Java Collections
Iterators
Chris La Pilla
CompSci 6
Summer 2007

Collections
• We’ve previously seen arrays and ArrayLists, which store elements in memory using contiguous blocks of memory
• These are examples of data structures, which are ways of organizing how data is stored in memory
• More abstractly, we can talk about collections, which are containers that hold multiple values (or elements) according to certain rules

Collections
• There are different types of collections, with their own rules for how objects are stored, e.g.:
  – ordered/unordered?
  – duplicates permitted?
• Primary types of collections we’ll discuss are lists, sets, and maps
• A particular type of collection may have many implementations; that is, we can use different data structures for the same collection

Java Collections Framework
• Java API includes many classes and interfaces related to collections in the java.util package (need to import)
• There are interfaces for each type of collection; this allows you to write code that will work with any underlying implementation
• Concrete classes provide alternative implementations for each collection type
  – We could treat implementations as a black boxes
  – However, every implementation has tradeoffs in efficiency (speed, memory)
• In this course, we will not discuss implementations in great detail (covered in CompSci 100)

Types of collections
• List: a sequence of values in a fixed ordering
• Set: a collection of values that cannot contain duplicate elements (models a mathematical notion of set)
• Map: an association of elements of one set (keys) to elements of another set (values)

Lists
• A sequence of values in a fixed ordering
  – Can add/remove values
  – Can get/set values at a particular position
• The List<T> interface represents an abstract notion of a list that contains objects of type T
• ArrayList is one particular implementation of List that uses partially filled arrays
• An alternate implementation is LinkedList, but we will not discuss this
Sets

- A collection of values that cannot contain duplicate elements
- A particular value is either a member of the set or not
- There is no particular ordering on elements added to a set
- There is an interface Set<T> that represents an abstract set of objects of type T
- TreeSet and HashSet are two implementations available

Sets

- When declaring variables, you should use the Set interface type
- This way, if you need to change implementation later, you only need to change the way the set object is constructed

```
Set<String> words = new TreeSet<String>();
```

Common set operations

- add: adds a value if it is not already present
- remove: removes a value from the set
- contains: checks whether a value is in the set
- iteration over all elements
- See Set interface in Java API Specification for details on each method

```
Set<String> words = new TreeSet<String>();
words.add("hello"); // {hello}
words.add("goodbye"); // {goodbye, hello}
words.add("hello"); // no change, already present
System.out.println(words.contains("hello")); // true
words.remove("hello"); // {goodbye}
System.out.println(words.contains("hello")); // false
```

Set iterator

- Unlike lists, each element in a set does not have a corresponding index
- How do we iterate over all elements in the set?
- Must use Iterator objects

```
SetIterator words = new TreeSet<String>();
```

Iterators

- An Iterator is an object that is used to traverse the elements in a collection
- Iterator<T> is an interface (see API) with two methods:
  - next(): returns the next element in the collection
  - hasNext(): returns whether there are more elements remaining
- Sets and Lists have a method iterator() which returns an Iterator object which can be used to iterate over the collection
Iterator Example

Set<String> words = new TreeSet<String>();
words.add("hello"); // {hello}
words.add("goodbye"); // {goodbye, hello}
Iterator<String> iter = words.iterator();
while (iter.hasNext()){
    String word = iter.next();
    System.out.println(word);
}

• Important: the order in which words in a set are traversed is not necessarily the order in which they were added!

Maps

• An association of elements of one set (keys) to elements of another set (values)
• Each key maps to one value
• Multiple keys may map to the same value
• Maps take two type parameters, e.g.
  Map<String, Integer> wordCounts;
  associates strings to integers
• TreeMap and HashMap are two available implementations

Map Operations

• put: associates a given key with a given value
• get: returns the value associated with a given key
• remove: removes a key and its associated value
• keySet: returns a Set object containing all keys in the Map

Map Example

Map<String, Integer> wordCounts =
    new TreeMap<String, Integer>();
wordCounts.put("hello", 10);
wordCounts.put("goodbye", 20);

// prints 10:
System.out.println(wordCounts.get("hello"));

wordCounts.put("hello", 40); // replaces 10 with 40

// prints 40:
System.out.println(wordCounts.get("hello"));

Iterating over all key/value pairs

• To iterate over all key value pairs:
  – use keySet() method to get the set of keys
  – use iterator() method on keySet to get an iterator
  – iterate over the keys, and use get() method to get the associated values
Set<String> keys = wordCounts.keySet();
Iterator<String> iter = keys.iterator();
while (iter.hasNext()){
    String word = iter.next();
    System.out.println(word + "->" + wordCounts.get(word));
}

How to choose collection type?

• What collections would you choose for:
  – Counting the number of unique words in a file?
  – Counting the number of times each unique word occurs in a file?
  – Keeping track of the grades earned in a course?
  – Keeping track of locations during a random walk?