CSE 142

Unordered Collections:
Sets and Maps

11/12/2003   (c) 2001-3, University of Washington   P2-1

Introduction

- **Quick Review:**
  - Ordered vs. Unordered collections
  - an Ordered Collection: ArrayList
- **Today:**
  - an Unordered Collection: Set (HashSet)
  - an Unordered Collection: Map (HashMap)
- **Reading**
  - Use these slides and JavaDoc for reference
  - Not covered in N&H textbook!
- **A word of assurance... this will not be on the midterm!**

11/12/2003   (c) 2001-3, University of Washington   P2-2

Ordered vs. Unordered Collections

- **Some collections have a natural order of their elements:**
  - the steps in a recipe
  - the list of daily weather observations
  - the list of shapes to be drawn (with later shapes layered over earlier shapes)
  - actors in a play "in order of appearance"
- **Lists (e.g. ArrayList) are good for these collections.**
- **Some collections don't have any obvious natural order:**
  - the ingredients in a recipe
  - the stars in the sky
  - the merchandise at Freddy Meyer's
- **ArrayLists are not ideal for these collections.**

11/12/2003   (c) 2001-3, University of Washington   P2-3

Sets

- **Sets in mathematics are collections...**
  - without any particular order in the elements
  - without duplicates
- **if you have a set and try to add to it something already in the set...**
    - It's not an error
    - the set remains unchanged
- **List or Set?**
  - Subscribers to a magazine (hint: what happens when they renew?)
  - Partners in the IPL waiting for a consultant on the day the homework is due
  - People at CLUE discussing Java
  - Colors of the rainbow

11/12/2003   (c) 2001-3, University of Washington   P2-4

Sets in Java

- You could define your own class and base it on an ArrayList, or...
- You could use directly one of two set implementations in java.util
- Classes are named *HashSet* and *TreeSet*
- Strange names, indeed
  - We'll explain and discuss in CSE143
  - In CSE142 all our examples are with *HashSet*
- Many HashSet methods are similar to methods of ArrayList
  - Not accidental! More when we discuss "inheritance"

HashSet Methods

- These should look *very* familiar!
- A partial interface:

```
public class HashSet {
    // Add the argument object to the set, if it wasn't there already
    public boolean add(Object obj);
    // Return whether the argument object is an element of the set
    public boolean contains(Object obj);
    // Remove the argument object from the set, if it was present
    public boolean remove(Object obj);
    // Return the number of elements in the set
    public int size( );
    // Return an iterator that will go through all the set's elements, in some order
    public Iterator iterator( );
    …
}
```

Using HashSets

```
HashSet set = new HashSet( );
set.add("Parsley");     set.add("Sage");     set.add("Oregano");
set.add("Rosemary");  set.add("Thyme");              // draw the picture!
int count = set.size( );                             // what is count?
set.remove("Oregano");                               // what is the picture now?
if (set.contains("Arsenic")) {
    System.out.println("Beware!");
}
Iterator iter = set.iterator( );
while (iter.hasNext( )) {
    String ingredient = (String) iter.next( );
    System.out.println(ingredient);
}                                                    // what is printed?
```

A Problem: Who Gets the Scholarship?

- A famous philanthropist want to give a scholarship to everyone who has taken CSE142 or MA126. He has a (long) list of each group's members. Who gets the scholarships?

A: Think of an algorithm which takes the two lists and produces a list with each person's name in it only once.

B: Think of a simpler algorithm which produces a set with each person in it only once.

Beyond Lists and Sets:
Keyed Collections

- Some collections have a way to look up each element, using the element's *key*.
- For example:
  - Each CD in a music collection could be looked up by title.
    The title is the key
  - Each student in the class could be looked up by name, or by student ID.
    The name could be the key, or the ID could be the key
  - Each entry in the dictionary can be looked up by the word the entry defines.
    The word could be the key
- A collection that links keys to data is called a *map*, or sometimes a *table* or a *dictionary*.

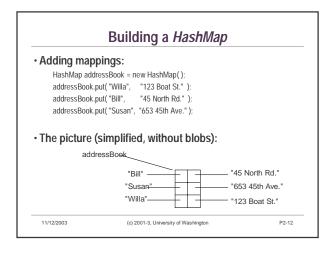11/12/2003   (c) 2001-3, University of Washington   P2-9

Maps In Java

- For a map to be possible, each item must have a key
  - Each key must be unique!
    Cannot have two different entries with the same key.
  - Not always true in real life
    so we often have to invent unique keys for things. (Can you think of any examples?)
- To implement a map...
  - you could use a single array or *ArrayList* or *HashSet*, and search it to find the element with a given key, or...
  - you could use one of the existing classes in java.util: *HashMap* or *TreeMap*
- We will describe and use *HashMap* in CSE142
  - Take CSE143 to find out about the funny name
  - Take a database course to learn even more about keys and fancy ways of organizing data for effective storage and retrieval
- Footnote for users of Perl, JavaScript, etc:
  - Many such languages have "associative arrays": same basic concept as a Map

11/12/2003   (c) 2001-3, University of Washington   P2-10

Class *HashMap*

- The methods are sometimes different from the List and Set interface
- Note that keys must be Objects (inconvenient sometimes)
- Values are also Objects

```
public class HashMap {
    // Return the number of key/value pairs in the map
    public int size();

    // Make key map to value in the map
    // (either by adding a new mapping or by changing what key maps to)
    public Object put(Object key, Object value);
    // Return the value that key maps to, or null if it isn't in the map
    public Object get(Object key);

    // Return whether the argument object is a key of the map
    public boolean containsKey(Object key);
    // Return whether the argument object is a value in the map
    public boolean containsValue(Object value);
    // Remove key and the value it maps to from the map, if it was present
    public boolean remove(Object key);
    ...
}
```

11/12/2003   (c) 2001-3, University of Washington   P2-11

Building a *HashMap*

- Adding mappings:

```
HashMap addressBook = new HashMap();
addressBook.put( "Willa",   "123 Boat St." );
addressBook.put( "Bill",    "45 North Rd." );
addressBook.put( "Susan",   "653 45th Ave." );
```

- The picture (simplified, without blobs):

11/12/2003   (c) 2001-3, University of Washington   P2-12

Examining a *HashMap*

```
HashMap addressBook = …;
…
String addr1 = (String) addressBook.get("Bill");        // what is addr1?
String addr2 = (String) addressBook.get("Bobbie");      // what is addr2?

if (addressBook.containsKey("Susan")) {
    System.out.println((String) addressBook.get("Susan"));
}

addressBook.remove("Willa");            // what does the picture look like
now?

// Bill moves in with Susan:
addressBook.put("Bill", addressBook.get("Susan"));      // what is the picture
now?
```

11/12/2003   (c) 2001-3, University of Washington   P2-13

null

- Recall: in Java, there is a special value, *null*, which is used to represent "nothing" or "undefined."
  - Instance variables are initialized by default to *null*.
  - Used only with references (objects), never with elementary types
- Many collection methods return *null* to mean that no such object exists.

```
HashMap notesToMyself = new HashMap ();
…
String task = (String) notesToMyself.get("Most Important To-Do Item");
if (task == null) {
    System.out.println("Nothing to do; go play!");
} else {
    System.out.println("Get busy on " + task);
}
```

11/12/2003   (c) 2001-3, University of Washington   P2-14

More HashMap Methods

```
public class HashMap {
    …
    // Return a Set (the interface of HashSet) of the keys of the map
    public Set keySet( );

    // Return a Collection (the interface of all collections) of the values of the map
    public Collection values( );
    …
}
```

11/12/2003   (c) 2001-3, University of Washington   P2-15

Iterating through a HashMap

- Maps in Java do not have iterators (at least, not directly)
- To iterate through a map you can either...
  - get the set of keys, and then iterate through them.
  - get the set of values, and then iterate through them.

```
HashMap musicCollection = …;
…
Set titles = musicCollection.keySet( );             // get the set of keys
Iterator iter = titles.iterator( );                 // get an iterator on the keys
while (iter.hasNext( )) {
    String title = (String) iter.next( );           // get the next key
    CD disk = (CD) musicCollection.get(title);      // lookup the key
    System.out.println("Now playing " + title);
    disk.play( );
}
```

11/12/2003   (c) 2001-3, University of Washington   P2-16