COP 3330, Spring 2013
Intro to Collections

Instructor : Arup Ghosh
02-11-13

School of Electrical Engineering and Computer Science
University of Central Florida

Collections

- A collection is an object used to handle groups of data items
- Java provides a number of useful classes whose objects can be used as containers
- An ArrayList is basically a growable array
- A TreeSet is a sorted collection that doesn't allow duplicates
- A TreeMap is basically an associative array

Collections

- Basically, reusable data structures that reduce coding effort significantly.

Generics

- New to 5.0
- Equivalent of templates in C++
- You can create ArrayLists, etc. of a specific type
- Example: Create an ArrayList of complex numbers
  - `ArrayList<Complex> foo = new ArrayList<Complex>();`

ArrayLists

- Useful ArrayList<E> methods:
  - `add(E foo)` – Adds foo to the end of the ArrayList
  - `get(int index)` – Equivalent of [ ] in an array for access
  - `set(int index, E foo)` – Changes the element at the specified index
  - `remove(int index)` – Removes the element at the specified index
  - `size()` – Returns the number of elements in the ArrayList

Primitives?

- We frequently want to put primitives into collections.
  - E.g., a list or a set of integers is a very natural idea.
- Unfortunately, Collections can only take **classes** as type parameters!
  - This rules out primitives.
- To get around this, Java defines *wrapper* classes for each primitive.
  - `Integer, Long, Float, Double, Character, Boolean, Byte, Short`.

Wrappers

- A fairly obvious idea. If we can’t put an int into a collection, define a *class* called Integer to play the role of an int.
- Integer just contains an int, and a bunch of methods like add(), multiply(), compareTo(), etc., that give it the same behavior as an int.
- There is a similar wrapper class for every primitive. Whenever you need an object but have a primitive, these will conveniently fill in. ☺

Old usage

- Before Java 5, we had to explicitly transform primitives into their wrappers and back again.

- `Integer x = new Integer(42);`
- `int y = x.intValue();`

- Tedious and annoying, as you might expect.

- Java 5 got rid of this with a neat idea called *autoboxing*.

Autoboxing

- Using a wrapper is like taking a primitive and putting it in a box. Transforming it back is like removing it from the box.
  - That's why they're *wrappers*, after all.

- Java 5 introduced autoboxing, which means that the conversions between object and primitive happen invisibly, without explicit code.

- For situations like method parameters and assignment statements, you can use the primitive or its wrapper interchangeably.
  - Keep in mind that you still can't call methods on the primitive, or use arithmetic/logical operators on the wrapper.

Usage

Note: NOT <int>, but <Integer>

```
ArrayList<Integer> li = new ArrayList<Integer>();

for(int i = 0; i < 10; i++) {
        int randVal = (int)(100 * Math.random());

        li.add(randVal);
}
```

A random **int**

Autoboxed into an Integer, because that's what add() expected

Example

- ArrayList Example

TreeSets

- A TreeSet is a sorted container based on red-black trees
- Only one of each element is allowed
- Provides O(log n) insertion, access, and deletion
- The type of the TreeSet must implement `Comparable`
- Example:

  -
    ```
    TreeSet<Integer> ts = new
    TreeSet<Integer>();
    ```

TreeSets

- Useful methods in TreeSet<E>:
  - `add(E foo)` – Adds foo in sorted order
  - `remove(E bar)` – Deletes bar
  - `contains(E bar)` – Returns `true` if the set contains `bar`, `false` otherwise
  - `size()` – Returns the number of elements in the set

Iterator `for` Loop

- Notice that TreeSet, ArrayList, and others implement `Iterable`
- This means that you can use a special type of `for` loop to go through them
- If you have a `TreeSet<E> set`, you can loop through all the elements using:

```
for(E bar : set) {
    // Do something with bar
}
```

Iterator `for` Loop

- Example: Print 25 random `double`s in sorted order

```
TreeSet<Double> set = new TreeSet<Double>();
while(set.size() < 25)
{
     set.add(Math.random());
}
for(Double doub : set)
{
     System.out.println(doub);
}
```

TreeMaps

- A TreeMap provides a mapping between objects
- One way to think of it is as an array that can be indexed by any type you want (An associative array)
- Found in `java.util.*`

TreeMaps

- A TreeMap needs two types: a key type and a value type
- Example: Create a mapping from Strings to Doubles

  -
    ```
    TreeMap<String,Double> foo = new
      TreeMap<String,Double>();
    ```

TreeMaps

- Useful methods in TreeMap<K,V>
  - `put(K key, V value)` – Makes `key` map to `value`
  - `containsKey(Object key)` – Returns `true` if `key` maps to a value
  - `get(Object key)` – Returns the value that `key` maps to, if any. Returns `null` if there is no mapping for `key`
  - `keySet()` – Returns the set of keys that have mappings

Example

- TreeMap

Example

- TreeSet