CS61B Lecture #18

Today:

- Overview of standard Java Collections classes.
  - Iterators, ListIterators
  - Containers and maps in the abstract
  - Views

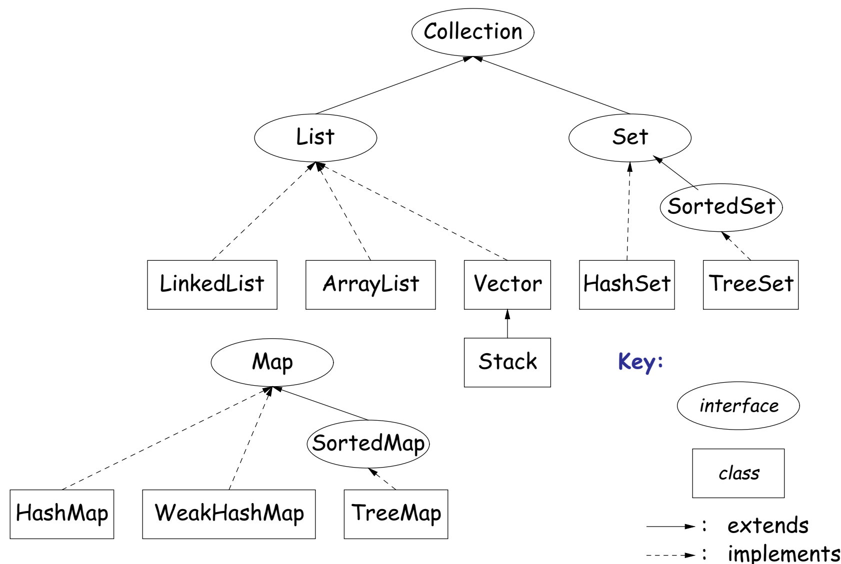
Readings for Today: *Data Structures*, Chapter 2.

Readings for next Topic: *Data Structures*, Chapter 3.

Data Types in the Abstract

- Most of the time, should *not* worry about implementation of data structures, search, etc.
- What they do for us—their specification—is important.
- Java has several standard types (in `java.util`) to represent collections of objects
  - Six interfaces:
    - `Collection`: General collections of items.
    - `List`: Indexed sequences with duplication
    - `Set`, `SortedSet`: Collections without duplication
    - `Map`, `SortedMap`: Dictionaries (key $\mapsto$ value)
  - Concrete classes that provide actual instances: `LinkedList`, `ArrayList`, `HashSet`, `TreeSet`.
  - To make change easier, purists would use the concrete types only for **new**, interfaces for parameter types, local variables.

Collection Structures in java.util



The Collection Interface

- Collection interface. Main functions promised:
  - Membership tests: `contains` ($\in$), `containsAll` ($\subseteq$)
  - Other queries: `size`, `isEmpty`
  - Retrieval: `iterator`, `toArray`
  - *Optional* modifiers: `add`, `addAll`, `clear`, `remove`, `removeAll` (set difference), `retainAll` (intersect)
- Design point (a side trip): Optional operations may throw

  `UnsupportedOperationException`

- An alternative design would have separate interfaces:

```
interface Collection { contains, containsAll, size, iterator, ... }
interface Expandable { add, addAll }
interface Shrinkable { remove, removeAll, difference, ... }
interface ModifiableCollection
   extends Collection, Expandable, Shrinkable { }
...
```

You'd soon have lots of interfaces. Perhaps that's why they didn't do it that way.)

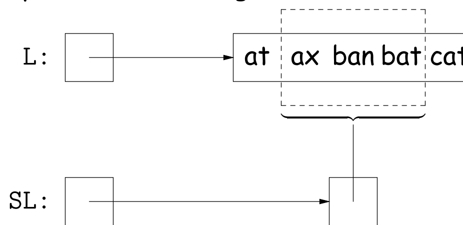
The List Interface

- Extends Collection
- Intended to represent *indexed sequences* (generalized arrays)
- Adds new methods to those of Collection:
  - Membership tests: indexOf, lastIndexOf.
  - Retrieval: get(i), listIterator(), sublist(B, E).
  - Modifiers: add and addAll with additional index to say *where* to add. Likewise for removal operations. set operation to go with get.
- Type ListIterator<Item> extends Iterator<Item>:
  - Adds previous and hasPrevious.
  - add, remove, and set allow one to iterate through a list, inserting, removing, or changing as you go.
  - **Important Question:** What advantage is there to saying List L rather than LinkedList L or ArrayList L?

Views

New Concept: A *view* is an alternative presentation of (interface to) an existing object.

- For example, the sublist method is supposed to yield a "view of" part of an existing list:




```
List<String> L = new ArrayList<String>();
L.add ("at"); L.add("ax"); ...
List<String> SL = L.sublist (1,4);
```

- Example: after L.set(2, "bag"), value of SL.get(1) is "bag", and after SL.set(1,"bad"), value of L.get(2) is "bad".
- Example: after SL.clear(), L will contain only "at" and "cat".
- Small challenge: "How do they *do* that?!"

Maps

- A Map is a kind of "modifiable function:"

```
package java.util;
public interface Map<Key,Value> {
  Value get (Object key);               // Value at KEY.
  Object put (Key key, Value value);    // Set get(KEY) -> VALUE
  ...
}
------------------------------------------------------
Map<String,String> f = new TreeMap<String,String> ();
f.put ("Paul", "George"); f.put ("George", "Martin");
f.put ("Dana", "John");
// Now f.get ("Paul").equals ("George")
//     f.get ("Dana").equals ("John")
//     f.get ("Tom") == null
```


Map Views

```
public interface Map<Key,Value> { // Continuation
           /* VIEWS */
  /** The set of all keys. */
  Set<Key> keySet ();
  /** The multiset of all values */
  Collection<Value> values ();
  /** The set of all (key, value) pairs */
  Set<Map.Entry<Key,Value>> entrySet ();
}
```

Using example from previous slide:

```
for (Iterator<String> i = f.keySet ().iterator (); i.hasNext ();)
   i.next ()  ===>  Dana, George, Paul
// or, just:
for (String name : f.keySet ())
   name  ===>  Dana, George, Paul

for (String parent : f.values ())
   parent ===>  John, Martin, George
for (Map.Entry<String,String> pair : f.entrySet ())
   pair  ===>  (Dana,John), (George,Martin), (Paul,George)
f.keySet ().remove ("Dana");   // Now f.get("Dana") == null
```