CSE1720

Week 08, **Class Meeting 19** (Lecture 16)

Winter 2013 ◆ Tuesday, Feb 25, 2013

YORK UNIVERSITÉ UNIVERSITY

In General…

- Welcome back from reading week!
  - Hopefully you found some rest…
  - … and also had a chance to do some solid coursework so you can finish the term in a strong way.
- Course reading schedule:
  - deadline for reading Chapter 10 is Feb 26th (today)
- Next lab test:
  - Thursday Feb 28; Friday Mar 01
  - **Topic**: adding the functionality as specified in Week 06 lab exercises

YORK UNIVERSITÉ UNIVERSITY

In General…

- Check-in for course work
  - Lab test: marking in progress
  - Term test: marking in progress
  - Week 05 Lab Exercises: 5 students did not submit at all
  - Week 06 In-class Exercises:

YORK UNIVERSITÉ UNIVERSITY

Goals for today's class meeting

- Questions about week 06 lab exercise?
  - adding movement to game shooter
- Discussion of different game functionalities
  - and the data structures to support them!!!

YORK UNIVERSITÉ UNIVERSITY

Motivation

- You need to understand **the collection framework** in order to implement and to analyze any type of non-trivial application.
- The collection framework is specific to Java, but it relates to a more general concept in computer science.
  - Abstract data type (ADT) : mathematical models for certain types of data structures, used in order to describe and analyze abstract algorithms.
- The collection framework is just Java's implementation of certain ADTs.

YORK UNIVERSITÉ UNIVERSITY

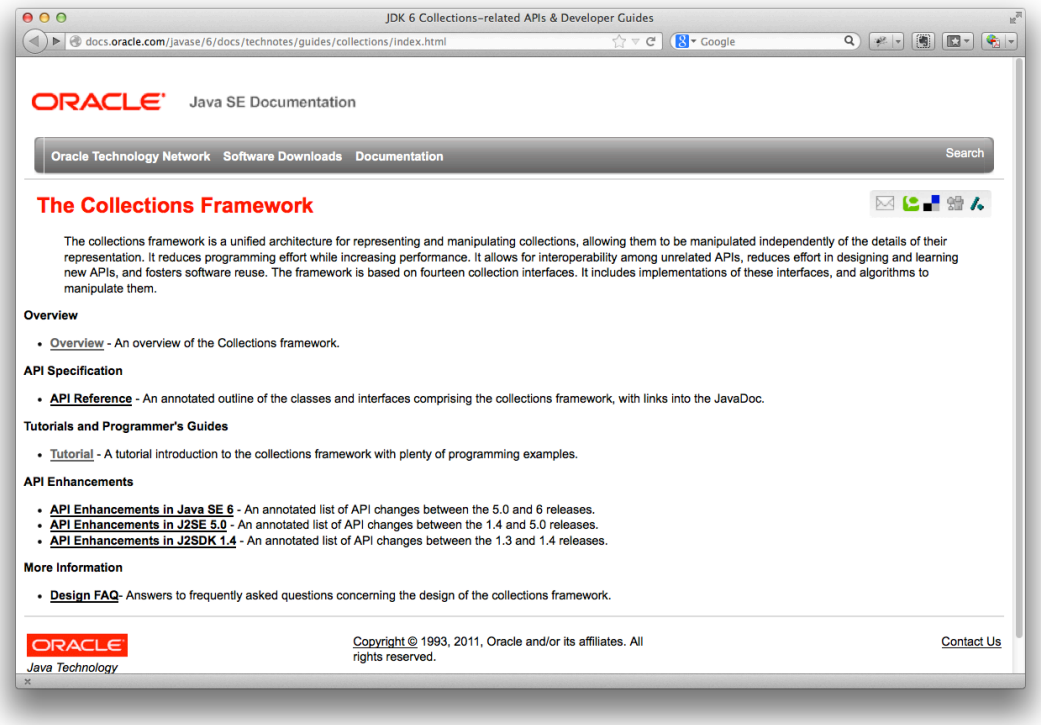
About Abstract Data Types…

- Examples of ADTs:
  - Collection*, Deque, List*, Map*, Queue, Set*, Stack, Tree, … (among others)
  - *indicates an ADT implemented in the Java platform throught the Collections Framework
- an ADT is described solely in terms of:
  - the operations that may be performed on it (sound familiar?)
  - the mathematical constraints on the effects of the operations (e.g., insertion should require additional memory, etc)
- Most programming language provide implementations for most or all of the ADTs;
  if not, you need to write them!

YORK UNIVERSITÉ UNIVERSITY

Java SE Documentation for the Collections Framework

http://docs.oracle.com/javase/6/docs/technotes/guides/collections/index.html



YORK UNIVERSITÉ UNIVERSITY

Chapter 10

- Provocation:
  is a **variable** considered to be a **data structure**?

YORK UNIVERSITÉ UNIVERSITY

Recap and Discussion

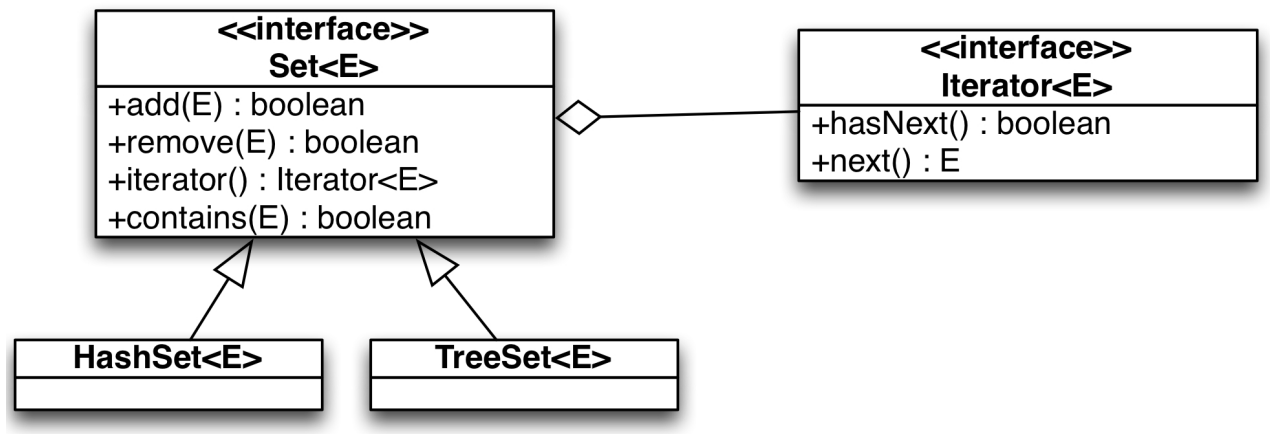
- We will discuss and distinguish among:
  - When should a **set** be used?
  - When should a **list** be used?
  - When should a **collection** be used?
  - When should a **map** be used?

We will distinguish between “use of services” for the purposes of *declaration* vs for the purposes of *instantiation*

9

YORK UNIVERSITÉ UNIVERSITY

The `Set<E>` Interface



10

YORK UNIVERSITÉ UNIVERSITY

The `Set<E>` Interface

- The interface is `Set<E>`
- Implementing classes are `HashSet<E>` and `TreeSet<E>`

```
Set<String> s = new HashSet<String>();
Set<String> s = new TreeSet<String>();
```

Declaration — Instantiation

YORK UNIVERSITÉ UNIVERSITY

The `Set<E>` Interface

- The `Set` interface is “generic”, which is indicated by the < and > in the interface name.
  - If you want to use the `Set` interface or the `HashSet` or `TreeSet` classes, you need to specify the type of the elements by writing it between < and >
- By doing this, the client ensures:
  - No rogue element can be inserted
  - No casting is needed upon retrieval

YORK UNIVERSITÉ UNIVERSITY

HashSet<E> vs TreeSet<E>

Suppose your set contains 128 elements, ($\log_2 128=7$)

- If you use a `HashSet<E>`, then
  - it will take 1 step to **add** an additional element
  - it will take 1 step to **remove** an element
  - it will take 1 step to **test whether a given element is found** within the set
- If you use a `TreeSet<E>`, then:
  - it will take 7 steps to **add** an additional element
  - it will take 7 steps to **remove** an additional element
  - it will take 7 steps to **test whether a given element is found** within the set

YORK UNIVERSITÉ UNIVERSITY

HashSet<E> vs TreeSet<E>

- If you use a `HashSet<E>`, then
  - the iterator will provide the elements in some sort of order that may or may not be sorted
- If you use a `TreeSet<E>`, then:
  - the iterator will provide the elements in a sorted order

*but wait – didn't we say that sets are **not sorted**?*

Yes, that's correct. The API doesn't require this, it just happens to be a kind of "bonus" of the TreeSet implementation

YORK UNIVERSITÉ UNIVERSITY

Pros and Cons…

Version #1

```
HashSet<String> s = new HashSet<String>();
TreeSet<String> s = new TreeSet<String>();
```

Version #2

```
Set<String> s = new HashSet<String>();
Set<String> s = new TreeSet<String>();
```

Discuss implication of versions #1 and #2

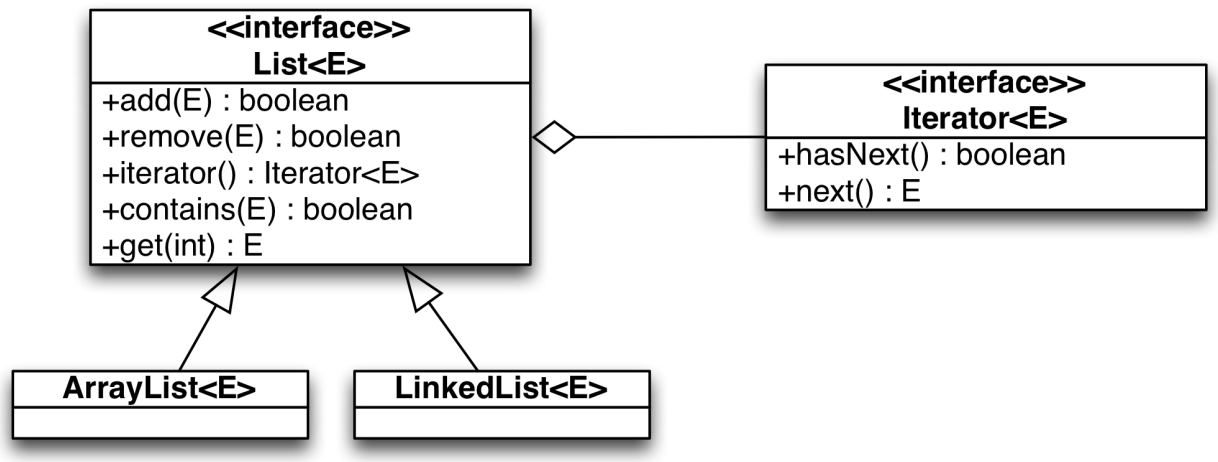
YORK UNIVERSITÉ UNIVERSITY

Best Practises

- *Declaration* as high up the hierarchy as possible
- *Instantiation* lower in the hierarchy

YORK UNIVERSITÉ UNIVERSITY

The `List<E>` Interface



YORK UNIVERSITÉ UNIVERSITY

`ArrayList<E>` vs `LinkedList<E>`

Suppose your list contains 128 elements ($\log_2 128=7$)

- If you use a `ArrayList<E>`, then
  - it will take 1 step to **get** an element
  - it may take up to 128 steps to **add** an additional element
  - it will take 128 steps to **remove** an element
- If you use a `LinkedList<E>`, then:
  - it will take 128 steps to **get** an element
  - it will take 1 step to **add** an additional element
  - it will take 7 steps to **remove** an additional element

`LinkedList<E>` is better if you need to add or remove elements

YORK UNIVERSITÉ UNIVERSITY

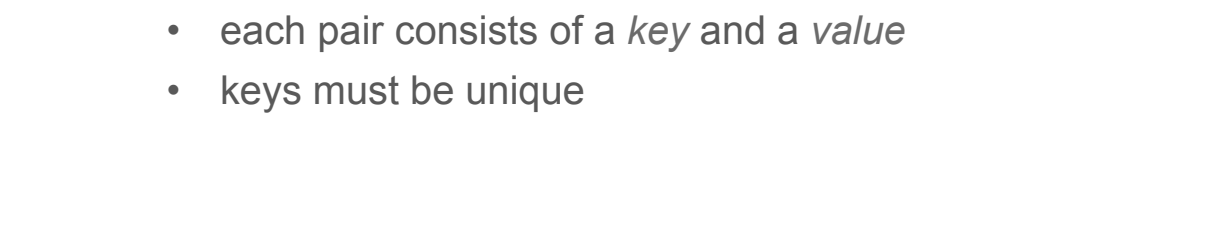
Discussion about Maps

- A Map is a kind of *generalized* collection
  - Sets – the elements are *objects*
    - all elements are unique (no duplicates)
    - elements are not ordered in any way
  - Lists – the elements are *objects*
    - albeit with possible duplicates
    - elements have an ordered defined
  - Map – the elements are *pairs of objects*
    - each pair consists of a *key* and a *value*
    - keys must be unique

YORK UNIVERSITÉ UNIVERSITY

Discussion about Maps

- a pair has two elements: the **key** and the **value**
- a dictionary is an example of a map
  - a dictionary consists of a list of pairs
  - the keys are sorted in lexicographic order



YORK UNIVERSITÉ UNIVERSITY

Discussion about Maps

- When declaring a map, we need to specify:
  - the type of the keys
  - the type of the values

```
Map<String, String> theDictionary;
Map<String, Integer> theTally;
```


Discussion about Maps

- The basic operations
  - `put` a pair into the `Map`
  - `remove` a pair
  - given a key, `get` its corresponding value
  - iterate *over the keys*
  - iterate *over the values*