CSSE 220 Day 20

Java Collections Framework
LinkedList Implementation
Work on Markov

CSSE 220 Day 20

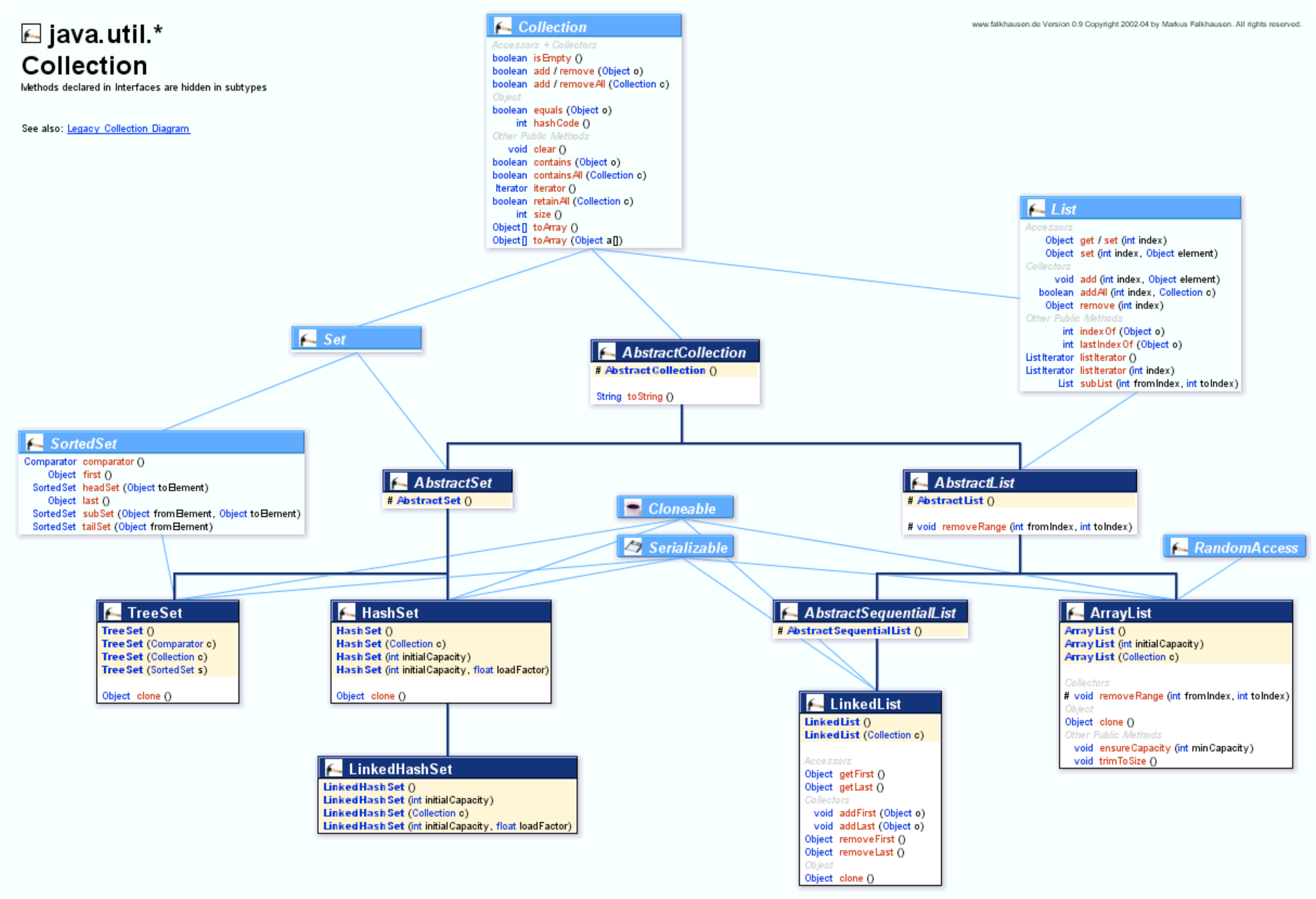
- Reminder: Exam #2 is (next) Friday, May 2.
- In order to reduce time pressure, you optionally may take the non-programming part 7:15-8:00 AM.
- Markov repositories:
  - http://svn.cs.rose-hulman.edu/repos/220-200820-markovXXX

- Questions?
- Today:
  - Java Collections
  - Iterators
  - Begin implementing lists

Data Structure Overview

Structure	find	insert/remove	Comments
Array	O(1)	can't do it	Constant-time access by position
Stack	top only O(1)	top only O(1)	Easy to implement as an array.
Queue	front only O(1)	O(1)	insert rear, remove front.
ArrayList	O(1)	O(N)	Constant-time access by position; O(log n) time to find arbitrary element if array is sorted
Linked List	O(n)	O(1)	O(N) to find insertion position, iterators (today) help.
HashSet/Map	O(1)	O(1)	If table not too full
TreeSet/Map	O(log N)	O(log N)	Kept in sorted order
MultiSet	O(log N)	O(log N)	keep track of multiplicities
PriorityQueue	min only O(1)	O(log N)	Can only find/remove smallest
Tree	O(log N)	O(log N)	If tree is balanced
Graph	O(N*M) ?	O(M)?	N nodes, M edges
Network			shortest path, maxFlow

Java Collections



java.util.*
Collection
Methods declared in Interfaces are hidden in subtypes

See also: Legacy Collection Diagram

www.falkhausen.de Version 0.9 Copyright 2002-04 by Markus Falkhausen. All rights reserved.

Collection

```
Accessors + Collectors
boolean  isEmpty ()
boolean  add / remove (Object o)
boolean  add / removeAll (Collection c)
Object
boolean  equals (Object o)
    int  hashCode ()
Other Public Methods
   void  clear ()
boolean  contains (Object o)
boolean  containsAll (Collection c)
Iterator iterator ()
boolean  retainAll (Collection c)
    int  size ()
Object[] toArray ()
Object[] toArray (Object a[])
```

List

```
Accessors
      Object  get / set (int index)
      Object  set (int index, Object element)
Collectors
        void  add (int index, Object element)
     boolean  addAll (int index, Collection c)
      Object  remove (int index)
Other Public Methods
         int  indexOf (Object o)
         int  lastIndexOf (Object o)
ListIterator  listIterator ()
ListIterator  listIterator (int index)
        List  subList (int fromIndex, int toIndex)
```

Set

AbstractCollection

```
# AbstractCollection ()

String  toString ()
```

SortedSet

```
Comparator  comparator ()
    Object  first ()
 SortedSet  headSet (Object toElement)
    Object  last ()
 SortedSet  subSet (Object fromElement, Object toElement)
 SortedSet  tailSet (Object fromElement)
```

AbstractSet

```
# AbstractSet ()
```

Cloneable

Serializable

AbstractList

```
# AbstractList ()

# void  removeRange (int fromIndex, int toIndex)
```

RandomAccess

TreeSet

```
TreeSet ()
TreeSet (Comparator c)
TreeSet (Collection c)
TreeSet (SortedSet s)

Object  clone ()
```

HashSet

```
HashSet ()
HashSet (Collection c)
HashSet (int initialCapacity)
HashSet (int initialCapacity, float loadFactor)

Object  clone ()
```

AbstractSequentialList

```
# AbstractSequentialList ()
```

ArrayList

```
ArrayList ()
ArrayList (int initialCapacity)
ArrayList (Collection c)

Collectors
# void  removeRange (int fromIndex, int toIndex)
Object
Object  clone ()
Other Public Methods
  void  ensureCapacity (int minCapacity)
  void  trimToSize ()
```

LinkedHashSet

```
LinkedHashSet ()
LinkedHashSet (int initialCapacity)
LinkedHashSet (Collection c)
LinkedHashSet (int initialCapacity, float loadFactor)
```

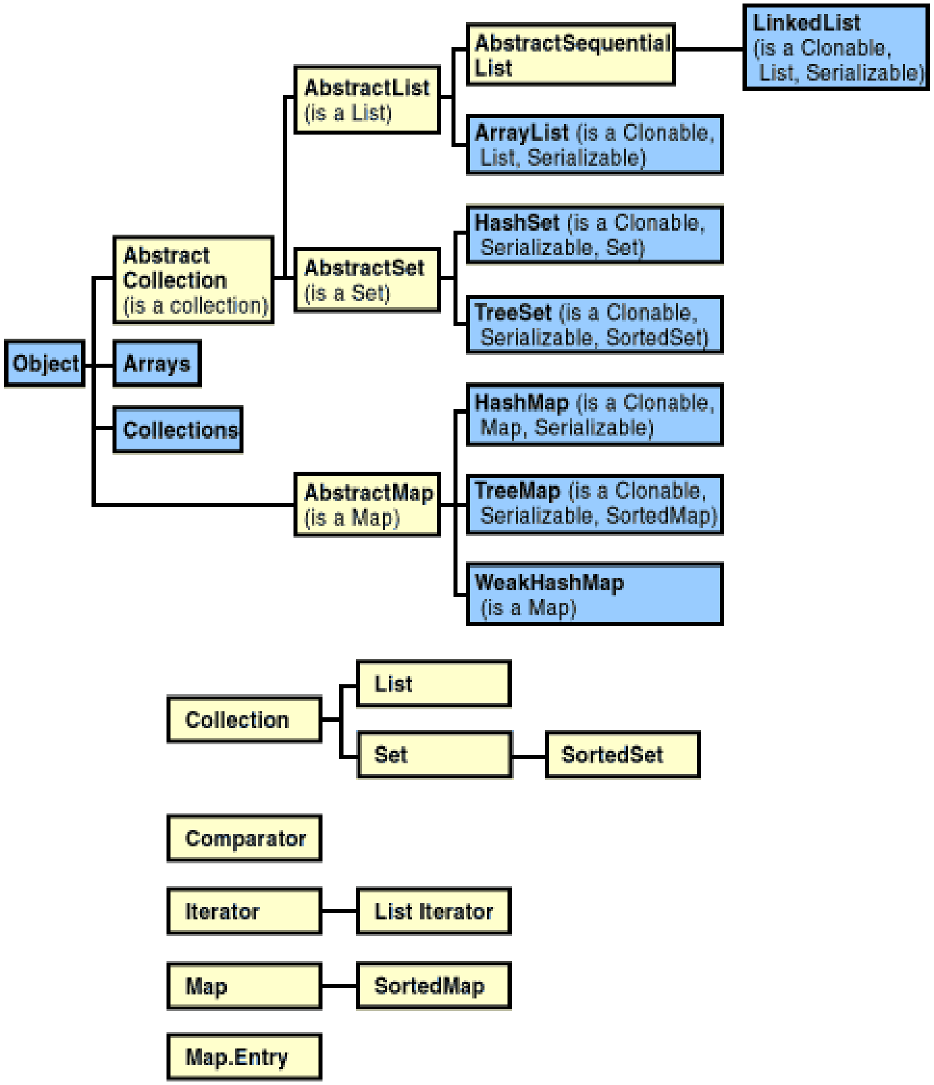
LinkedList

```
LinkedList ()
LinkedList (Collection c)

Accessors
Object  getFirst ()
Object  getLast ()
Collectors
  void  addFirst (Object o)
  void  addLast (Object o)
Object  removeFirst ()
Object  removeLast ()
Object
Object  clone ()
```

http://www.falkhausen.com/en/diagram/html/java.util.Collection.html

Collections classes and interfaces (classes at top, interfaces at bottom)


Object
Abstract Collection (is a collection)
Arrays
Collections
AbstractList (is a List)
AbstractSet (is a Set)
AbstractMap (is a Map)
AbstractSequential List
LinkedList (is a Clonable, List, Serializable)
ArrayList (is a Clonable, List, Serializable)
HashSet (is a Clonable, Serializable, Set)
TreeSet (is a Clonable, Serializable, SortedSet)
HashMap (is a Clonable, Map, Serializable)
TreeMap (is a Clonable, Serializable, SortedMap)
WeakHashMap (is a Map)
Collection
List
Set
SortedSet
Comparator
Iterator
List Iterator
Map
SortedMap
Map.Entry

Handy Refs: Java Collections Framework documentation

- Introductory page:
  - http://java.sun.com/j2se/1.5.0/docs/guide/collections/index.html
- Outline of the classes:
  - http://java.sun.com/j2se/1.5.0/docs/guide/collections/reference.html
- What's new in JDK 1.5 and 1.6:
  - http://java.sun.com/j2se/1.5.0/docs/guide/collections/changes5.html
  - http://java.sun.com/developer/technicalArticles/J2SE/Desktop/javase6/beta2.html

Specifying an ADT in Java

- The main Java tool for specifying an ADT is …

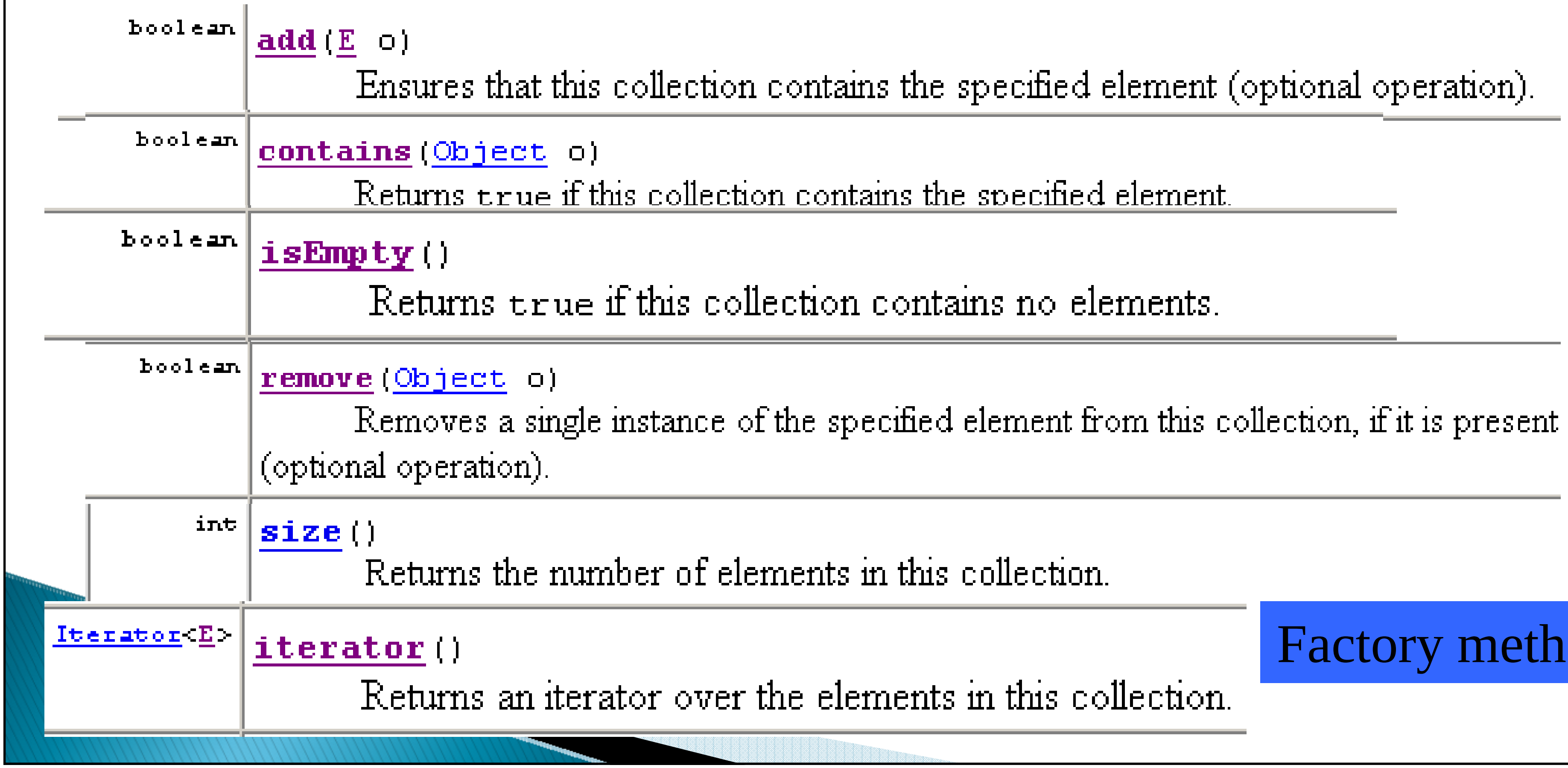
  … an interface

  Major example: The java.util.Collection interface.

- Some important methods from this interface:

java.util

Interface Collection<E>



boolean	**add**(E o) Ensures that this collection contains the specified element (optional operation).
boolean	**contains**(Object o) Returns `true` if this collection contains the specified element.
boolean	**isEmpty**() Returns `true` if this collection contains no elements.
boolean	**remove**(Object o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
int	**size**() Returns the number of elements in this collection.
Iterator<E>	**iterator**() Returns an iterator over the elements in this collection.

Factory method

Iterators

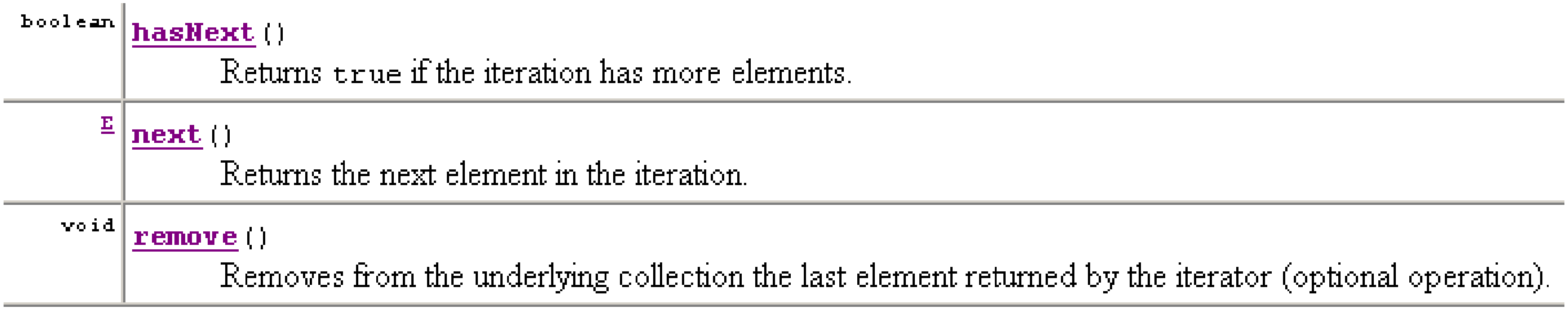
- Consider a loop to fund the sum of each element in an array:

```
for (int i = 0; i < ar.length; i++) {
  sum += ar[i];
}
```

We want to generalize this beyond arrays

What's an iterator?

- More specifically, what is a `java.util.Iterator`?
  - It's an interface:
  - **`interface java.util.Iterator<E>`**
  - with the following methods:

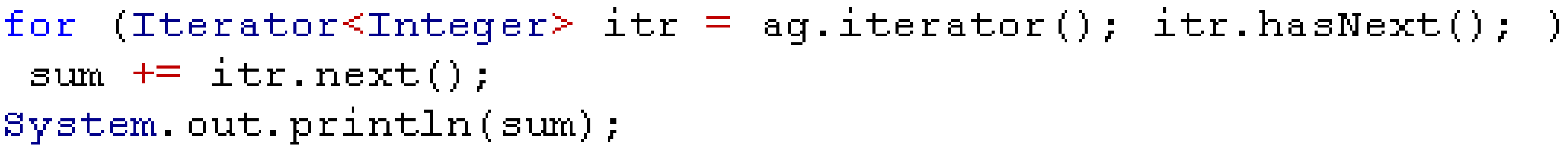


`boolean`	`hasNext()` Returns `true` if the iteration has more elements.
`E`	`next()` Returns the next element in the iteration.
`void`	`remove()` Removes from the underlying collection the last element returned by the iterator (optional operation).

- We create a new concrete instance of an iterator, but use an interface return type (using polymorphism). This is what a factory method does.
- The advantage is that if we change the type of collection used in main(), then we don’t have to change the iterator type.

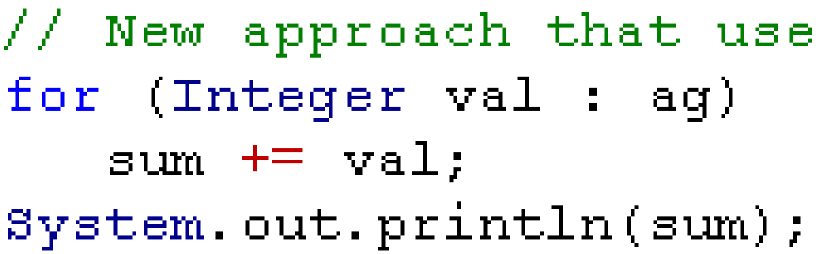
Example: Using an Iterator

ag is a Collection object.



```
for (Iterator<Integer> itr = ag.iterator(); itr.hasNext(); )
 sum += itr.next();
System.out.println(sum);
```

Using Java 1.5’s “foreach” construct:

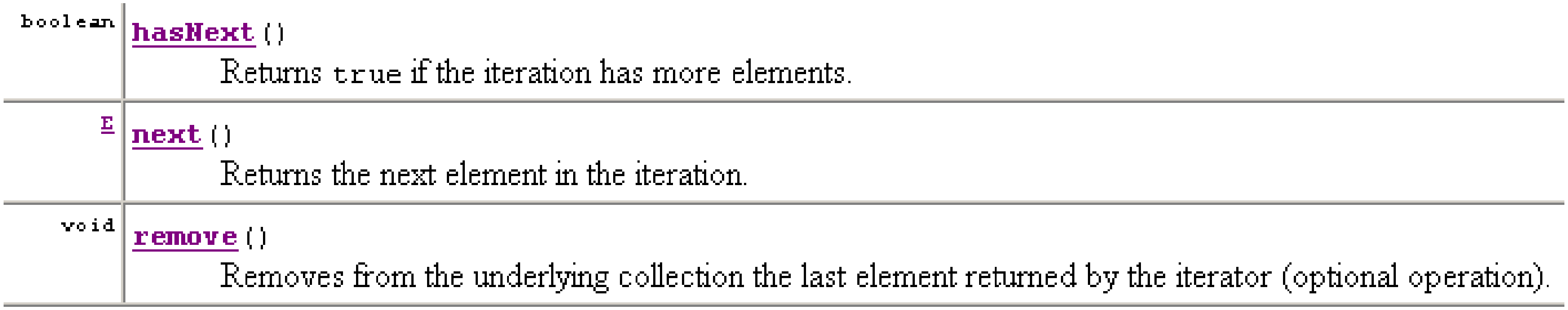


```
// New approach that uses an implicit iterator:
for (Integer val : ag)
    sum += val;
System.out.println(sum);
```

Note that the Java compiler essentially translates the latter code into the former.

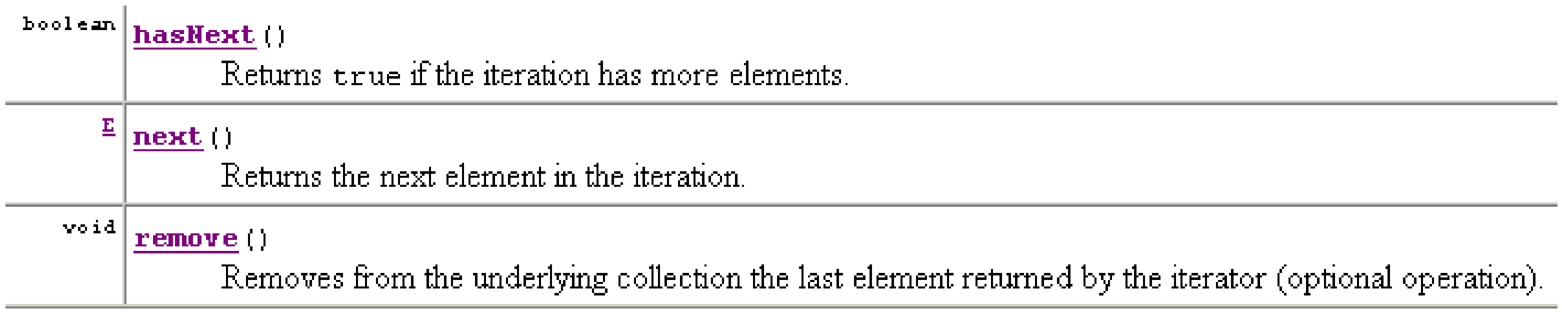
What's an iterator?

- More specifically, what is a `java.util.Iterator`?
  - It's an interface:
  - **`interface java.util.Iterator<E>`**
  - with the following methods:



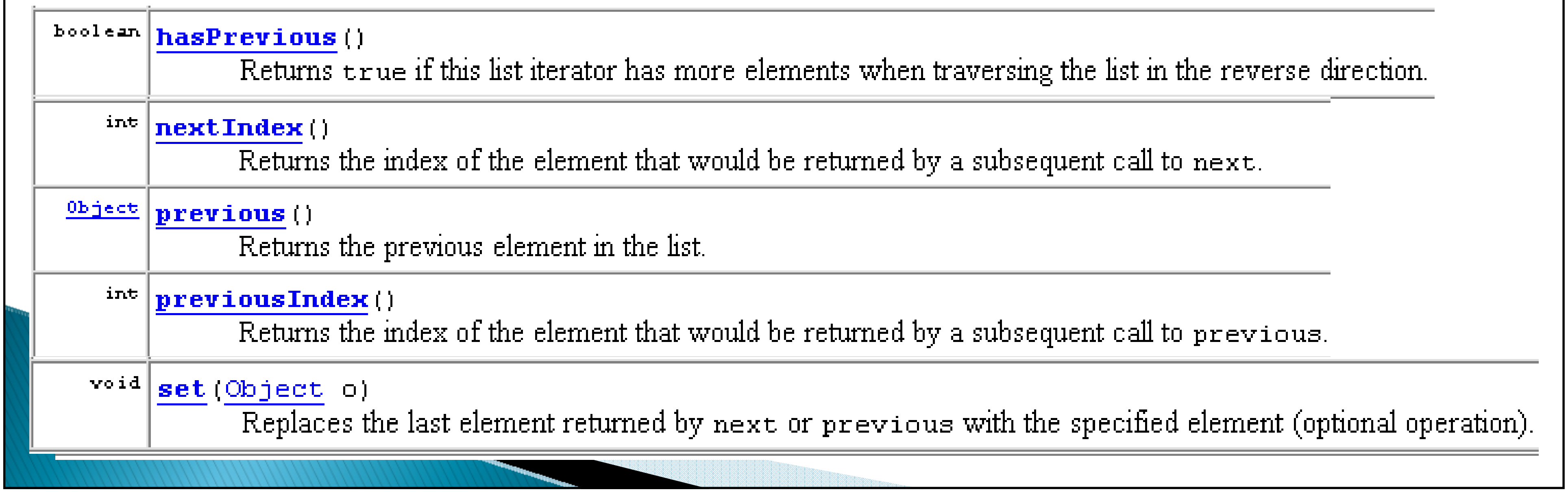
`boolean`	`hasNext()` Returns `true` if the iteration has more elements.
`E`	`next()` Returns the next element in the iteration.
`void`	`remove()` Removes from the underlying collection the last element returned by the iterator (optional operation).

- Why do iterators have their own **remove** method, separate from the Collections' **remove**?



boolean	**hasNext**() Returns `true` if the iteration has more elements.
E	**next**() Returns the next element in the iteration.
void	**remove**() Removes from the underlying collection the last element returned by the iterator (optional operation).

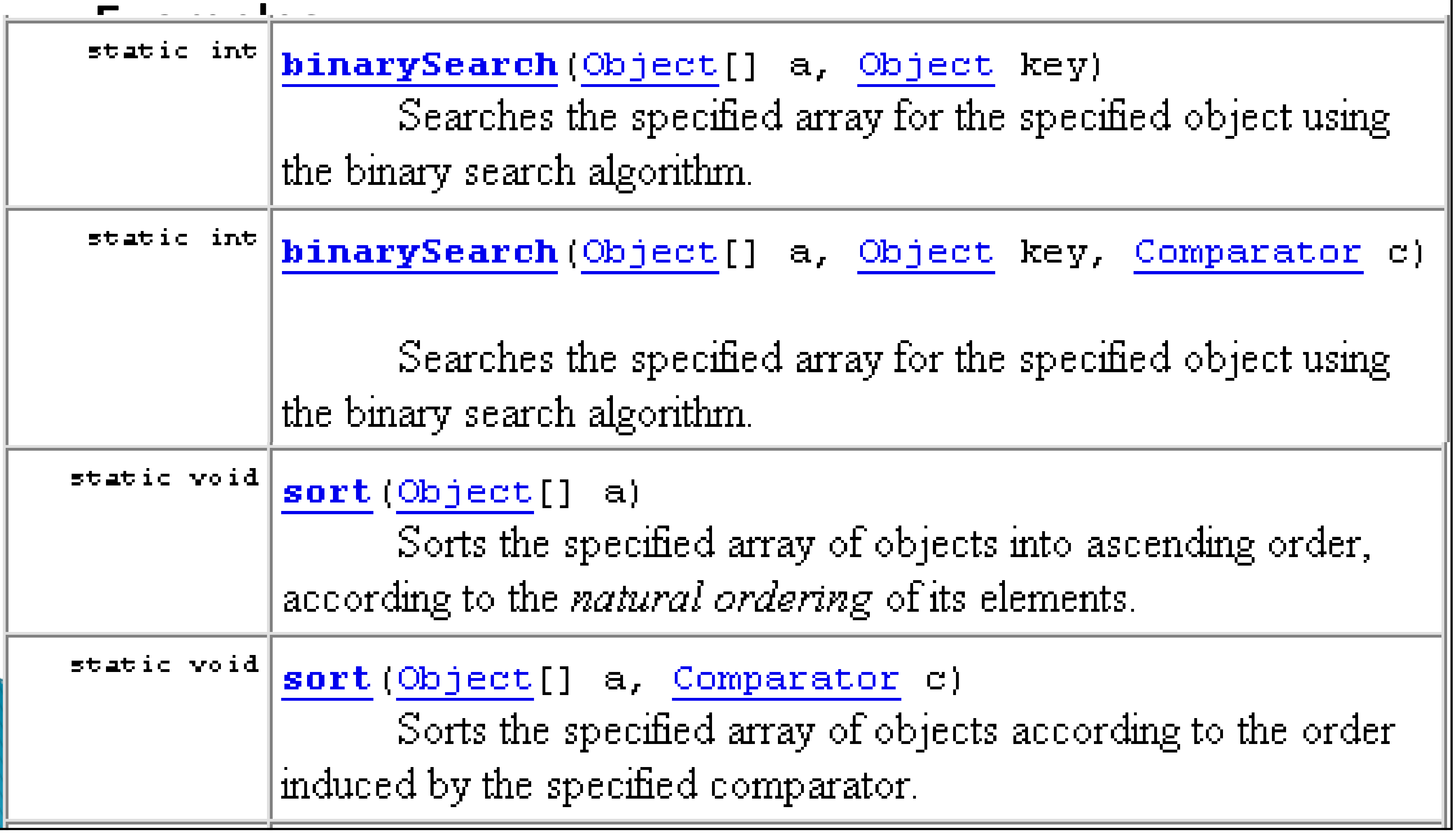
An extension, `ListIterator`, adds:



boolean	**hasPrevious**() Returns `true` if this list iterator has more elements when traversing the list in the reverse direction.
int	**nextIndex**() Returns the index of the element that would be returned by a subsequent call to `next`.
Object	**previous**() Returns the previous element in the list.
int	**previousIndex**() Returns the index of the element that would be returned by a subsequent call to `previous`.
void	**set**(Object o) Replaces the last element returned by `next` or `previous` with the specified element (optional operation).

Sort and Binary Search

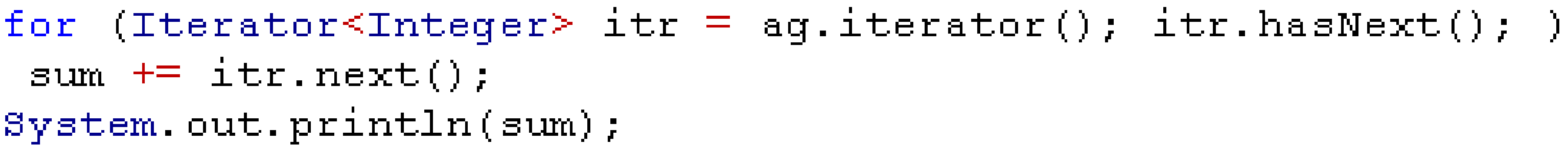
- The `java.util.Arrays` class provides static methods for sorting and doing binary search on arrays. Examples:



`static int`	`binarySearch(Object[] a, Object key)` Searches the specified array for the specified object using the binary search algorithm.
`static int`	`binarySearch(Object[] a, Object key, Comparator c)` Searches the specified array for the specified object using the binary search algorithm.
`static void`	`sort(Object[] a)` Sorts the specified array of objects into ascending order, according to the *natural ordering* of its elements.
`static void`	`sort(Object[] a, Comparator c)` Sorts the specified array of objects according to the order induced by the specified comparator.

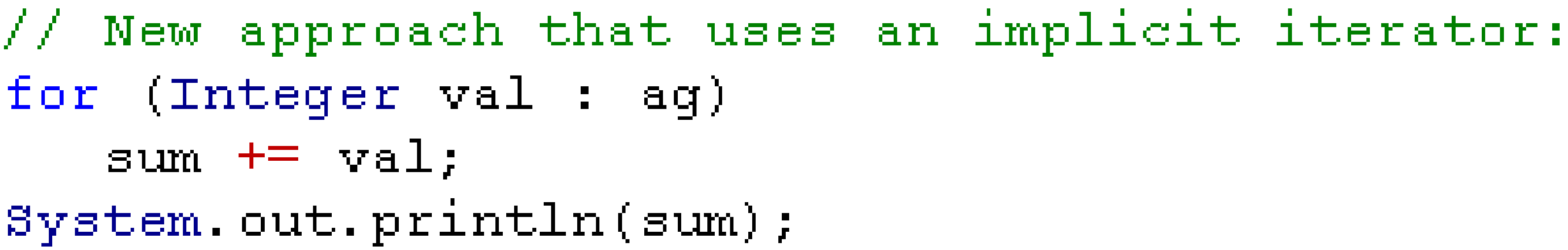
Example: Using an Iterator

ag can be any `Collection` of `Integer`s



```
for (Iterator<Integer> itr = ag.iterator(); itr.hasNext(); )
 sum += itr.next();
System.out.println(sum);
```

In Java 1.5 we can simplify it even more.



```
// New approach that uses an implicit iterator:
for (Integer val : ag)
    sum += val;
System.out.println(sum);
```

Note that the Java compiler translates the latter code into the former.

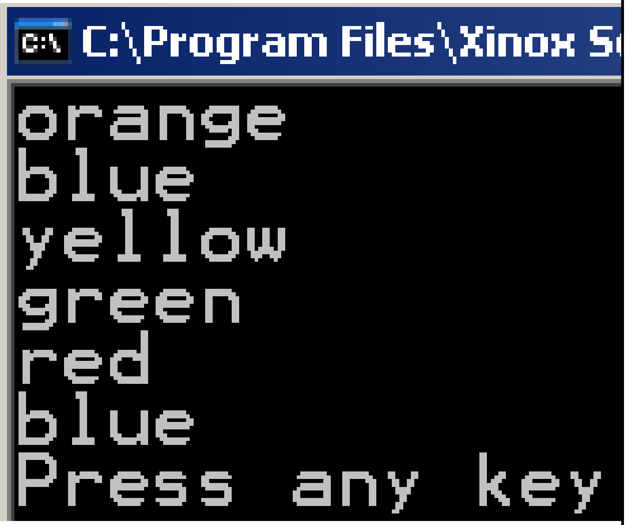
Tangent: Iterating over an enumerated type

```
class EnumTest {
    enum MyColors {orange, blue, yellow, green, red};

    public static void main (String[] args) {
        for (MyColors c : MyColors.values()) {
            System.out.println(c);
        }

        MyColors cc = MyColors.blue;

        switch (cc) {
            case orange:
              System.out.println("It is orange!")
              break;
            case green:
              System.out.println("Oh no! Not green!");
              break;
            case blue:
              System.out.println("blue");
              break;
            default:
              System.out.println("other");
        }
    }
}
```



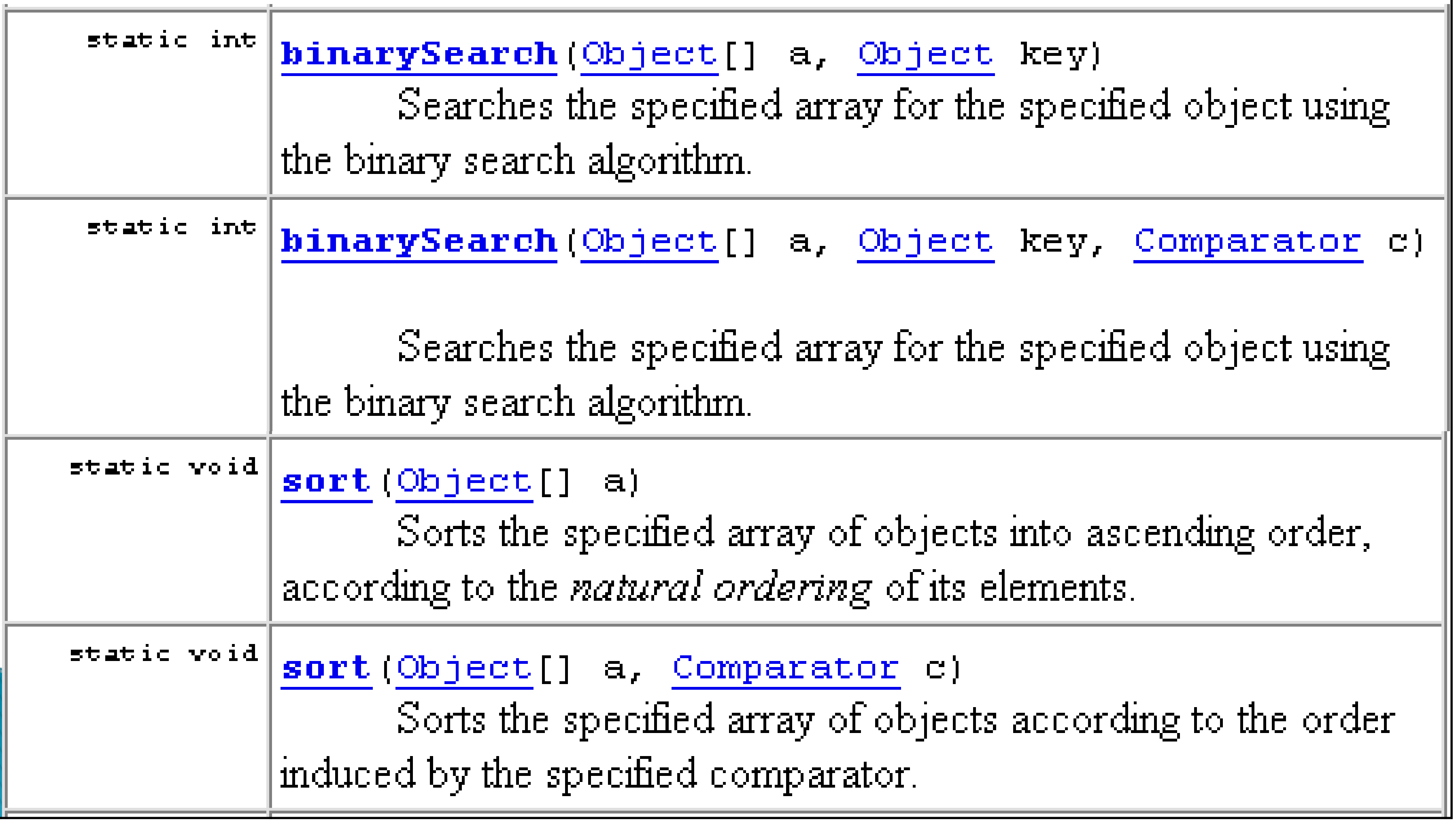
```
orange
blue
yellow
green
red
blue
Press any key
```

Additional Methods from the Collection Interface

- addAll – add all of the elements from another collection to this one
- containsAll – does this collection contain all of the elements of the other collection?
- removeAll – removes all of this collections elements that are also contained in the other collection
- retainAll – removes all of this collections elements that are not contained in the other collection
- toArray – returns an array that contains the same elements as this collection.

Sort and Binary Search

- The `java.util.Arrays` class provides static methods for sorting and doing binary search on arrays. **Examples:**



static int	`binarySearch(Object[] a, Object key)` Searches the specified array for the specified object using the binary search algorithm.
static int	`binarySearch(Object[] a, Object key, Comparator c)` Searches the specified array for the specified object using the binary search algorithm.
static void	`sort(Object[] a)` Sorts the specified array of objects into ascending order, according to the *natural ordering* of its elements.
static void	`sort(Object[] a, Comparator c)` Sorts the specified array of objects according to the order induced by the specified comparator.

Sort and Binary Search

- The `java.util.Collections` class provides similar static methods for sorting and doing binary search on `Collections`. Specifically `Lists`.
- **Look up the details in the documentation.**

The weiss.util and weiss.nonstandard packages

- In **weiss.util**, the author shows "bare bones" possible implementations of some of the classes in **java.util**.
- He picks the methods that illustrate the essence of what is involved in the implementation, for educational purposes.
- Some other Data Structures classes are in **weiss.nonstandard**.

The weiss.util and weiss.nonstandard packages

- In **weiss.nonstandard**, the author shows implementations of some common data structures that are not part of the **java.util** package, and he also shows alternate approaches to implementing some classes (like **Stack** and **LinkedList**) that are in **java.util**.

The weiss.util and weiss.nonstandard packages

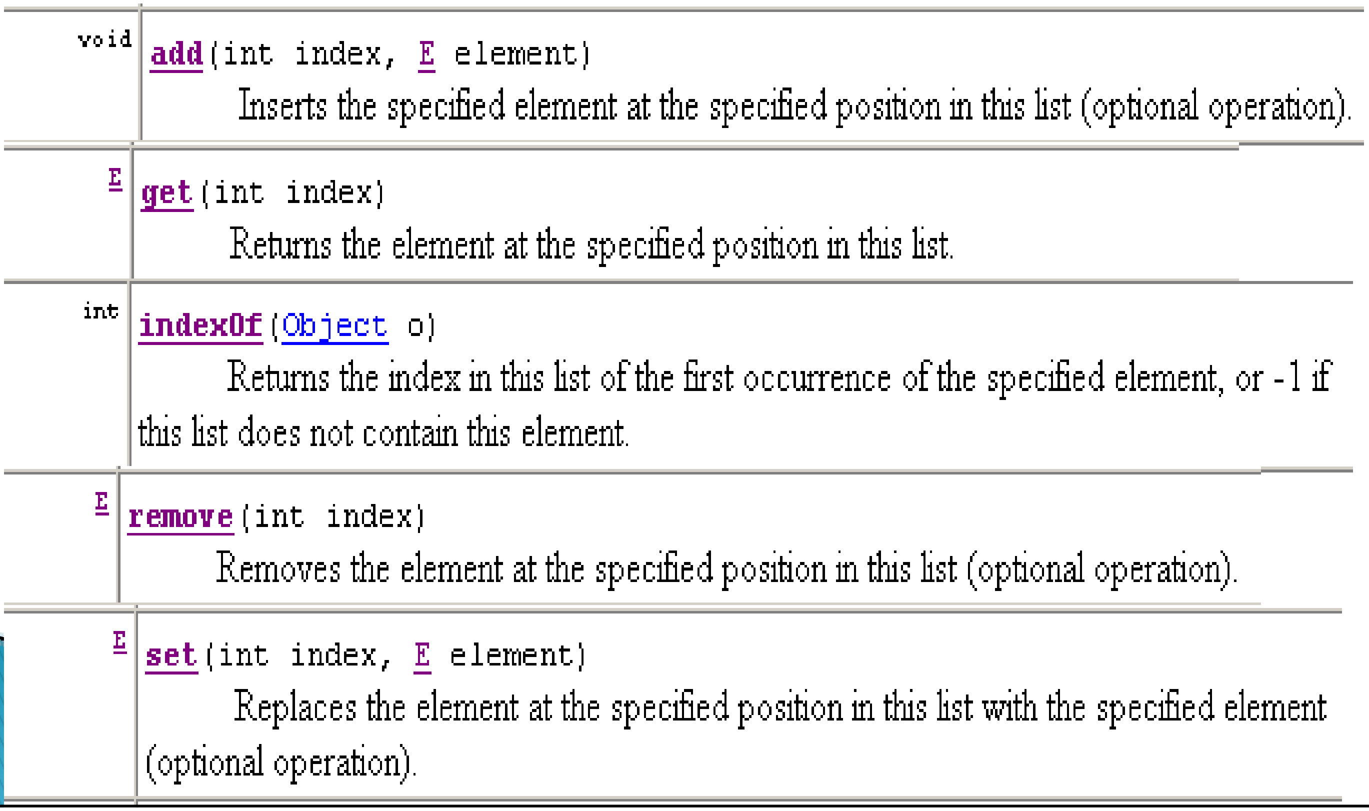
- If you followed the directions in assignment 1, both of these packages should be accessible to your code.
  - import weiss.nonstandard.*;
- Documentation is available, and you can copy it to your computer.

Now that we know about using some data structures …

- It's time to look at an implementation.

List Interface (extends Collection)

- A List is an ordered collection, items accessible by position. Here, *ordered* does not mean *sorted*.
- interface java.util.List<E>
- User may insert a new item at a specific position.
- Some important List methods:



void	add(int index, E element) Inserts the specified element at the specified position in this list (optional operation).
E	get(int index) Returns the element at the specified position in this list.
int	indexOf(Object o) Returns the index in this list of the first occurrence of the specified element, or -1 if this list does not contain this element.
E	remove(int index) Removes the element at the specified position in this list (optional operation).
E	set(int index, E element) Replaces the element at the specified position in this list with the specified element (optional operation).

`ArrayList` implementation of the `List` Interface

- Store items contiguously in a "growable" array.
- Looking up an item by index takes constant time.
- Insertion or removal of an object takes linear time in the worst case and on the average (why?).
- If `Comparable` list items are kept in sorted order in the ArrayList, finding an item takes log N time (how?).
- Let's sketch some of the implementation together.
  - Fields, constructor for empty list.