Collection in Action

Agenda

Collection Introduction

Collection Hierarchy

List

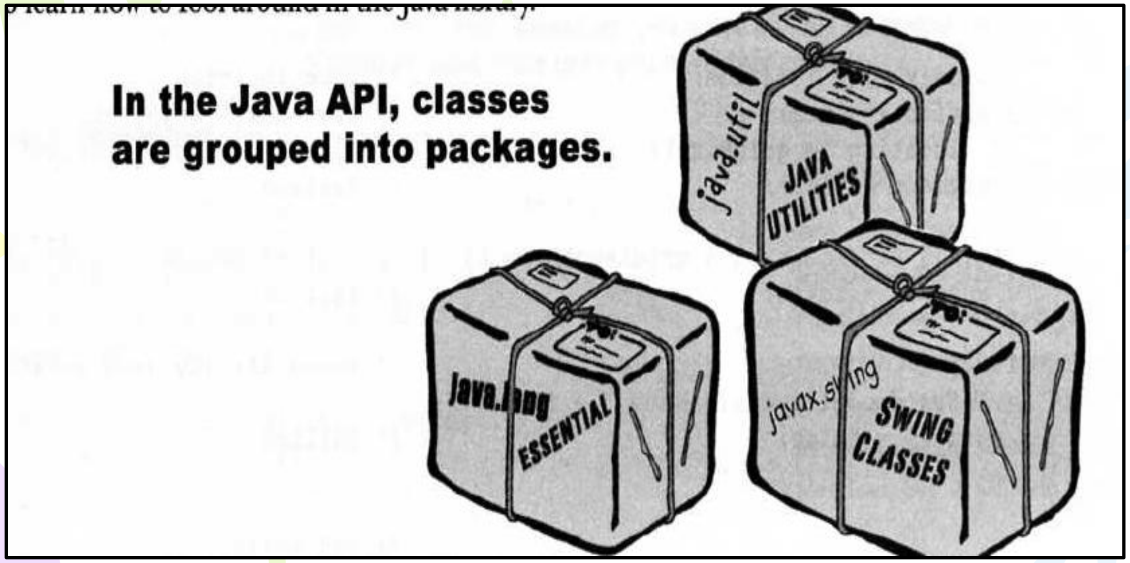
Set

Map

Sorting in Collection

Generics

Auto-Boxing



I want to make the playlist of songs and, also want check no duplicate song in the playlist and then sort the list.

For this i want to write lot of code, if i am a “***C or C++***” programmer.

Thank God! I am using java and java provides us ***java.util*** package, which provides inbuilt data-structure like ***ArrayList, LinkedList, TreeSet, HashMap, Collections*** etc.

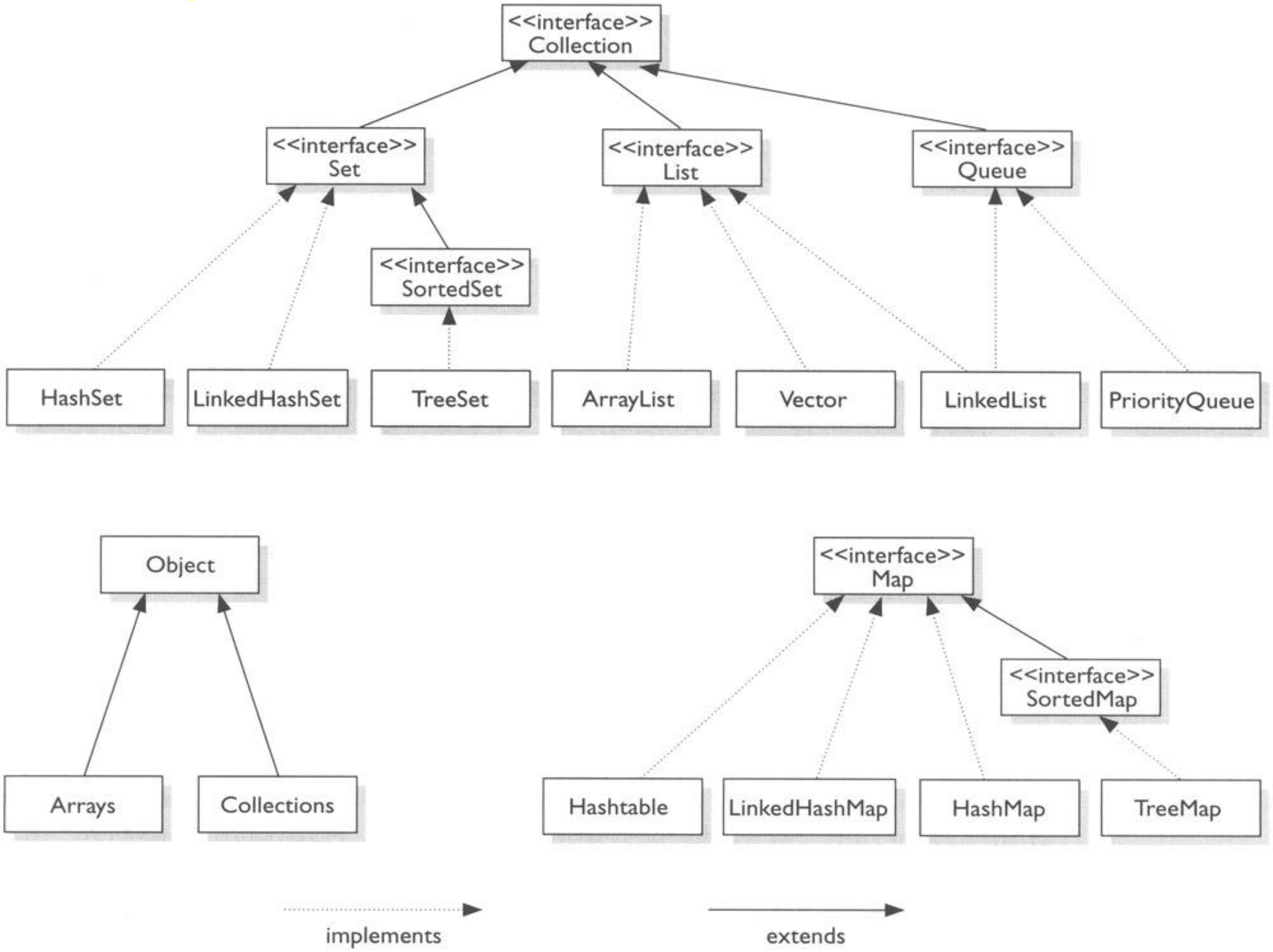
So programmer not worried to write long and boring codes for data-structure.

Java Collection Framework

So What Do You Do with a Collection?

- ✓ There are a few basic operations you'll normally use with the collections.
- ✓ Add Objects to the collection.
- ✓ Remove Object from the collection.
- ✓ Find out if an object in the collection.
- ✓ Reterive an object from the collection.
- ✓ Iterate through the collection.

Note: Collection is a collection of objects.
The Collection framework in java, which took shape with the release of JDK1.2 and was expanded in JDK1.4.


<<interface>> Collection
<<interface>> Set
<<interface>> List
<<interface>> Queue
<<interface>> SortedSet
HashSet
LinkedHashSet
TreeSet
ArrayList
Vector
LinkedList
PriorityQueue
Object
<<interface>> Map
<<interface>> SortedMap
Arrays
Collections
Hashtable
LinkedHashMap
HashMap
TreeMap
implements
extends

<u>Collections comes in four flavours</u>

1) List- List of Things, It cares about index.

2) Set – Unique Things

3) Map- Things with a Unique Id.

4) Queue- Things arranged by the order in which they are to be processed. (Java 5 / JDK 1.5 onwards)

List

List Interface – It cares about the index.

1. ArrayList
2. Vector
3. LinkedList

KICK START

Collection	create	add	remove	update	traverse
ArrayList	ArrayList names=new ArrayList();	names.add("amit"); names.add("ram",0);	names.remove(0); names.remove("ram");	names.set(0,"Amit");	names.get(0);
Vector	Vector names=new Vector(4);	Same as above + names.addElement("amit");	Same as above + names.removeElement("ram"); names.removeElementAt(0);	names.set(0,"Amit"); names.setElementAt("Amit",0);	names.get(0);
LinkedList	LinkedList names=new LinkedList();	Same as ArrayList + names.addFirst("Shyam "); names.addLast("mohan");	Same as ArrayList + names.removeFirst(); names.removeLast();// return object	Same as ArrayList	Same as ArrayList + names.getLast(); names.getFirst();
HashMap	HashMap phones=new HashMap();	phones.put("amit",new Long(9811223344));	phones.remove("amit");	Not Exist	phones.get("amit");
Hashtable	Hashtable phones=new Hashtable();	phones.put("amit",new Long(9811223344));	phones.remove("amit");	Not Exist	phones.get("amit");

List

Birthday party

Today is my birthday and i want to invite my friends for the party.
now i have to prepare a list of my friends.
So i just use an array of string and put it all the names in array. Now i want to add a new friend name so i have to make a logic to add a new name in array , and i have to check my array has empty space, because we know array is of fixed size.
Now i want to search a particular friend in that array, so i also have to make a logic for search.
For this purpose we have ArrayList and Vector. It provides predefine functions to perform **CRUD** (Create,Read,Update,Delete) operartions.

ArrayList	**Vector**
Think of this as a growable array	Same as ArrayList
It gives you fast iteration and fast random access.	Slower than ArrayList
ArrayList methods are not synchronized.	But vector methods are synchronized for thread safety.
Not a Legacy class.	Vector is a legacy class, and it is holdover from the earlier days of Java.

List

ArrayList and Vector Operations

operation	ArrayList	Vector

Creation (Here i am creating my party list)	ArrayList partyList=new ArrayList();
	or
	List partyList=new ArrayList();
		Vector partyList=new Vector();
		Vector partyList=new Vector(4);

Add (Adding my friend on my partylist)	partyList.add("Anil"); //return boolean
	partyList.add("Sunil");
	partyList.add(1,"Ram");
	partyList.add(new Integer(90));
	//any kind of value it can take
	//but can't take primitive type
		partyList.addElement("Hello");
		//void type
		//Remaining same

Remove (No i don't want to invite this friend)	System.out.println(partyList.remove(1)); //Ram print and remove it
	System.out.println(obj.remove("Sunil")); //print true and remove it.
		partyList.removeAllElements();
		partyList.removeElement("sunil");
		partyList.removeElementAt(1);

List

ArrayList and Vector Operations continued....

operation	ArrayList	Vector
Traverse -I (Now going to print partylist)	**First way to print a partylist** for(int i=0;i<partyList.size();i++) { System.out.println(partyList.get(i)); }	Enumeration e=partyList.elements(); while(e.hasMoreElements()) { System.out.println(e.nextElement()); }
Traverse-II	**2) Second Way to print a partylist** Iterator j=partyList.iterator(); while(j.hasNext()) { System.out.println(j.next()); }	Rest ways of traverse is same as ArrayList.
update	partyList.set(1,"tim");	partyList.set(1,"Tim"); partyList.setElementAt("Kim",2);

aa

List

ArrayList on job

Suppose I want to store 10 records of my customer in the ArrayList.

Create a Java Bean Name Customer , following are the properties.

- ✓ **Customerid**
- ✓ **name**
- ✓ **address**
- ✓ **city**
- ✓ **zip**
- ✓ **email**
- ✓ **phone**
- ✓ **mobile**
- ✓ **gender**

After adding 10 records, print the records.

List

LinkedList

Same like an ArrayList except that the elements are doubly linked to one another.
Keep in mind , your LinkedList is slower than ArrayList when you traverse it. But it is good choice when you required fast insertion and deletion.

Example:

```
   LinkedList l=new LinkedList();
    l.add("Amit");
    l.addFirst("Amit Srivastava");
    l.addLast("Ram");
    System.out.println("Linked List Elements are "+l);
    System.out.println("Return Object "+l.remove(2));
    System.out.println("Return Boolean "+l.remove("Amit"));
    System.out.println("Return First Object "+l.removeFirst());
    System.out.println("Return Last Object "+l.removeLast());
```

Set

No Duplicate Customer Records Allowed.

Suppose i want to make a list of my customers , but i don't want a duplicate
record of my customer.
What i need to do ?

Answer is Set

Set- It cares about uniqueness. The ***equals()*** method determines whether two objects are identical
There are three sets.

HashSet	**LinkedHashSet**	**TreeSet**
A HashSet is unsorted, unordered, no duplicates.	It is an ordered version of HashSet and no duplicate.	Sorted, no duplicate.
Inherit from Set	Inherit from set	Inherit from SortedSet.
Heterogeneous	Heterogeneous	Homogeneous

Set

Set Operations

Set	Creation	add	remove	Getting element
HashSet	HashSet names=new HashSet();	names.add("Amit");	names.remove("Amit");	Iterator i=names.iterator();
LinkedHashSet	LinkedHashSet names=new LinkedHashSet();	names.add("Amit");	names.remove("Amit");	Iterator i=names.iterator();
TreeSet	TreeSet names=new TreeSet();	names.add("Amit");	names.remove("amit");	Iterator i=names.iterator();

Map

Who's phone number is this ?

Suppose i want to store 100 phone numbers so i use **“ArrayList or Vector”** for this. Now i want to reterive the phone number of Amit , so for the “ArrayList or Vector” i have to remember the index number of Amit phone number.
So instead of remembering the index number of each phone number we can give a unique key to each index , and that unique key is name of a phone person.

A **Map** cares about unique identifiers. You may a **Unique key(the ID)** to a specific value, where both the key and the value are, of course objects.
So to use key(Name of a person) and value (phone number) we use “**HashMap or Hashtable”.**

HashMap	Hashtable
The HashMap gives you an unsorted , unordered Map.	Hashtable is a legacy class.
HashMap allows one null key and multiple null values in collection.	Hashtable doesn't let you have anything that's null.
HashMap is not synchronized.	Hashtable is the synchronized counterpart to HashMap.

Map

Map Operations

Operation	HashMap	Hashtable

creation	HashMap phones=new HashMap();
	or
	Map phones=new HashMap();
		Hashtable ht=new Hashtable()

add	HashMap phones=new HashMap();
	phones.put("Amit",new Long(9891580011));
	phones.put("Mohit",new Long(9818999900));
	phones.put("Sumit",new Long(9911223344));
	phones.put("Rohit",new Long(9988776655));
	//no duplicate key and take the last
	//one if duplicate
		Same

Map

Operation	HashMap	Hashtable
Getting the value	System.out.println(phones.get("Amit"));	Same
Getting al the keys	Set s=phones.keySet();	Same
Remove	phones.remove("Sumit");	

Traverse	Set s=phones.keySet();
	Iterator i=s.iterator();
	while(i.hasNext())
	{
	System.out.println(s.get(s.next());
	}
		Enumeration ee=phones.elements();
		//Getting all the values not keys
		while(ee.hasMoreElements())
		{
		System.out.println(ee.nextElement());
		}
		Enumeration s=phones.keys();
		//Getting all the keys

Map

Hashtable example

```
Hashtable ht=new Hashtable();
   ht.put("A","AAA");
   ht.put("B","BBB");
   Enumeration values=ht.elements();
   Enumeration keys=ht.keys();

   System.out.println("Printing Values ");
   while(values.hasMoreElements())
   {
     System.out.println(values.nextElement());
   }
   System.out.println("Printing Keys ");
   while(keys.hasMoreElements())
   { System.out.println(keys.nextElement());
   }
   System.out.println("Printing Keys , Values ");
   keys=ht.keys();
   while(keys.hasMoreElements())
   { String temp=(String)keys.nextElement();
     System.out.println("Key is "+temp +" Value is "+ht.get(temp));
   }
```

Map

A Map cares about unique identifiers. You map a unique key (the ID) to a specific value, where both the key and the value are, of course, objects.

HashMap	Hashtable	LinkedHashMap	TreeMap
Gives you unsorted and unordered Map.	Same like HashMap , but it is a legacy class.	Maintains insertion order.	It is sorted map.
HashMap h=new HashMap();	Hashtable ht=new Hashtable();	LinkedHashMap hm=new LinkedHashMap();	TreeMap tm=new TreeMap();
Where the keys land in the Map is based on the key's hashcode, so, like HashSet, the more efficient your hashCode() implementation, the better access performance you'll get. HashMap allows one null key and multiple null values in a collection.	Hashtable is the synchronized And Hashtable doesn't let you have anything that's null.	Although it will be somewhat slower than HashMap for adding and removing elements, you can expect faster iteration with a LinkedHashMap.	

TABLE 7-5 Key Methods in List, Set, and Map

Key Interface Methods	List	Set	Map	Descriptions

`boolean add(element)`
`boolean add(index, element)`
	X
	X
		X		Add an element. For Lists, optionally add the element at an index point.

`boolean contains(object)`
`boolean containsKey(object key)`
`boolean containsValue(object value)`
	X	X	
			X
			X
				Search a collection for an object (or, optionally for Maps a key), return the result as a `boolean`.

`object get(index)`
`object get(key)`
	X		
			X
				Get an object from a collection, via an index or a key.

`int indexOf(object)`	X			Get the location of an object in a List.
`Iterator iterator()`	X	X		Get an Iterator for a List or a Set.
`Set keySet()`			X	Return a Set containing a Map's keys.
`put(key, value)`			X	Add a key/value pair to a Map.

`remove(index)`
`remove(object)`
`remove(key)`
	X
	X
		
		X
			
			X
				Remove an element via an index, or via the element's value, or via a key.

`int size()`	X	X	X	Return the number of elements in a collection.

`Object[] toArray()`
`T[] toArray(T[])`
	X	X		Return an array containing the elements of the collection.

Map

HashMap on job

Exercise:

WAP to create four beans

- ✓**CustomerDetail Bean**
- ✓**CustAddress Bean**
- ✓**Order bean**
- ✓**ProductDetail bean.**

Put "**Customer bean**" objects and "**CustAddress"** bean objects in two different HashMaps
and
put "**Order Bean"** and "**ProductDetail bean"** objects in another two different HashMaps ,

and then put four hashmap objects into one hashmap and give keys ***customer ,product,order and address.***

Take input from the user if user ask for the customer so print CustomerDetail bean data by using last hashmap u made.
If user ask for Product, so print ProductDetail bean data.

QUESTION ???

I have a list of names in ArrayList, now i want to sort the names list ascending order.

Answer is :)

```
ArrayList names=new ArrayList();
names.add(“Ram”);
names.add(“Amit”);
names.add(“Anil”);
names.add(“Sunil”);
Collections.sort(names);
System.out.println(“After Sorting the list elements are “+names);
```

SORTING

QUESTION ? ? ?

Now fine after sorting a list of names, i have a list of customers object , and i want to sort list of customers by customer names?

Answer is :)

For Sorting user-defined objects , Collections.sort() fails, it cannot sort

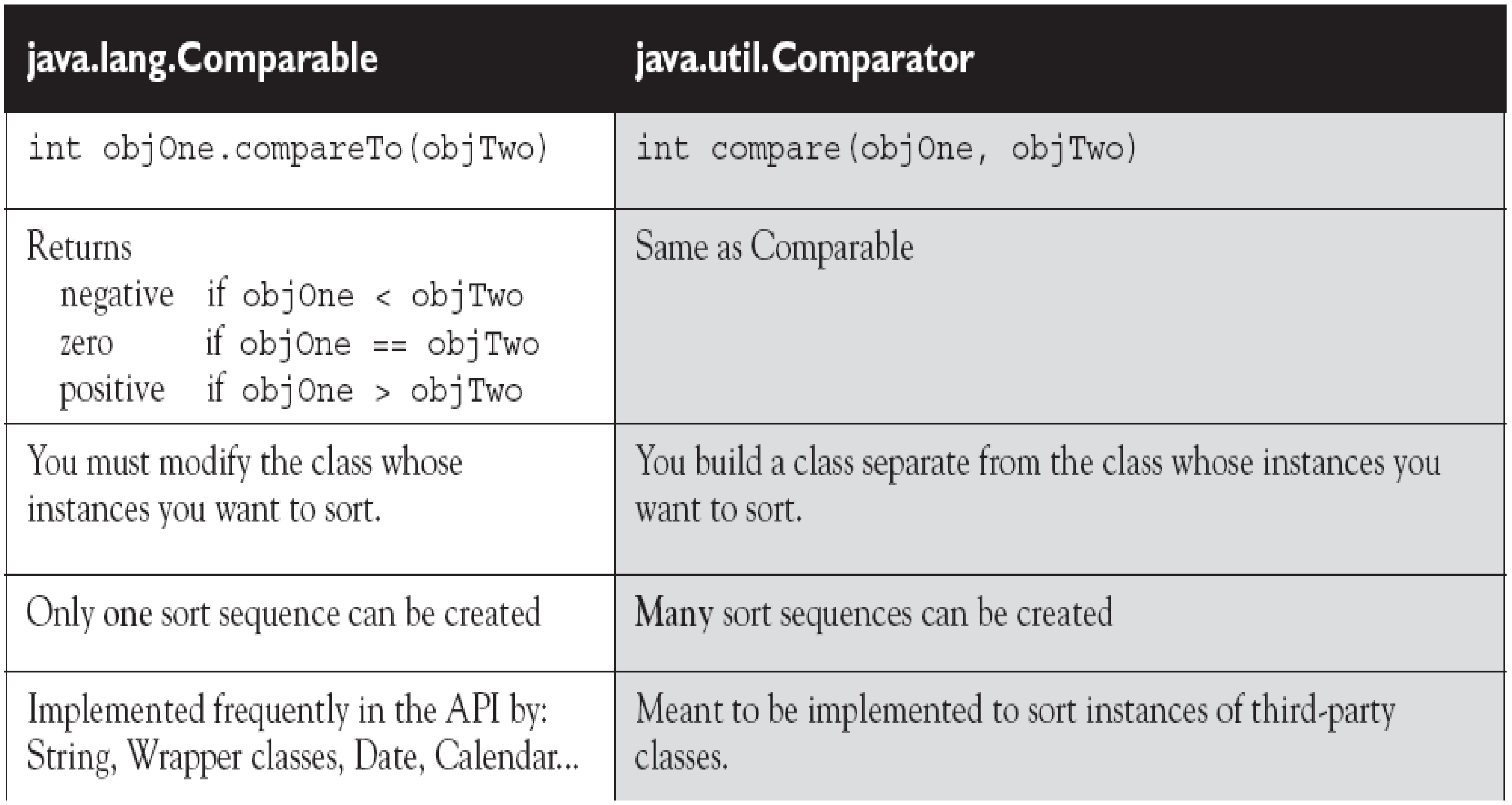
user-defined objects, so sorting user-defined objects , we required

Comparable

or

Comparator interface.

Comparable Vs Comparator



java.lang.Comparable	java.util.Comparator
`int objOne.compareTo(objTwo)`	`int compare(objOne, objTwo)`
Returns negative if `objOne < objTwo` zero if `objOne == objTwo` positive if `objOne > objTwo`	Same as Comparable
You must modify the class whose instances you want to sort.	You build a class separate from the class whose instances you want to sort.
Only one sort sequence can be created	Many sort sequences can be created
Implemented frequently in the API by: String, Wrapper classes, Date, Calendar...	Meant to be implemented to sort instances of third-party classes.

java.lang.Comparable Example

```
import java.util.*;
class Customer implements java.lang.Comparable //or java.lang.Comparable<Customer>
{
private int custid;
private String name;
private int sal;
Customer()
{
custid=1001;
name="Amit";
sal=99090;
}
Customer(int custid,String name,int sal)
{
this.custid=custid;
this.name=name;
this.sal=sal;
}
public int compareTo(Object o)
{
Customer e=(Customer)o;
return this.name.compareToIgnoreCase(e.name);
}
public String toString()
{
return "Customer id is "+custid+ " Name is "+name+" Salary is "+sal;
}
}
public class Sorting1
{
    public static void main(String[] args)
   {
   Customer obj1=new Customer();
   Customer obj2=new Customer(1002,"Ram",4444);
   Customer obj3=new Customer(1003,"Anil",6666);
   Customer obj4=new Customer(1004,"Sunil",7777);

   ArrayList l=new ArrayList();
   l.add(obj1);
   l.add(obj2);
   l.add(obj3);
   l.add(obj4);

   Collections.sort(l);  //The Comparable Interface is used by Collections.sort() and Arrays.sort()

   System.out.println(l);
    }
}
```

```
import java.util.*;
class SortByName implements java.util.Comparator<Customer>
{
public int compare(Customer first,Customer second)
{
return  first.name.compareToIgnoreCase(second.name);
}
}
class Customer
{
 int custid;
String name;
 int sal;
Customer()
{
custid=1001;
name="Amit";
sal=9090;
}
Customer(int custid,String name,int sal)
{
this.custid=custid;
this.name=name;
this.sal=sal;
}
public int compareTo(Object o)
{
Customer e=(Customer)o;
return this.name.compareToIgnoreCase(e.name);
}
public String toString()
{
return "Customer id is "+custid+ " Name is "+name+" Salary is "+sal;
}
}
public class Sorting2
{   public static void main(String[] args)
    {
```

```
Customer obj1=new Customer();
 Customerobj2
=newCustomer(1002,"Ram",4444);
Customer obj3=
new Customer(1003,"Anil",6666);
Customer obj4=
new Customer(1004,"Sunil",7777);
   ArrayList l=new ArrayList();
    l.add(obj1);
    l.add(obj2);
    l.add(obj3);
    l.add(obj4);
    Collections.sort(l,new SortByName());
    System.out.println(l);
     }
}
```

Queue- **Things arranged by the order in which they are to be processed.**

Priority Queue- The purpose of a PriorityQueue is to create a "priority-in, priority out" queue as opposed to a typical FIFO queue. A PriorityQueue's elements are ordered either by natural ordering (in which case the elements that are sorted first will be accessed first) or according to a Comparator. In either case, the elements' ordering represents their relative priority.

Priority Queue orders its elements using a user-defined priority.

In addition, a PriorityQueue can be ordered using a Comparator, which lets you define any ordering you want. Queues have a few methods not found in other collection interfaces: `peek()`, `poll()`, and `offer()`.

`offer()` method to add elements to the PriorityQueue

`poll()` method, which returns the highest priority entry in Proirity-Queue AND removes the entry from the queue.

`peek()` returns the highest priority element in the queue without removing it, and `poll()` returns the highest priority element, AND removes it from the queue.

```
class PQ {
  static class PQsort
          implements Comparator<Integer> {   // inverse sort
    public int compare(Integer one, Integer two) {
      return two - one;                      // unboxing
    }
  }
  public static void main(String[] args) {
    int[] ia = {1,5,3,7,6,9,8 };             // unordered data
    PriorityQueue<Integer> pq1 =
      new PriorityQueue<Integer>();          // use natural order

    for(int x : ia)                          // load queue
      pq1.offer(x);
    for(int x : ia)                          // review queue
```

Help

```
      System.out.print(pq1.poll() + " ");
    System.out.println("");

    PQsort pqs = new PQsort();               // get a Comparator
    PriorityQueue<Integer> pq2 =
      new PriorityQueue<Integer>(10,pqs);    // use Comparator

    for(int x : ia)                          // load queue
      pq2.offer(x);
    System.out.println("size " + pq2.size());
    System.out.println("peek " + pq2.peek());
    System.out.println("size " + pq2.size());
    System.out.println("poll " + pq2.poll());
    System.out.println("size " + pq2.size());
    for(int x : ia)                          // review queue
      System.out.print(pq2.poll() + " ");
  }
}
```

Arrays class and Collections class

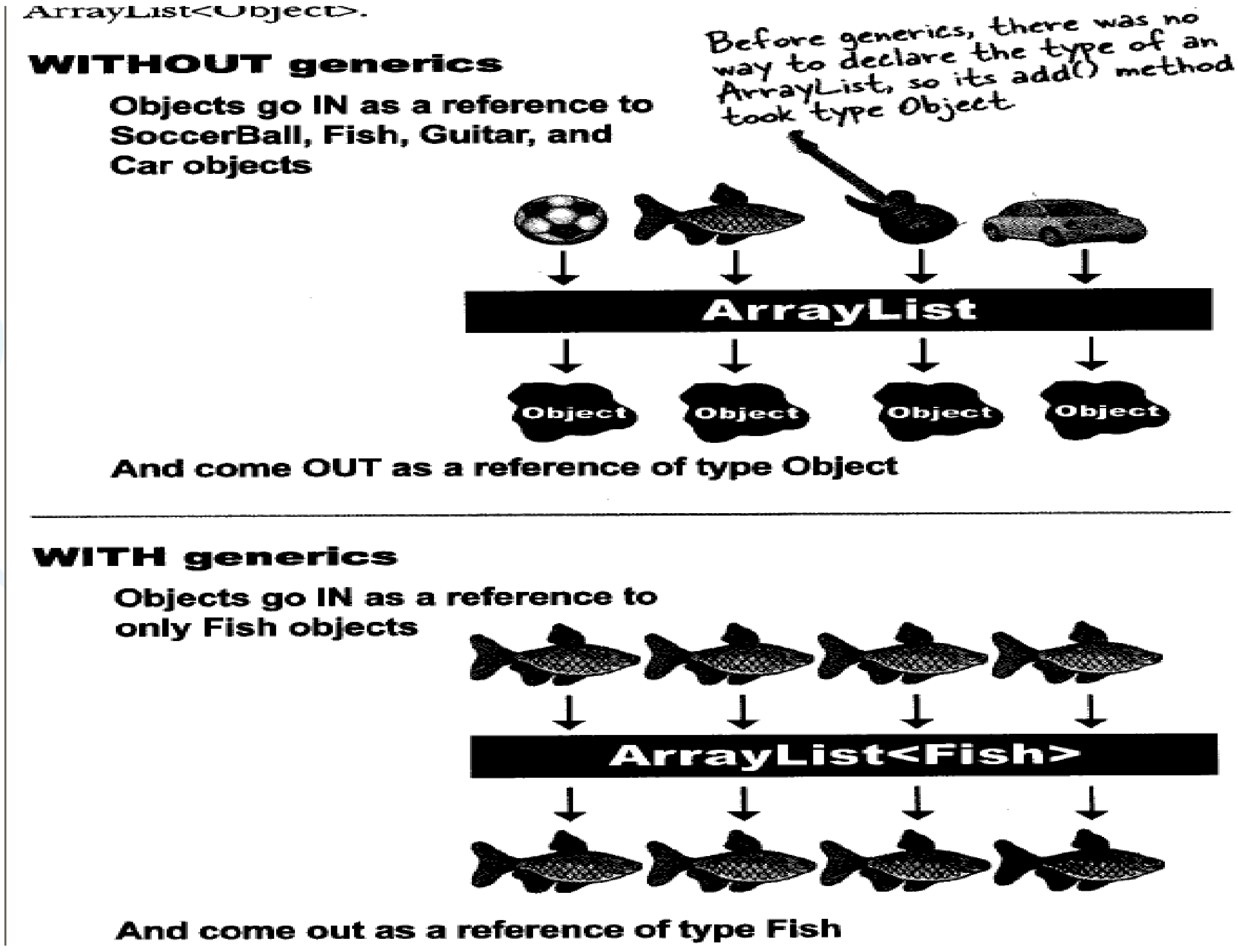
TABLE 7-4 Key Methods in Arrays and Collections

Key Methods in java.util.Arrays	Descriptions
static List **asList**(T[])	Convert an array to a List, (and bind them).
static int **binarySearch**(Object[], key) static int **binarySearch**(primitive[], key)	Search a sorted array for a given value, return an index or insertion point.
static int **binarySearch**(T[], key, Comparator)	Search a Comparator-sorted array for a value.
static boolean **equals**(Object[], Object[]) static boolean **equals**(primitive[], primitive[])	Compare two arrays to determine if their contents are equal.
public static void **sort**(Object[ ]) public static void **sort**(primitive[ ])	Sort the elements of an array by natural order.
public static void **sort**(T[], Comparator)	Sort the elements of an array using a Comparator.
public static String **toString**(Object[]) public static String **toString**(primitive[])	Create a String containing the contents of an array.

Key Methods in java.util.Collections	Descriptions
static int **binarySearch**(List, key) static int **binarySearch**(List, key, Comparator)	Search a "sorted" List for a given value, return an index or insertion point.
static void **reverse**(List)	Reverse the order of elements in a List.
static Comparator **reverseOrder**() static Comparator **reverseOrder**(Comparator)	Return a Comparator that sorts the reverse of the collection's current sort sequence.
static void **sort**(List) static void **sort**(List, Comparator)	Sort a List either by natural order or by a Comparator.

Generics

Generics means more type safety.


ArrayList<Object>.

WITHOUT generics

Objects go IN as a reference to SoccerBall, Fish, Guitar, and Car objects

Before generics, there was no way to declare the type of an ArrayList, so its add() method took type Object

And come OUT as a reference of type Object

WITH generics

Objects go IN as a reference to only Fish objects

And come out as a reference of type Fish

Generics

Non-Generic	Generic

ArrayList obj=new ArrayList();
or
List obj=new ArrayList();
	ArrayList<String> obj=new ArrayList<String>();
	or
	List<String> obj=new ArrayList<String>();

obj.add(new Integer(20)); //Ok Take Integer
obj.add("Hello"); //Take String
obj.add(new Float(90.10f)); //Take Float
	obj.add("Amit"); //Only hold String type
	obj.add(new Integer()); //Compiler Error !

String a=(String)obj.get(1);	String a=obj.get(1); // No Need of Type casting

Object o;
for(int i=0;i<=obj.size();i++)
{
o=obj.get(i); //Everything treated as an object.
System.out.println(o);
}
	for(String z:obj)
	{
	System.out.println(z);
	}

Non Generic is a Legacy code	Generic Introduce in JDK1.5
Use Iterator or old for loop or Enumeration to traverse	Use Enhance for loop , this loop is also introduced in JDK1.5

Mixing Generic and Non-Generic

Polymorphism in Generic

Generic Methods

Generic Class

MIXING GENERIC AND NON-GENERIC COLLECTIONS

```
package mypackage;
import java.util.ArrayList;
import java.util.List;

public class C
{
static void show(List l)
{
  l.add("XYZ");  //No Error treated as Legacy type not generic
  l.add(new Float(90.89));
  l.add(new Integer(10));
  System.out.println("List Value is "+l);
}
  public static void main(String args[])
  {
    ArrayList<Integer> l=new ArrayList<Integer>();
    l.add(90);
    l.add(100);
    show(l);
    //l.add("Hello");  //Error


  }
}
```

Polymorphism and Generics

You've already seen that polymorphism applies to the "base" type of the collection:
List<Integer> myList = new ArrayList<Integer>();

List<Object> myList = new ArrayList<JButton>(); // NO!
List<Number> numbers = new ArrayList<Integer>(); // NO!
// remember that Integer is a subtype of Number

But these are fine:
List<JButton> myList = new ArrayList<JButton>(); // yes
List<Object> myList = new ArrayList<Object>(); // yes
List<Integer> myList = new ArrayList<Integer>(); // yes
So far so good. Just keep the generic type of the reference and the generic type of the object to which it refers identical.

Object[] myArray = new JButton[3]; // yes

but not this:

List<Object> list = new ArrayList<JButton>(); // NO!

List<?> obj=new ArrayList<?>();

List<?> obj=new ArrayList<? extends String>();

List<?> obj=new ArrayList<? super String()>();

NOTE: The same extends keyword use for the class and interface in wild card.

<u>Auto-Boxing in Generics</u>

Auto-boxing is introduced in JDK1.5, Auto-boxing has two terms

1) Boxing – Converting Primitive type into reference type.

2) Un-Boxing- Converting reference type into primitive type.

Example:

```
ArrayList<Integer> l=new ArrayList<Integer>();
l.add(10); //Boxing
l.add(new Integer(10));  //Legacy way
int i=l.get(i);  //UnBoxing
```

Thank You !