key and also key is allow null as value and also its avoiding duplicate value and taking latest one as value in key.

\*/

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.HashMap;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes5

{

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

{

HashMap map = **new** HashMap();

map.put(**null**, **null**);

map.put(**null**, **null**);

map.put(**null**, **null**);

map.put(**null**, **null**);

map.put(**null**, **null**);

map.put(**null**, "test");

System.*out*.println("Elements of map list is: ");

System.*out*.println(map);

}

}

/\*

In this program we are passing null values as key and element also. After that last value we have pass test as in case of elements that whether this both key and elements are taking null value

or not and also its taking latest value if still keeping old value as an key and elements.

Here is the output of this program:

Elements of map list is:

{null=test}

From this output we got to know that this key and elements value can be passing as null and also HashMap allow duplicate value without any error or exception but its taking the latest

value not allow to keep duplicate value inside list.

\*/

* HashMap key allow null value and also elements also allow null value but only one time means if we are adding more than one value for key and elements as null then also it will keep the latest value not the duplicate value.
* In HashMap key should be unique and key should not be duplicate.

HashTable:

HashTable is synchronized one or thread safed one where HashMap is non synchronized one.

In HashTable it will not allow null value as key and also for elements also. If we will pass then we will get NullPointerException.

Duplicate key or elements will not allow in list but while adding key and elements that time its allow this key and elements but while we are checking or printing key and value then it will shows latest values, it will not shows duplicate values.

Example:

**package** com.lara.CollectionNotes.MapNotes.HashTableNotes;

**import** java.util.Hashtable;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashTableNotes1

{

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

{

Hashtable table = **new** Hashtable();

table.put("abc1", "Hello1");

table.put("abc2", "Hello2");

table.put("abc3", "Hello3");

table.put("abc4", "Hello4");

table.put("abc5", "Hello5");

System.*out*.println("Elements inside HashTable is: ");

System.*out*.println(table);

}

}

/\*

In this program we are just created one HashTable class objects and pass some value and key in its object using put() method.

Here is the output of this program:

Elements inside HashTable is:

{abc5=Hello5, abc4=Hello4, abc3=Hello3, abc2=Hello2, abc1=Hello1}

\*/

Example:

**package** com.lara.CollectionNotes.MapNotes.HashTableNotes;

**import** java.util.Hashtable;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashTableNotes2

{

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

{

Hashtable table = **new** Hashtable();

table.put("abc1", "Hello1");

table.put("abc1", "Hello2");

table.put("abc1", "Hello3");

table.put("abc1", "Hello4");

table.put("abc1", "Hello5");

System.*out*.println("Elements in HashTable:");

System.*out*.println(table);

}

}

/\*

In this program we are trying to add multiple value to one key and checking that whether its taking all values or its only accepting latest value of all these values or it will keep only same

value as previously it was there.

Here is the output of this program:

Elements in HashTable:

{abc1=Hello5}

From this program we got to know that it is taking latest value of all these values.

\*/

**How to read elements from Map Stream?**

Here in Map Interface we can fetch particular element from list using key without iterating all elements of this list. Using get() method we can fetch element from the list where this method taking one parameter as key. When we will pass an argument inside this get() method then JVM will check that arguments available in key section if that argument is there then related that key JVM will fetch element form that list.

Syntax:

map.get(key);

Example:

**package** com.lara.CollectionNotes.MapNotes.HashTableNotes;

**import** java.util.Hashtable;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashTableNotes1

{

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

{

Hashtable table = **new** Hashtable();

table.put("abc1", "Hello1");

table.put("abc2", "Hello2");

table.put("abc3", "Hello3");

table.put("abc4", "Hello4");

table.put("abc5", "Hello5");

System.*out*.println("Elements inside HashTable is: ");

System.*out*.println(table);

}

}

/\*

In this program we are just created one HashTable class objects and pass some value and key in its object using put() method.

Here is the output of this program:

Elements inside HashTable is:

{abc5=Hello5, abc4=Hello4, abc3=Hello3, abc2=Hello2, abc1=Hello1}

\*/

Example:

**package** com.lara.CollectionNotes.MapNotes.HashTableNotes;

**import** java.util.Hashtable;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashTableNotes2

{

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

{

Hashtable table = **new** Hashtable();

table.put("abc1", "Hello1");

table.put("abc1", "Hello2");

table.put("abc1", "Hello3");

table.put("abc1", "Hello4");

table.put("abc1", "Hello5");

System.*out*.println("Elements in HashTable:");

System.*out*.println(table);

}

}

/\*

In this program we are trying to add multiple value to one key and checking that whether its taking all values or its only accepting latest value of all these values or it will keep only same

value as previously it was there.

Here is the output of this program:

Elements in HashTable:

{abc1=Hello5}

From this program we got to know that it is taking latest value of all these values.

\*/

Example:

**package** com.lara.CollectionNotes.MapNotes.HashTableNotes;

**import** java.util.HashMap;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashTableNotes3

{

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

{

HashMap map = **new** HashMap();

map.put("abc", "xyz");

map.put("key1", 200);

map.put(10, 1000);

System.*out*.println("Element of key 10 is: "+map.get(10));

System.*out*.println("Element of key abc is: "+map.get("abc"));

//int i = map.get("key1");

**int** value = (Integer)map.get("key1");

System.*out*.println("Key1 value is: "+value);

}

}

/\*

In this program we are trying to fetch particular element from the list using get() method of HashMap class.

Here is the output of this program:

Element of key 10 is: 1000

Element of key abc is: xyz

From this output we got to know that we can get the value of any element from this HashTable Object list using this get() method by passing key inside this method parameter.

If we want to retrieve the specific value and also store into related types then we have to type caste during fetching this element using get() method, if we will not use type caste related to

that types then we will get Compile Time Error due to mismatch type.

Here is the below example and also output of this condition:

int i = map.get("key1");

Here key1 value or element is an int type and we want to store this value into another int type variable but here we are using type caste so CTE is showing there. If we will use type caste then

it will return the actual value. Here we don;t have mentioned generic type so for this reason we can add different type datatypes.

Here is the output of this program:

Key1 value is: 200

From this output we got to know that if we want to fetch any key value and store into specific type variable then we have to use type caste for this then only it will return that value otherwise

it will return CTE.

\*/

**How to read all keys and elements/value from the HashMap without knowing the keys or values?**

Using only one process through which we can get the keys or elements from the HashMap list.

This process is Iteration process. Using this Iterator we can fetch either keys or values/elements.

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.HashMap;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes6

{

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

{

HashMap map = **new** HashMap();

map.put(120, "abc1");

map.put(120, "abc2");

map.put("key1", "hello1");

map.put("key1", "hello2");

System.*out*.println("Elements of map list: ");

System.*out*.println(map);

}

}

/\*

In this program we are trying to add duplicate element in same key and checking that whether its taking duplicate value and showing also or not in list.

Here is the output of this program:

Elements of map list:

{key1=hello2, 120=abc2}

From this output we got to know that its taking the duplicate value but its not showing in the list because its update that latest value with old value in the key.

\*/

* Key can be any object and elements also can be any object.
* Key cannot be duplicate but element can be duplicate.

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.HashMap;

**class** A

{

**int** i;

A(**int** i)

{

**this**.i = i;

}

**public** String toString()

{

**return** "i: "+i;

}

**public** **int** hashCode()

{

**return** Integer.*toString*(i).hashCode();

}

**public** **boolean** equals(Object obj)

{

A a = (A)obj;

**return** **this**.i == a.i;

}

}

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes7

{

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

{

HashMap map = **new** HashMap();

map.put(**new** A(90), 1000);

map.put(**new** A(90), 1001);

map.put(**new** A(90), 1002);

map.put(**new** A(90), 1003);

map.put(**new** A(90), 1004);

map.put(**new** A(90), 1005);

System.*out*.println("Elements in HashMap object map list: ");

System.*out*.println(map);

}

}

/\*

In this program we have created a class A where we have declared a variable int i and also override equals(), toString() and hashCode() method from Object class due to check for equality

of data.

In HahsMapNotes7 class we have created object of class A and with same key we are adding more than one different values/elements into this list and checking that whether its taking

these different values or not or it will override and taking latest value.

Here is the output of this program:

Elements in HashMap object map list:

{i: 90=1005}

From this output we got to know that its override this latest elements into key.

\*/

**How to Sort HashMap elements?**

For sorting elements of HashMap TreeMap class is there like TreeSet through which we can sort elements of HashMap list.

Syntax:

TreeMap map = new TreeMap(map1);

Here map1 is the object reference of HashMap class.

Using putAll() method of TreeMap class we can sorting elements of HashMap list.

But it will not sorting elements it will sort only key.

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.HashMap;

**import** java.util.TreeMap;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes8

{

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

{

HashMap map = **new** HashMap();

map.put("abc", 123);

map.put("xyz", 23);

map.put("hello", 13);

map.put("test", 3);

map.put("pqr", 12);

map.put("mine", 90);

System.*out*.println("Before Sorting elements into HashMap list:");

System.*out*.println(map);

TreeMap map1 = **new** TreeMap();

map1.putAll(map);

System.*out*.println("After Sorting an elements into HashMap list:");

System.*out*.println(map1);

}

}

/\*

In this program we are trying to sort an elements of HashMap list. Using putAll() method of TreeMap Class we can sort elements of HashMap class elements.

Here is the output of this program:

Before Sorting elements into HashMap list:

{hello=13, abc=123, test=3, pqr=12, mine=90, xyz=23}

After Sorting an elements into HashMap list:

{abc=123, hello=13, mine=90, pqr=12, test=3, xyz=23}

From this output we got to know that it will sort only key not elements. That means this sorting process is happening based on keys.

\*/

* Inside TreeMap class we have putAll() method is there through which we can sort key not elements of HashMap list.
* While sorting key of HashMap then we should remember one thing is that we have to keep key as same datatypes otherwise it will throws ClassCastException and if we will put null as key then it will throws NullPointerException.

**How to use compareTo() method in case of TreeMap?**

By using compareTo() method of Comparable interface we can

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.TreeMap;

@SuppressWarnings("rawtypes")

**class** B **implements** Comparable

{

**int** i;

B(**int** i)

{

**this**.i = i;

}

**public** String toString()

{

**return** "i: "+i;

}

**public** **int** compareTo(Object obj)

{

**return** **this**.i - ((B)obj).i;

}

**public** **int** hashCode()

{

**return** Integer.*toString*(i).hashCode();

}

**public** **boolean** equals(Object obj)

{

B b1 =(B)obj;

**return** **this**.i == b1.i;

}

}

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes9

{

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

{

TreeMap map1 = **new** TreeMap();

map1.put(**new** B(90), "abc");

map1.put(**new** B(9), "abc1");

map1.put(**new** B(0), "abc2");

map1.put(**new** B(190), "abc3");

map1.put(**new** B(910), "abc4");

map1.put(**new** B(20), "abc5");

map1.put(**new** B(20), "abc6");

System.*out*.println("Elements into HashMap list object: ");

System.*out*.println(map1);

}

}

/\*

In this program we are trying to add new different different elements into HashMap object list using class B object constructor and elements as values with different. Here we have added

repeated value in elements as well as key also.

In this program we are trying to check that whether its taking duplicate values or not and same time also we are checking equality of elements or values adding inside TreeMap and also we

are sorting elements based on keys.

In class B we have override toString(), equals() and hashCode() method for checking equality of objects.

Here is the output of this program:

Elements into HashMap list object:

{i: 0=abc2, i: 9=abc1, i: 20=abc6, i: 90=abc, i: 190=abc3, i: 910=abc4}

\*/

**How to use compare() method in TreeMap?**

By creating another class which will implements Comparator interface and also use its compare() method for checking equality of object’s content of another class.

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.Comparator;

**import** java.util.TreeMap;

**class** B1

{

**int** i;

B1(**int** i)

{

**this**.i = i;

}

**public** String toString()

{

**return** "i: "+i;

}

**public** **boolean** equals(Object obj)

{

B1 b1 = (B1)obj;

**return** **this**.i == b1.i;

}

**public** **int** hashCode()

{

**return** Integer.*toString*(i).hashCode();

}

}

@SuppressWarnings("rawtypes")

**class** Test **implements** Comparator

{

**public** **int** compare(Object obj1, Object obj2)

{

B1 b1 = (B1)obj1;

B1 b2 = (B1)obj2;

**return** b1.i - b2.i;

}

}

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes10

{

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

{

TreeMap map = **new** TreeMap(**new** Test());

map.put(**new** B1(90), "abc");

map.put(**new** B1(9), "abc1");

map.put(**new** B1(0), "abc2");

map.put(**new** B1(190), "abc3");

map.put(**new** B1(910), "abc4");

map.put(**new** B1(20), "abc5");

map.put(**new** B1(20), "abc6");

System.*out*.println("Elements in TreeMap is : ");

System.*out*.println(map);

}

}

/\*

In this program we are checking the equality of objects of class B1 and also we are checking that whether this TreeMap object is accepting duplicate object or not.

In this program we are using compare() method of Comparator interface.

In this program we have not implements Comparator interface inside class B1 its using in another class Test where we have use this compare() method and there we are checking these

class B1 objects for equality. This class Test object we are passing inside TreeMap constructor.

Here is the output of this program:

Elements in TreeMap is :

{i: 0=abc2, i: 9=abc1, i: 20=abc6, i: 90=abc, i: 190=abc3, i: 910=abc4}

From this output we got that its comparing and also checking equality between class B1 object reference.

\*/

How to Iterate HashMap list objects using Set interface and Entry inner class?

Inside Map interface one inner class is there which return key as well as values of Set interface.

Here in this program we are doing same thing using Iterator for iterating purpose of list of HashMap class object reference.

* One key value pair called one entry.
* Entry is an inner interface of interface Map or we can call this as sub interface.
* We can’t create an object of an interface even also using inner interface also we can’t create an object an interface.
* Here through Entry interface we can get the value and key value of an interface.

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.Map;

**import** java.util.Set;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes11

{

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

{

HashMap map = **new** HashMap();

map.put("abc", "xyz");

map.put("test1", "abc1");

map.put("test2", "test5");

map.put("test3", "hello");

Set entries = map.entrySet();

Iterator it = entries.iterator();

Map.Entry entry = **null**;

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

{

entry = (Map.Entry)it.next();

System.*out*.println("Key: "+entry.getKey()+" && "+"Element: "+entry.getValue());

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

}

}

}

/\*

Using Set interface and inner interface Entry of Map interface we are trying to iterate the list of HashMap class object values list. Here we are using Iterator for iterate list objects.

In this class we are trying to explain that how we are using Entry class for getting keys and related values.

Here is the output of this program:

Key: test1 && Element: abc1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Key: abc && Element: xyz

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Key: test2 && Element: test5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Key: test3 && Element: hello

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

**What are the method’s available in Navigable Set?**

* E lower(E e);
* E floor(E e);
* E ceiling(E e);
* E higher(E e);
* E pollFirst();
* E pollLast();
* Iterator<E> iterator();
* NavigableSet<E> descendingSet();
* Iterator<E> descendingIterator();
* NavigableSet<E> subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive);
* NavigableSet<E> headSet(E toElement, boolean inclusive);
* NavigableSet<E> tailSet(E fromElement, boolean inclusive);
* SortedSet<E> subSet(E fromElement, E toElement);
* SortedSet<E> headSet(E toElement);
* SortedSet<E> tailSet(E fromElement);

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.TreeSet;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes12

{

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

{

TreeSet set = **new** TreeSet();

set.add(90);

set.add(0);

set.add(9);

set.add(5);

set.add(3);

set.add(15);

System.*out*.println("Elements in set before calling ceilling method: ");

System.*out*.println(set);

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

System.*out*.println("Elements in set After calling ceilling method: ");

System.*out*.println(set.ceiling(9));

System.*out*.println(set.ceiling(10));

System.*out*.println(set.ceiling(50));

System.*out*.println(set);

}

}

/\*

In this program we are checking what exactly this ceiling method is doing while we are passing any arguments inside this method and calling this method with TreeSet list. Actually this

method is fetching the element which is greater or equal to this value which we are passing in ceiling() method.

Here is the different example we have added below:

Elements in set before calling ceilling method:

[0, 3, 5, 9, 15, 90]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elements in set After calling ceilling method:

9

15

90

[0, 3, 5, 9, 15, 90]

\*/

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.TreeSet;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes13

{

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

{

TreeSet set = **new** TreeSet();

set.add(90);

set.add(0);

set.add(9);

set.add(5);

set.add(3);

set.add(15);

System.*out*.println("Elements in set before floor method: ");

System.*out*.println(set);

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

System.*out*.println("Elements in set After floor method: ");

System.*out*.println(set.floor(9));

System.*out*.println(set.floor(10));

System.*out*.println(set.floor(50));

System.*out*.println(set);

}

}

/\*

In this program we are checking the uses of floor() method of NavigableSet Interface. This method is checking that value whatever user is passing inside this floor() method that value is

exist inside this this list or not if not then it will check less than this value if there then it will return that value from this list.

Here is the output of this program:

Elements in set before floor method:

[0, 3, 5, 9, 15, 90]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elements in set After floor method:

9

9

15

[0, 3, 5, 9, 15, 90]

\*/

Example:

**package** com.lara.CollectionNotes.MapNotes.HashMapNotes;

**import** java.util.TreeSet;

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** HashMapNotes14

{

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

{

TreeSet set = **new** TreeSet();

set.add(90);

set.add(0);

set.add(9);

set.add(5);

set.add(3);

set.add(15);

System.*out*.println("Elements in set before headSet method: ");

System.*out*.println(set);

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

System.*out*.println("Elements in set After headSet method: ");

System.*out*.println(set.headSet(9));

System.*out*.println(set.headSet(10));

System.*out*.println(set.headSet(50));

System.*out*.println(set);

}

}

/\*

In this program we are checking headSet() method of NavigableSet Interface. Here in this program this headSet() method is used for getting before elements available of the value which

user is passing through headSet() method.

It will fetch all those elements before this value.

Here is the output of this program:

Elements in set before headSet method:

[0, 3, 5, 9, 15, 90]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elements in set After headSet method:

[0, 3, 5]

[0, 3, 5, 9]

[0, 3, 5, 9, 15]

[0, 3, 5, 9, 15, 90]

\*/

**How to create our own ArrayList or Custom ArrayList?**

This list is based on custom type means user defined ArrayList where we can do those facilities whatever we are doing in Java ArrayList.

Example:

**package** com.lara.CollectionNotes.ListNotes.CustomList;

**public** **class** MyArrayList

{

**private** Object elements[] = **null**;

**private** **int** size;

**private** **int** capacity;

//Inside default constructor we are initializing capacity variable and also we are creating the object of Object class array with passing capacity as size of the array of Object class.

**public** MyArrayList()

{

capacity = 10;

elements = **new** Object[capacity];

}

**public** **void** add(Object obj)

{

**if**(size == capacity)

{

alterCapacity();

}

elements[size++]= obj;

}

**private** **void** alterCapacity()

{

capacity = capacity\*2;

Object temp[] = elements;

elements = **new** Object[capacity];

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

{

elements[i] = temp[i];

}

}

**public** **int** size()

{

**return** size;

}

**public** Object get(**int** index)

{

**if**(index<0 || index>=size)

{

**throw** **new** IndexOutOfBoundsException();

}

**return** elements[index];

}

**public** String toString()

{

StringBuffer sb = **new** StringBuffer("[");

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

{

sb.append(elements[i]+", ");

}

sb.append("]");

**return** sb.substring(0, sb.length()-1);

}

}

**package** com.lara.CollectionNotes.ListNotes.CustomList;

**public** **class** CustomArrayList

{

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

{

MyArrayList list = **new** MyArrayList();

list.add("Sailendra");

list.add(20);

list.add(**true**);

list.add(20.15);

list.add(**null**);

list.add(9599480640l);

list.add('s');

list.add(20);

list.add(**true**);

list.add(20.15);

list.add(**null**);

list.add(9599480640l);

list.add('s');

System.*out*.println(list);

System.*out*.println(list.get(5));

}

}

Example:

**package** com.lara.CollectionNotes.ListNotes.CustomList;

**public** **class** AList

{

**private** Object elements[] = **null**;

**private** **int** size;

**private** **int** capacity;

**public** AList()

{

capacity = 10;

elements = **new** Object[capacity];

}

**public** **void** add(Object obj)

{

**if**(size != capacity)

{

alterCapacity();

}

elements[size++] = obj;

}

**private** **void** alterCapacity()

{

capacity = capacity\*2;

Object temp[] = elements;

elements = **new** Object[capacity];

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

{

elements[i] = temp[i];

}

}

**public** **int** size()

{

**return** size;

}

**public** Object get(**int** index)

{

**if**(index<0 || index>=size)

{

**throw** **new** IndexOutOfBoundsException();

}

**return** elements[index];

}

**public** String toString()

{

StringBuffer sb = **new** StringBuffer("[");

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

{

sb.append(elements[i]+", ");

}

sb.replace(sb.length()-1, sb.length(), "");

**return** sb.substring(0, sb.length()-1)+"]";

}

}

Example: