Collections

(http://docs.oracle.com/javase/tutorial/collections/index.html)

Objectives

- How to use the collection framework for performing common operations on a group of objects
  - List: ArrayList, Vector → Duplicates are agreed
  - Set: HashSet, TreeSet → Duplicates are not agreed
  - Map: HashMap, TreeMap
- Understand and use Generic in the collection framework.

Content

- Abstract Data Types
- Collection framework
  - List
  - Set
  - Map
- Generics

Abstract Data Types

- An ***abstract data type (ADT)*** is a mathematical model for a data type that is determined based on **generalization in which data structure**, a way for storing data, is omitted. An abstract data type describes a general concept in reality.

  For example: a collection is an abstract data type.

- An abstract data type is defined by its behaviors from the point of view of a user. Programming languages, such as Java, define an ADT as an interface in which a set of behaviors are identified and declared.

- When an ADT is used, a concrete class implementing appropriate interfaces must be defined. A group of elements can be viewed as in some ways:
  - ✓ a list (group of objects in which duplications are allowed),
  - ✓ a set (a group of distinct objects),
  - ✓ a map(group of objects in which each object is defined by). So, all of them are ADTs.

The Collections Framework

- The Java 2 platform includes a new *collections framework*.
- A *collection* is an object that represents a group of objects.
- The Collections Framework is a unified architecture for representing and manipulating collections.
- The collections framework as a whole is not threadsafe.

The Collections Framework…

- **Reduces programming effort** by providing useful data structures and algorithms so you don't have to write them yourself.
- **Increases performance** by providing high-performance implementations of useful data structures and algorithms.
- **Provides interoperability between unrelated APIs** by establishing a common language to pass collections back and forth.
- **Reduces the effort required to learn APIs** by eliminating the need to learn multiple ad hoc collection APIs.
- **Reduces the effort required to design and implement APIs** by eliminating the need to produce ad hoc collections APIs.
- **Fosters software reuse** by providing a standard interface for collections and algorithms to manipulate them.

Collection Interfaces

- java.lang.**Iterable**<T>
  - java.util.**Collection**<E>
    - java.util.**List**<E>
    - java.util.**Queue**<E>
      - java.util.**Deque**<E>
    - java.util.**Set**<E>
      - java.util.**SortedSet**<E>
        - java.util.**NavigableSet**<E>
- java.util.**Map**<K,V>
  - java.util.**SortedMap**<K,V>
    - java.util.**NavigableMap**<K,V>

Methods declared in these interfaces can work on a list containing elements which belong to arbitrary type. T: type, E: Element, K: Key, V: Value

Details of this will be introduced in the topic Generic

3 types of group:
List can contain duplicate elements
Set can contain distinct elements only
Map can contain pairs <key, value>. Key of element is data for fast searching

Queue, Deque contains methods of restricted list.

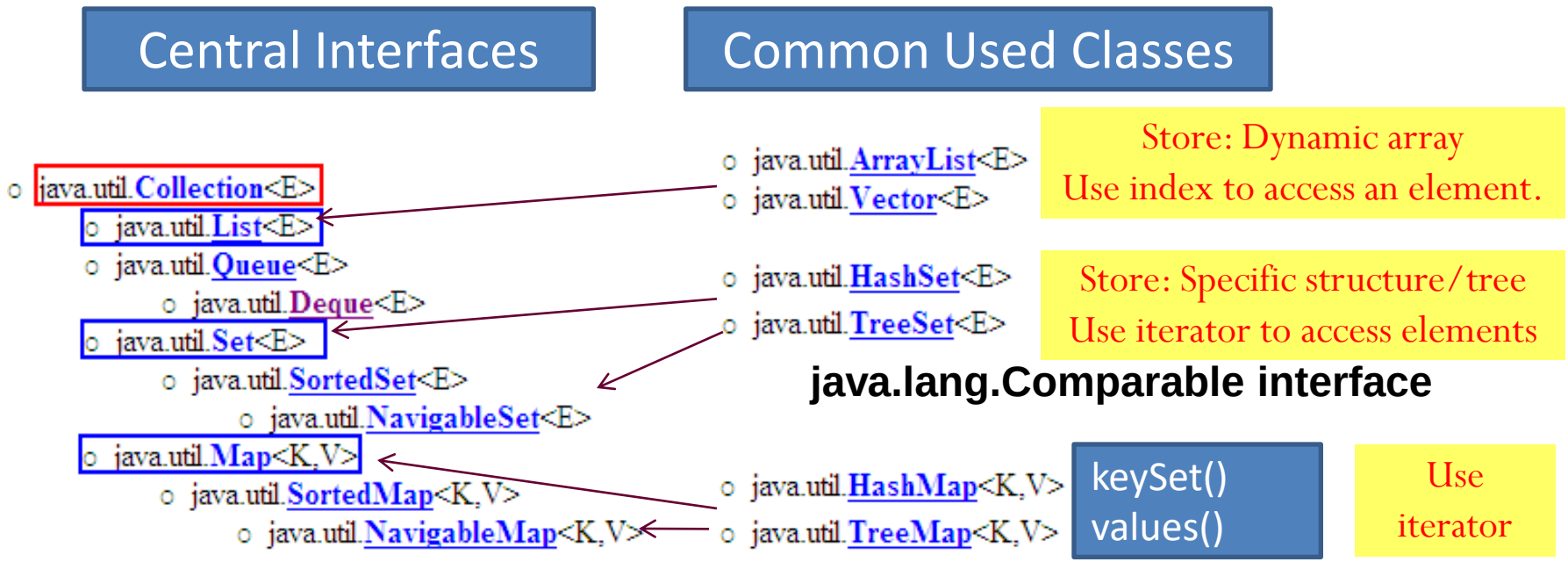
Common methods on group are: Add, Remove, Search, Clear,…

Common Methods of the interface Collection

Method	Description
`add(Object x)`	Adds x to this collection
`addAll(Collection c)`	Adds every element of c to this collection
`clear()`	Removes every element from this collection
`contains(Object x)`	Returns true if this collection contains x
`containsAll(Collection c)`	Returns true if this collection contains every element of c
`isEmpty()`	Returns true if this collection contains no elements
`iterator()`	Returns an Iterator over this collection (see below)
`remove(Object x)`	Removes x from this collection
`removeAll(Collection c)`	Removes every element in c from this collection
`retainAll(Collection c)`	Removes from this collection every element that is not in c
`size()`	Returns the number of elements in this collection
`toArray()`	Returns an array containing the elements in this collection

Elements can be stored using some ways such as an array, a tree, a hash table. Sometimes, we want to traverse elements as a list → We need a list of references → Iterator

The Collection Framework…



A TreeSet will stored elements using ascending order. Natural ordering is applied to numbers and lexicographic (dictionary) ordering is applied to strings.

If you want a TreeSet containing your own objects, you must implement the method compareTo(Object), declared in the Comparable interface.

Lists

- A List keeps it elements in the order in which they were added.
- Each element of a List has an index, starting from 0.
- Common methods:
  - **void add(int index, Object x)**
  - **Object get(int index)**
  - **int indexOf(Object x)**
  - **Object remove(int index)**

Classes Implementing the interface List

- AbstractList
- ArrayList
- Vector (like ArrayList but it is synchronized)
- LinkedList: *linked lists can be used as a stack, queue, or double-ended queue (deque)*

List Implementing Classes

```
ArrayList list= new ArrayList();
for (int i = 101; i <= 110; i++) {
        list.add(i);
}
for (int i = 0; i < list.size(); i++) {
         System.out.println(list.get(i));
}
//or using Iterator
/*
     Iterator iter = list.iterator();
     while (iter.hasNext()) {
         System.out.println(iter.next());
     }
*/
```

Using the Vector class

java.util.**Vector**<E> (implements java.lang.Cloneable, java.util.List<E>, java.util.RandomAccess, java.io.Serializable)

The Vector class is obsolete from Java 1.6 but it is still introduced because it is a parameter in the constructor of the javax.swing.JTable class, a class will be introduced in GUI programming.

```
import java.util.Vector;
class Point {
  int x,y;
  Point() { x=0; y=0; }
  Point(int xx, int yy){
      x=xx; y=yy;
  }
  public String toString() { return "[" + x + "," + y + "]";}
}
public class UseVector {
  public static void main(String[] args) {
      Vector v = new Vector();
      v.add(15);
      v.add("Hello");
      v.add(new Point());
      v.add(new Point(5,-7));
      System.out.println(v);
      v.remove(2);
      System.out.println(v);
      for (int i=0;i<v.size();i++) System.out.print(v.get(i) + ", ");
      System.out.println();
  }
}
```

```
Output - Chapter08 (run)
run:
[15, Hello, [0,0], [5,-7]]
[15, Hello, [5,-7]]
15, Hello, [5,-7],
```

Sets

- Lists are based on an ordering of their members. Sets have no concept of order.
- A Set is just a cluster of references to objects.
- Sets may not contain duplicate elements.
- Sets use the equals() method, not the == operator, to check for duplication of elements.

```
void addTwice(Set set) {
    set.clear();
    Point p1 = new Point(10, 20);
    Point p2 = new Point(10, 20);
    set.add(p1);
    set.add(p2);
    System.out.println(set.size());
}
```

will print out 1, not 2.

Sets…

- Set extends Collection but does not add any additional methods.
- The two most commonly used implementing classes are:
  - TreeSet
    - Guarantees that the sorted set will be in ascending element order.
    - log(n) time cost for the basic operations (add, remove and contains).
  - HashSet
    - Constant time performance for the basic operations (add, remove, contains and size).

TreeSet and Iterator

- Ordered Tree – Introduced in the subject Discrete Mathematics
- Set: Group of different elements
- TreeSet: Set + ordered tree, each element is called as node
- Iterator: An operation in which references of all node are grouped to make a linked list. Iterator is a way to access every node of a tree.
- Linked list: a group of elements, each element contains a reference to the next

TreeSet = Set + Tree

```
Random r = new Random();
TreeSet myset = new TreeSet();
for (int i = 0; i < 10; i++) {
    int number = r.nextInt(100);
    myset.add(number);
}
//using Iterator
Iterator iter = myset.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next());
}
```

The result may be:

```
7
27
36
41
43
46
49
57
75
83
```

Using the TreeSet class & Iterator

```
import java.util.TreeSet;
import java.util.Iterator;
public class UseTreeSet {
  public static void main (String[] args){
      TreeSet t= new TreeSet();
      t.add(5); t.add(2); t.add(9);t.add(30); t.add(9);
      System.out.println(t);
      t.remove(9);
      System.out.println(t);
      Iterator it= t.iterator();
      while (it.hasNext())
          System.out.print(it.next() + ", ");
      System.out.println();
  }
}
```

```
Output - Chapter08 (run)
run:
[2, 5, 9, 30]
[2, 5, 30]
2, 5, 30,
```

A TreeSet will stored elements using ascending order. Natural ordering is applied to numbers and lexicographic (dictionary) ordering is applied to strings.

If you want a TreeSet containing your own objects, you must implement the method compareTo(Object), declared in the Comparable interface.

Hash Table

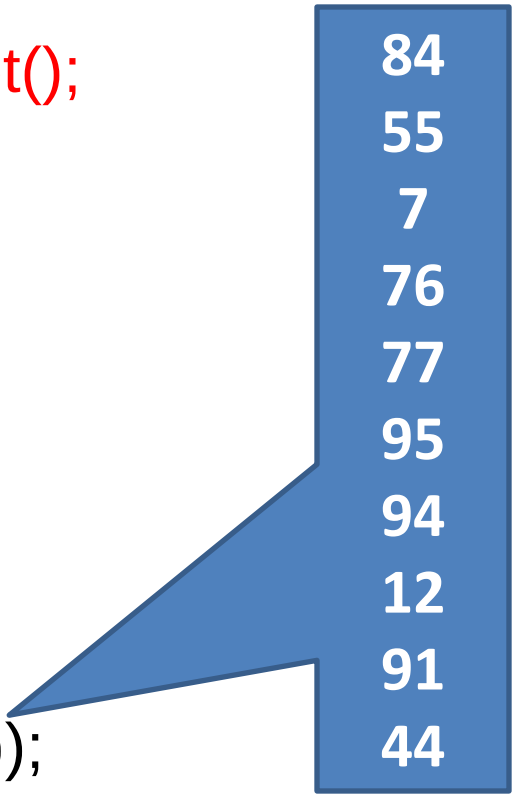
- In array, elements are stored in a contiguous memory blocks → Linear search is applied → slow, binary search is an improvement.
- Hash table: elements can be stored in a different memory blocks. The index of an element is determined by a function (hash function) → Add/Search operation is very fast (O(1)).

The hash function f may be:
'S'*10000+'m'*1000+'i'*100+'t'*10+'h' % 50

HashSet = Set + Hash Table

```
Random r = new Random();
HashSet myset = new HashSet();
for (int i = 0; i < 10; i++) {
    int number = r.nextInt(100);
    myset.add(number);
}
//using Iterator
Iterator iter = myset.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next());
}
```

The result may be:



HashSet or TreeSet?

- If you care about iteration order, use a Tree Set and pay the time penalty.
- If iteration order doesn't matter, use the higher-performance Hash Set.

How to TreeSet ordering elements?

- Tree Sets rely on all their elements implementing the interface java.lang.Comparable.

public int compareTo(Object x)

- Returns a positive number if the current object is “greater than” x, by whatever definition of “greater than” the class itself wants to use.

How to TreeSet ordering elements?

```
class Student implements Comparable{
    int no;
    ...
   public int compareTo(Object o) {
         Student st = (Student) o;
         if(no > st.getNo())
             return 1;
         else if(no == st.getNo())
             return 0;
         else
             return -1;
     }
    . . .
}
```

Comparing 2 students based on their IDs (field ***no***)

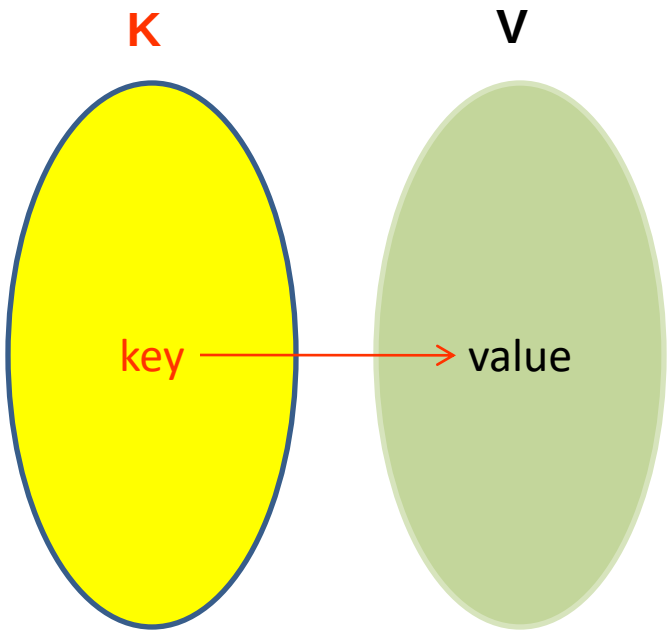
How to TreeSet ordering elements?

```
public static void main(String[] args) {
   Random r =  new Random();
   TreeSet myset = new TreeSet();
   for (int i = 0; i < 10; i++) {
        int no = r.nextInt(100);
        Student st = new Student(no, "abc");
        myset.add(st);
   }
   //using Iterator
   Iterator iter = myset.iterator();
   while (iter.hasNext()) {
        Student st = (Student)iter.next();
        System.out.println("No: " + st.getNo());
   }
}
```

```
No: 2
No: 8
No: 11
No: 19
No: 33
No: 52
No: 78
No: 83
No: 92
No: 96
```

Maps

- Map doesn't implement the java.util.Collection interface.
- A Map combines *two* collections, called keys and values.
- The Map's job is to associate exactly one value with each key.
- A Map like a dictionary.
- Maps check for key uniqueness based on the equals() method, not the == operator.
- IDs, Item code, roll numbers are keys.
- The normal data type for keys is String.



Each element: <key,value>

Maps..

- Java's two most important Map classes:
  - HashMap (mapping keys are unpredictable order – hash table is used, hash function is pre-defined in the Java Library).
  - TreeMap (mapping keys are natural order)-> all keys must implement Comparable (a tree is used to store elements).

HashMap

```
public static void main(String[] args) {
        HashMap mymap = new HashMap();
        mymap.put(1, “One");
        mymap.put(2, “Two");
        mymap.put(3, “Three");
        mymap.put(4, “Four");
        //using Iterator
        Iterator iter = mymap.keySet().iterator();
        while (iter.hasNext()) {
            Object key = iter.next();
            System.out.println(key + ": " + mymap.get(key));
        }
    }
```

```
//output
1: One
2: Two
3: Three
4: Four
```

Key: integer, value: String

Using HashMap class & Iterator

```
1  import java.util.HashMap;
2  import java.util.Iterator;
3  public class UseHashMap {
4      public static void main(String[] args){
5          HashMap h = new HashMap();
6          h.put("Sáu Tấn", "Huỳnh Anh Tuấn");
7          h.put("Bình Gà", "Nguyễn Tấn Sầu");
8          h.put("Ba Địa", " Trần Mai Hoà");
9          System.out.println(h);
10         h.put("Sáu Tấn","Nguyễn Văn Tuấn");
11         System.out.println(h);
12         h.remove("Bình Gà");
13         System.out.println(h);
14         Iterator it = h.keySet().iterator();
15         while (it.hasNext())
16         { String key= (String)(it.next());
17           String value = (String)(h.get(key));
18           System.out.println(key + ", " + value);
19         }
20     }
21 }
```

Key: String, value: String

```
Output - Chapter08 (run)
run:
{Ba Địa= Trần Mai Hoà, Sáu Tấn=Huỳnh Anh Tuấn, Bình Gà=Nguyễn Tấn Sầu}
{Ba Địa= Trần Mai Hoà, Sáu Tấn=Nguyễn Văn Tuấn, Bình Gà=Nguyễn Tấn Sầu}
{Ba Địa= Trần Mai Hoà, Sáu Tấn=Nguyễn Văn Tuấn}
Ba Địa,  Trần Mai Hoà
Sáu Tấn, Nguyễn Văn Tuấn
BUILD SUCCESSFUL (total time: 1 second)
```

What is Generics?

- A technique allows programmers creating general processes on data whose data types are not determined (generic is not used) or they can be determined (generic is used) when they are used.
- A way allows programmer implementing general algorithms which can be used to process multi-type input → Polymorphism.

Generic Classes in java.util

- Almost of interfaces and classes related to lists in the Java API declared as generic.
- Type Parameter Naming Conventions
  - By convention, type parameter names are single, uppercase letters.
  - The most commonly used type parameter names are:
    - E : Element/ K: Key
    - N – Number/ T - Type
    - V - Value
    - S,U,V etc. - 2nd, 3rd, 4th types

- java.lang.**Object**
  - java.util.**AbstractCollection**<E>
    - java.util.**AbstractList**<E>
      - java.util.**AbstractSequentialList**<E>
        - java.util.**LinkedList**<E>
      - java.util.**ArrayList**<E>
      - java.util.**Vector**<E>
        - java.util.**Stack**<E>
    - java.util.**AbstractQueue**<E>
      - java.util.**PriorityQueue**<E>
    - java.util.**AbstractSet**<E>
      - java.util.**EnumSet**<E>
      - java.util.**HashSet**<E>
        - java.util.**LinkedHashSet**<E>
      - java.util.**TreeSet**<E>
  - java.util.**AbstractMap**<K,V>
    - java.util.**EnumMap**<K,V>
    - java.util.**HashMap**<K,V>
      - java.util.**LinkedHashMap**<K,V>
    - java.util.**IdentityHashMap**<K,V>
    - java.util.**TreeMap**<K,V>
    - java.util.**WeakHashMap**<K,V>

Advantages of Generics

- Generics add stability to your code by making more of your bugs detectable at compile time.
- Generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods and limits on parametric types may be declared.
- Code that uses generics has many benefits over non-generic code.
  - Stronger type checks at compile time
  - Elimination of casts.
  - Enabling programmers to implement generic algorithms.

Generics are not used

- The package java.util supports general-purpose implementations which allows lists containing arbitrary elements
- The cost of this flexibility is we may have to use a casting operator when accessing an element.

Generic1.java *

```
1  import java.util.Vector;
2  class Person {
3      String name; int age;
4      Person(String n, int a)
5          { name=n; age=a; }
6      void print ()
7        { System.out.println( name + ", " + age);}
8  }
9  public class Generic1 {
10   public static void main(String[] args) {
11       Vector v = new Vector();
12       v.add (new Person("Hoa", 23));
13       v.add (new Person("Tuấn", 27));
14       for (int i= v.size()-1; i>=0; i--)
15           ((Person)(v.get(i))).print();
16   }
17 }
```

Output - Chapter08 (run)

```
run:
Tuấn, 27
Hoa, 23
BUILD SUCCESSFUL (total time: 0 seconds)
```

Generics are used

- If all elements of the collection are homogeneous(identical), the generic technique should be used.
- Generics add stability to your code by making more of your bugs detectable at compile time. Casting can not be used.

Generic1.java | Generic2.java

```
1  import java.util.Vector;
2  class Person2 {
3      String name; int age;
4      Person2(String n, int a)
5          { name=n; age=a; }
6      void print ()
7        { System.out.println( name + ", " + age); }
8  }
9  public class Generic2 {
10   public static void main(String[] args) {
11       Vector<Person2> v = new Vector<Person2> ();
12       v.add (new Person2("Hoa", 23));
13       v.add (new Person2("Tuấn", 27));
14       for (int i= v.size()-1; i>=0; i--)
15           v.get(i).print();
16   }
17 }
18
```

Output - Chapter08 (run)

```
run:
Tuấn, 27
Hoa, 23
BUILD SUCCESSFUL (total time: 1 second)
```

Using Generics- Syntax

- **Invoking and Instantiating a Generic Type**
  - *Box<Integer> integerBox = new Box<Integer>();*
- **The Diamond**
  - *Box<Integer> integerBox = new Box<>();*
- **Multiple Type Parameters**
  - *Pair<String, Integer> p1 = new OrderedPair<String, Integer>("Even", 8);*
- **Parameterized Types**
  - *OrderedPair<String, **Box<Integer>**> p = new OrderedPair<>("primes", new Box<Integer>(...));*

How generic class is treated?

- Compiler will save generic information in this class to class files (file.class)
- When this class is used (an object of this class is created)
  - If an argument types are declared: Compiler updates type information.
  - If no argument type is declared, type information in parameters are erased or changed to Object

Summary

- An ***abstract data type (ADT)****:* is a mathematical model for a data type that is determined based on
- Use the ***Collection framework:*** providing high-performance implementations of useful data structures and algorithms
- ***Generics*** add stability to your code by making more of your bugs detectable at compile time and elimination of casts.