Java Collections

Based on lecture notes from
Prof. Michael Langer

Key concepts

- Goal: Learn how to use ADTs and their implementation in Java
- Interface
- Generic class/interface
- Java Collection interfaces and implementations

Java interface

- A Java interface is similar to a class, but only the method signatures are provided

This is called a generic interface. T is the type of objects stored in the list

```
interface List<T> {
   void add(T);       // adds object at the end of the list
   void add(int, T);  // inserts object at specific position
   T remove(int);     // removes object at position
   boolean isEmpty();
   T get( int );      // returns object at position
   int size();
}
```

- Java interface = Abstract Data Type
- An interface cannot be instantiated:

List l = new List<Integer> (); ← Illegal

https://docs.oracle.com/javase/7/docs/api/java/util/List.html

Implementing an interface

- A class can *implement* an interface: it provides code for each of the methods of the interface

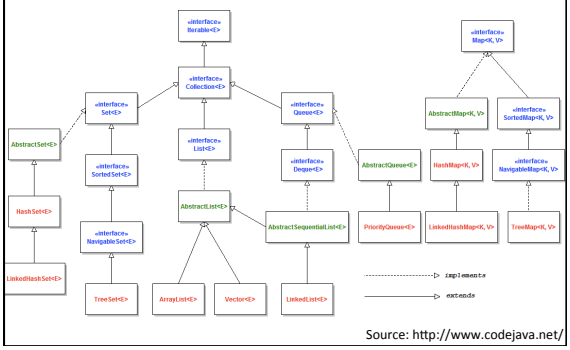
This is called a generic class. T is the type of objects stored in the list

This enforces that the class provides all the methods of the List interface

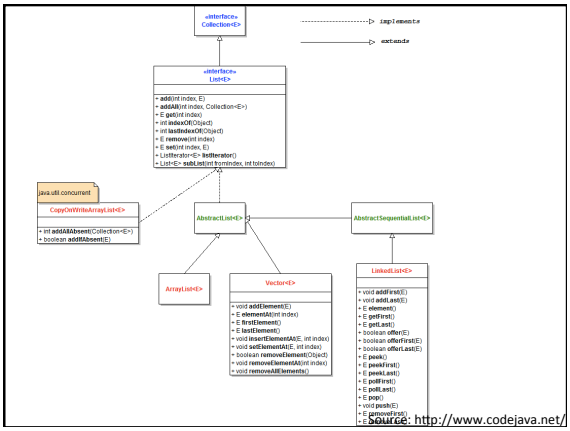
```
class ArrayList<T> implements List {
   T myArray[]; // stores the elements of the list
   int size;       // number of elements in the list
   void add(T element) { .... }
   void add(int position, T element) { .... }
   T remove(int position) { .... }
   boolean isEmpty() { .... }
   T get( int position) { .... }
   int size() { .... }
}
```

- A generic class can be instantiated as follows
  ArrayList<String> myList = new ArrayList<String> ();

The Java Collections (incomplete)



Source: http://www.codejava.net/



Source: http://www.codejava.net/

The Iterator interface

An Iterator is used to traverse a Collection of objects

```
interface Iterator {
   boolean hasNext(); // are there more elements?
   T next(); // returns current and advances to next
}
```

Example:

```
ArrayList<String> list=new ArrayList<String>();
list.add("Ravi");
list.add("Vijay");
//Traversing list using Iterator
Iterator<String> itr = list.iterator();
while( itr.hasNext() ) {
      String s = itr.next();
      System.out.println( s );
}
```

Java enhanced for loops with iterators

Standard use of iterator:

```
ArrayList<String> list=new ArrayList<String>();
list.add("Ravi");
list.add("Vijay");
// Traversing list using Iterator
Iterator<String> itr = list.iterator();
while( itr.hasNext() ) {
      String s = itr.next();
      System.out.println( s );
}
```

Enhanced looping:

```
ArrayList<String> list=new ArrayList<String>();
list.add("Ravi");
list.add("Vijay");
// Traversing list using enhanced loop
for ( String s : list ) {
    System.out.println( s );
}
```

- Applicable to any class that implements Iterable.

The Iterable interface

```
interface Iterable {
   Iterator iterator();
}
```

If a class implements Iterable, then the class has an iterator() method.

Since all types of Collection interfaces extend the Iterable interface, all are iterable.

The Comparable interface

Java defines the Comparable interface as follows:

```
interface Comparable<T> {
   int compareTo( T other );      // returns: 0                if this.equals( other )
                                  //          positive number  if this > other
                                  //          negative number  if this < other
}
```

➔ Note: Integer, Float, String all implement the Comparable interface

```
class SportTeam implements Comparable {
...
   int compareTo (SportTeam other) {
      if (this.victories == other.victories ) return 0;  // Note: According to Java conventions,
                                                         // compareTo should only return 0 if this.equals(other)
      if (this.victories> other.victories) return 1;
      if (this.victories < other.victories) returns -1;
}}
```

➔ Note: Any Collection of Comparable elements can be sorted using Java 8 **Collection**s class:

```
ArrayList<SportTeam> myTeams = new ArrayList(10);
Collections.sort(myTeams);
```

Example: Finding the max of a Collection

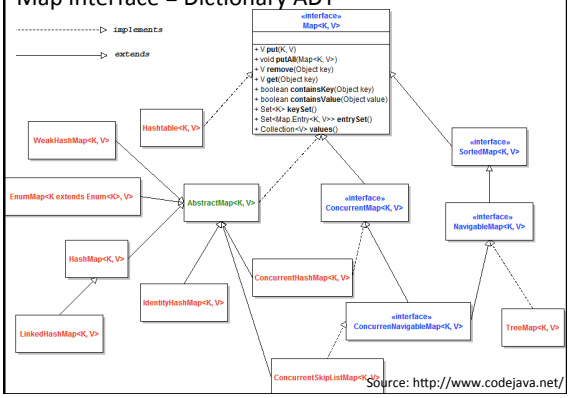
Method operates on a type T that implements the Comparable interface

Method returns object of type T

Argument is a Collection of objects of type T

```
public static <T implementsComparable> T max( Collection<T> coll) {
   Iterator<T> i = coll.iterator();
   T maxSoFar = i.next();
   while ( i.hasNext() ) {
      T current = i.next();
      if (current.compareTo(maxSoFar) > 0) maxSoFar = current;
   }
   return maxSoFar;
}
public static void main(String args[]) {
    ArrayList<SportTeam> firstList=new ArrayList<SportTeam>(10);
    LinkedList<String> secondList = new LinkedList<String> ();
    // Some code to fill the two lists
    SportTeam m = max(firstList);
    String s = max(secondList);   // Note how the same method is used on the two lists!
}
```

Map interface = Dictionary ADT



Source: http://www.codejava.net/

Using HashMaps

```
import java.util.*;
class TestCollection13{
  public static void main(String args[]){
    HashMap<Integer,String> hm=new HashMap<Integer,String>();
    hm.put(100,"Amit");
    hm.put(101,"Vijay");
    hm.put(102,"Rahul");
    for(Map.Entry m : hm.entrySet() ){
      System.out.println(m.getKey()+" "+m.getValue());
 }
 }
}
```

Java Application Programming Interface (API)

Complete documentation:
https://docs.oracle.com/javase/7/docs/api

- ArrayList:
https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

- LinkedList:
https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

- HashMap:
https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html