

Collections

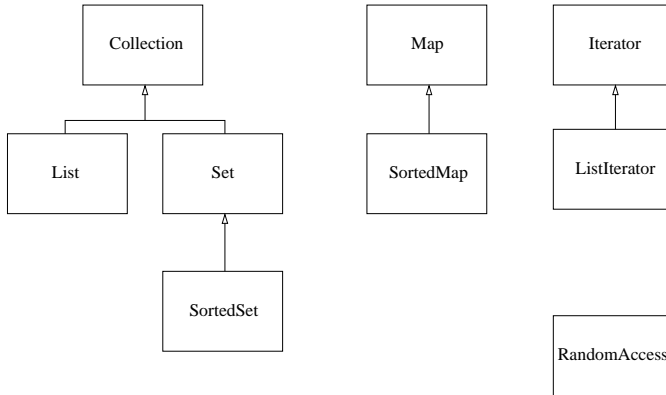
Tim Kovacs
kovacs@cs.bris.ac.uk

Frederik Vercauteren
frederik@cs.bris.ac.uk

• **Collections framework**: unified architecture for collection classes

- Collection **interfaces**: sets, lists and maps
- **Abstract** implementations: provides partial implementations
- **General-purpose** implementations
- **Algorithms**: useful functions on collections, e.g. sorting . . .
- **Legacy** implementations: older classes

Collections - Interfaces



Collection Interface

- **Modification** methods in the collection interfaces are **optional**
 - **UnsupportedOperationException** if they are attempted

```
Iterator iterator()
int size()
boolean isEmpty()
boolean contains(Object o)
boolean containsAll(Collection c)

boolean add(Object o) (optional)
boolean addAll(Collection c) (optional)
boolean remove(Object o) (optional)
boolean removeAll(Collection c) (optional)
boolean retainAll(Collection c) (optional)
void clear() (optional)

Object[] toArray()
Object[] toArray(Object[] a)
```

Map and SortedMap Interface

- **Map** interface: contains inner interface **Map.Entry**

```
boolean containsKey(Object key)
boolean containsValue(Object value)
Set keySet()
Set entrySet()
Collection values()
Object get(Object key)
Object remove(Object key) (optional)
Object put(Object key, Object value) (optional)
void putAll(Map t) (optional)
int size()
boolean isEmpty()
void clear() (optional)
```
- **SortedMap** interface: extra functionality involving **comparisons**

```
Comparator comparator()
Object firstKey()
Object lastKey()
SortedMap headMap(Object toKey)
SortedMap tailMap(Object fromKey)
SortedMap subMap(Object fromKey, Object toKey)
```

Collections - Interfaces

- **Collection**: group of objects
 - No assumptions about the order or about duplicate elements
- **List**: ordered collection, also known as a **sequence**
 - Duplicates are generally permitted; allows positional access
- **Set**: familiar set abstraction
 - No duplicate elements permitted
- **SortedSet**: set whose elements are automatically sorted
 - Using **Comparable** interface or by a **Comparator** object
- **Map**: mapping from **keys to values**
 - Each key can map to at most one value.
- **SortedMap**: map automatically sorted by key
 - Using **Comparable** interface or by a **Comparator** object

List and Set Interface

- **List** interface: extra **positional** versions of collection interface

```
Object get(int index)
Object remove(int index) (optional)
Object set(int index, Object element) (optional)
List sublist(int fromIndex, int toIndex)
void add(int index, Object element) (optional)
boolean addAll(int index, Collection c) (optional)
int indexOf(Object o)
int lastIndexOf(Object o)
ListIterator listIterator()
ListIterator listIterator(int index)
```

- **Set** interface: same methods as **Collection** interface
 - Different behaviour, since **no duplicates** are allowed
 - Influences: **add**, **addAll**, **equals**, **hashCode**

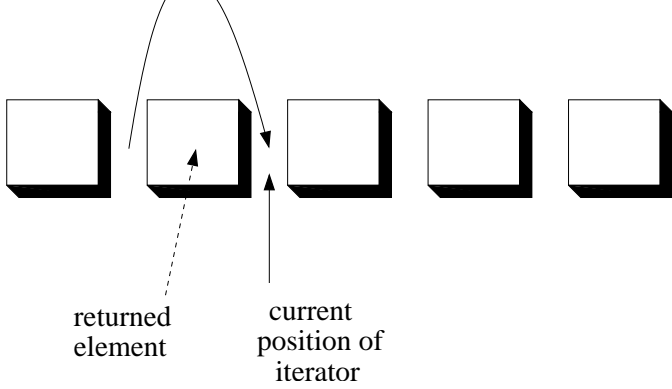
Iterator Interface

- **Iterator**: visit elements in Collection one by one
 - **remove** method removes element returned by **next**

```
boolean hasNext()
Object next()
void remove() (optional)
```

- **ListIterator**: iterator for use with lists
 - **bi-directional**, index retrieval, element replacement, . . .

```
void add(Object o) (optional)
boolean hasPrevious()
int nextIndex()
Object previous()
int previousIndex()
void set(Object o) (optional)
```



```

Iterator iter = c.iterator();
while (iter.hasNext())
{
    Object obj = iter.next();
    // do something with obj
}

// Removing elements from collection

iter = c.iterator();
iter.next(); // jump passed first element
iter.remove(); // remove first element of list

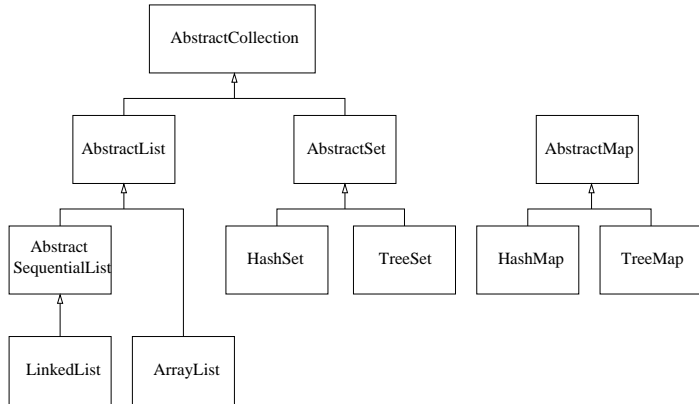
// can't call remove() again before calling next()

iter.next(); // passing new first element
iter.remove(); // removing new first element

```

Collections - Classes

Collections - Classes



- Names of classes typically is **<Implementation-style><Interface>**

	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

- Support **all** of the **optional operations** in the collection interfaces
- No restrictions on the elements they may contain
- Abstract classes provide skeletal implementations of the core collection interfaces

Collections - Lists

Collections - Sets

- ArrayList**: **resizable-array** implementation of the List interface
 - Best all-around implementation of the List interface
 - Implements **RandomAccess** interface (SDK 1.4)

```

ArrayList()
ArrayList(Collection c)
ArrayList(int initialCapacity)
void ensureCapacity(int minCapacity)
void trimToSize()

```

- LinkedList**: **doubly-linked list** implementation of List interface
 - Better than ArrayList if elements frequently inserted or deleted

```

LinkedList()
LinkedList(Collection c)
void addFirst(Object o)
void addLast(Object o)
Object getFirst()
Object getLast()
Object removeFirst()
Object removeLast()

```

- Hash table**: array of linked lists called buckets
 - Computes hash code of object, used as index in array
 - Very efficient to find object in hash table
 - Need to redefine **equals** and **hashCode** of **Object** class
 - hashCode** should return **int** (will be reduced mod # buckets)

- HashSet**: **hash table** implementation of the Set interface
 - Best all-around implementation of the Set interface
 - Iterator** returns elements in **random order**

```

HashSet()
HashSet(Collection c)
HashSet(int initialCapacity)
HashSet(int initialCapacity, float loadFactor)

```

Collections - Sets

Object Comparison

- LinkedHashSet**: **hash table + linked list** implementation of Set
 - Insertion-ordered Set that runs nearly as fast as HashSet
 - Iterator** returns elements in insertion order

```

LinkedHashSet()
LinkedHashSet(Collection c)
LinkedHashSet(int initialCapacity)
LinkedHashSet(int initialCapacity, float loadFactor)

```

- TreeSet**: **red-black tree** implementation of SortedSet interface
 - Iterator** returns elements in sorted order
 - Slower than **hashSet**, but elements are ordered

```

TreeSet()
TreeSet(Collection c)
TreeSet(Comparator c)
TreeSet(SortedSet s)

```

- Ordered** collections need to **compare** objects to sort them
- Default**: class implements **Comparable** interface
 - Class provides **int compareTo(Object other)** method
 - Problem: only one such method for a given class
 - What happens if class does not implement **Comparable**

- Flexible**: pass object that implements **Comparator** interface

```

public interface Comparator
{
    int compare(Object o1, Object o2);
}

```

- Defined and created using **anonymous class** (see example p.21)

```
Object getKey()
Object getValue()
Object setValue(Object value)
```

- **HashMap**: **hash table** implementation of the Map interface
 - Best all-around implementation of the Map interface

```
HashMap()
HashMap(int initialCapacity)
HashMap(int initialCapacity, float loadFactor)
HashMap(Map m)
```

- **TreeMap**: **red-black tree** implementation of SortedMap interface

```
TreeMap()
TreeMap(Comparator c)
TreeMap(Map m)
TreeMap(SortedMap m)
```

```
// remove an entry
staff.remove("567-24-2546");

// replace an entry
staff.put("456-62-5527", new Employee("Francesca Miller"));

// iterate through all entries
Set entries = staff.entrySet();
Iterator iter = entries.iterator();
while (iter.hasNext())
{
    Map.Entry entry = (Map.Entry)iter.next();
    Object key = entry.getKey();
    Object value = entry.getValue();
    System.out.println("key=" + key + ", value=" + value);
}
```

Collections - Algorithms

- **Generic collection interfaces**
 - Only need to implement algorithms **once**
 - Select **minimal** collection **interface** (highest in the hierarchy)
 - Use provided methods to implement algorithm

```
public static Object max(Collection c)
{
    if (c.isEmpty()) throw new NoSuchElementException();
    Iterator iter = c.iterator();
    Comparable largest = (Comparable) iter.next();
    while (iter.hasNext())
    {
        Object next = iter.next();
        if (largest.compareTo(next) < 0)
            largest = next;
    }
    return largest;
}
```

Collections - Algorithms

- Example of **sorting** list using **Comparator** object
 - Notice the use of an anonymous class

```
List staff = new LinkedList();
// add some stuff to the linked list

Collections.sort(staff, new
Comparator()
{
    public int compare(Object a, Object b)
    {
        double salaryDifference = ((Employee)a).getSalary()
            - ((Employee)b).getSalary();

        if (salaryDifference < 0) return -1;
        if (salaryDifference > 0) return 1;
        return 0;
    }
});
```

Collections - Algorithms

- **Collections class** (with **s**) contains useful methods

```
static int binarySearch(List list, Object key)
static int binarySearch(List list, Object key, Comparator c)
static void copy(List dest, List src)
static void reverse(List list)
static void shuffle(List list)
static void sort(List list)
static void sort(List list, Comparator c)
static void swap(List list, int i, int j)
```

Legacy Collections

- Classes that existed since the beginning
 - **Hashtable** and **Properties** subclass
 - **Vector** and **Stack** subclass
 - **BitSet** class
- **Hashtable** and **Vector** have **synchronized** methods
 - Shouldn't use these classes unless need synchronization
 - Use **HashMap** and **ArrayList** instead
- Use **Enumeration** instead of **Iterator** interface
 - **hasMoreElements** instead of **hasNext**
 - **nextElement** instead of **next**