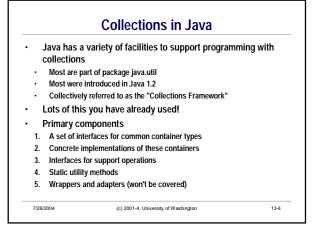




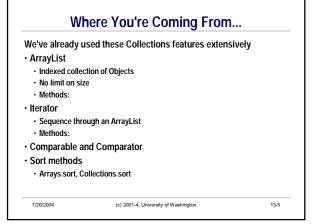
· Along the way, point out useful applications

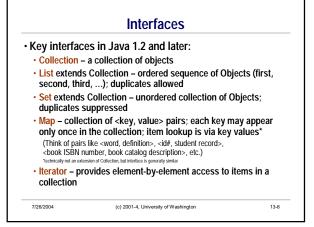
(c) 2001-4, University of Washington

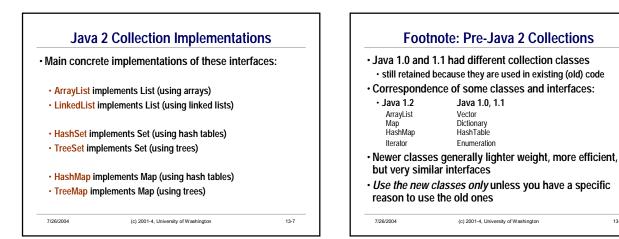
13-3



7/26/2004



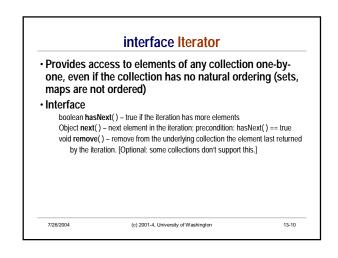


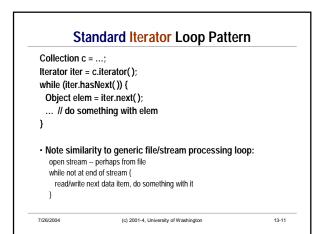


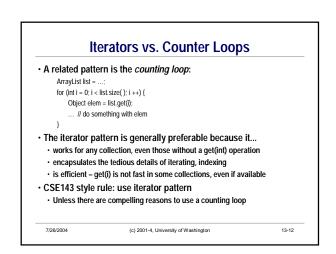
13-8

interface Collection

· Basic methods available on most collections: int size() - # of items currently in the collection boolean isEmpty() - (size() == 0) boolean contains(Object o) - true if o is in the collection [how to compare o with the elements already in the collection?] boolean add(Object o) - ensure that o is in the collection, possibly adding it; return true if collection altered; false if not. [leaves a lot unspecified....] boolean addAll(Collection other) - add all elements in the other collection boolean remove(Object o) - remove one o from the collection, if present; return true if something was actually removed void clear() - remove all elements Iterator iterator() - return an iterator object for this collection · Note: much richer interface than an array 7/26/2004 (c) 2001-4, University of Washington 13-9







Collection Contents: Objects

All Java Collections store Objects

- Cannot store primitive types directly
- Use wrapper classes if needed

7/26/2004

Values returned from Collections must be cast back to a type
 Integer age = new Integer(21);
 ArrayList ageList = new ArrayList();
 ageList.add(0, age);
 Integer ageAgain = ageList.get(0);
 // type error!

 Object ageAgain = ageList.get(0);
 // correct - but not always useful!

 Integer ageAgain = (Integer) ageList.get(0);
 // correct and useful

Contrast: Arrays are declared with a single, specific element type
 Could be any type: Object, primitive type, interface, abstract class, concrete class, another array, etc.

13-13

(c) 2001-4, University of Washington

Lists as Collections • In some collections, there is no natural order • Toys in a toybox, grocery items in a bag, grains of sand on the beach

- In other collections, the order of elements is natural and important
- Chapters of a book, floors in a building, people camping out to buy Star Wars tickets
- Lists are collections where the elements have an order
 Each element has a definite position (first, second, third, ...)

(c) 2001-4, University of Washington

13-14

positions are generally numbered from 0

7/26/2004

<section-header><section-header><section-header><list-item><list-item><list-item><text>

interface ListIterator extends Iterator the iterator() method for a List actually returns an instance of ListIterator (extends Iterator) an also use listIterator (extends Iterator) an also use listIterator (int pos) to get a ListIterator starting at ite given position in the list tistIterator returns objects in the list collection in the order they appear in the collection supports additional methods: hasPrevious() – ro replace the current element with something else add(Object o) – to insert an element after the current element

List Implementations

ArrayList – internal data structure is an array

- Fast iterating
- · Fast access to individual elements (get(int), set(int, Object))
- · Slow add/remove except at the end of the list
- LinkedList internal data structure is a linked list
 - Fast iterating

7/26/2004

· Slow access to individual elements (get(int), set(int, Object)) · Fast add/remove, even in the middle of the list

(c) 2001-4, University of Washington

13-17

We'll dissect both forms of implementation shortly

interface Set extends Collection

· As in math, a Set is an unordered collection, with no duplicate elements

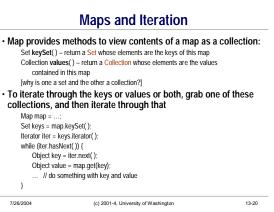
- · attempting to add an element already in the set does not change the set
- · Interface is same as Collection, but refines the specifications
- · The specs are in the form of comments
- interface SortedSet extends Set
- · Same as Set, but iterators always return set elements in order · Requires that elements be Comparable: implement the compareTo(Object) method, returning a negative, 0, or positive number to mean <=, ==, or >=, respectively

13-18

7/26/2004 (c) 2001-4, University of Washington

interface Map · Collections of <key, value> pairs · keys are unique, but values need not be · Doesn't extend Collection, but does provide similar methods size(), isEmpty(), clear() · Basic methods for dealing with <key, value> pairs: Object put(Object key, Object value) - add <key, value> to the map, replacing the previous <key, value> mapping if one exists void putAll(Map other) - put all <key, value> pairs from other into this map Object get(Object key) - return the value associated with the given key, or null if key is not present Object remove(Object key) - remove any mapping for the given key boolean containsKey(Object key) - true if key appears in a <key, value> pair boolean containsValue(Object value) - true if value appears in a <key, value> } (c) 2001-4, University of Washington 13-19

7/26/2004





 SortedMap can be used for maps where we want to store key/value pairs in order of their keys
 Requires keys to be Comparable, using compareTo

Sorting affects the order in which keys and values are iterated through

(c) 2001-4, University of Washington

keySet() returns a SortedSet

7/26/2004

13-21

Interoperating It is relatively easy to shift between various collections One collection to another: addAll(Collection) Arrays to Lists: static List Arrays.asList(array) Lists to Arrays: Object[] aList.toArray(List)

Preview of Coming Attractions

- 1. Study ways to implement these interfaces
- Array-based vs. link-list-based vs. hash-table-based vs. treebased
- 2. Compare implementations
- What does it mean to say one implementation is "faster" than another?
- Basic complexity theory O() notation
- 3. Use these and other data structures in our programming

(c) 2001-4,

7/26/2004

(c) 2001-4, University of Washington

13-23

CSE143 Sp04