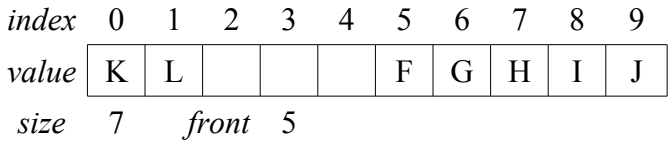
CSE 373 Section Handout #4
Syntax Reference

`Deque<E>` Methods *(a double-ended queue with O(1) add/remove at both ends)*

- implementations: `ArrayDeque`, `LinkedList`

`addFirst(`**value**`)`	inserts the given element at the front of the deque
`addLast(`**value**`)`	appends the given element at the back of the deque
`isEmpty()`	returns true if deque does not contain any elements
`iterator()`	an object to traverse the deque in front-to-back order
`descendingIterator()`	an object to traverse the deque in back-to-front order
`peekFirst()`	returns element at front of the deque (NoSuchElementException if empty)
`peekLast()`	returns element at back of the deque (NoSuchElementException if empty)
`removeFirst()`	removes/returns element at front (NoSuchElementException if empty)
`removeLast()`	removes/returns element at back (NoSuchElementException if empty)
`size()`	returns number of elements in the deque
`toString()`	string representation of deque in front-back order, e.g. `"[a, b, c]"`

Circular Array Buffer Deque Implementation *(An array with a movable front index that wraps around)*



`Iterator<E>` Methods *(an object for traversing the elements of any collection)*

- almost any Java collection has an `iterator()` method that returns an `Iterator`

`boolean hasNext()`	true if there are more elements to examine
`E next()`	returns the next element to be examined, and advances the iterator by one element; if there are no more elements, throws NoSuchElementException
`void remove()`	removes from the collection the last element returned by `next()`; if `next()` has not been called yet, throws IllegalStateException; sometimes not supported and throws UnsupportedOperationException

```
Set<Student> students = new TreeSet<Student>();
...
Iterator<Student> itr = students.iterator();
while (itr.hasNext()) {
    Student student = itr.next();
    if (student.gpa() < 2.0) {
        itr.remove();
    }
}
```

CSE 373 Section Handout #4

Deques

1. dequeMystery1

What are the contents of the `Deque` returned by each of the following calls? Write the deque's contents from the client perspective as they would appear by a call to the `toString` method.

```
public static Deque<String> mystery(String[] elements) {
    Deque<String> deque = new LinkedList<String>();
    for (int i = 0; i < elements.length; i++) {
        if (i % 3 == 0) {
            deque.addLast(elements[i]);
        } else {
            deque.addFirst(elements[i]);
        }
    }
    return deque;
}
```

a) `{"fee", "fie", "foe", "fum"}`

b) `{"Dana", "Jake", "Sara", "Mikey", "Janette", "Conor", "April"}`

c) `{"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m"}`

2. circularArrayBuffer

Suppose a deque is implemented using a circular array buffer as in our assignment. Assume that the buffer starts at length 10 and doubles in length when it cannot accommodate an element. Draw the state of the deque's internal array at each of the points indicated after the following calls are made. It may help you to also think about the state from the client's perspective (what would be returned by `toString`).

```
Deque<Integer> deque = new ArrayDeque<Integer>();
deque.addLast(10);
deque.addLast(20);
deque.addLast(30);
deque.addLast(40);
deque.addLast(50);    // a) _____

deque.removeFirst();
deque.removeFirst();
deque.removeFirst();  // b) _____

deque.addLast(60);
deque.addFirst(70);
deque.addFirst(80);   // c) _____

deque.addFirst(90);
deque.addFirst(100);
deque.addFirst(200);  // d) _____

for (int i = 1; i <= 6; i++) {
    deque.removeFirst();
    deque.addLast(i);
}
                      // e) _____

deque.addLast(300);
deque.addLast(400);
deque.addLast(500);
deque.addLast(600);
deque.addFirst(700);
deque.addFirst(800);  // f) _____
```

CSE 373 Section Handout #4

Deques, continued

3. `wrapHalf`

Write a method named `wrapHalf` that accepts a `Deque` of integers as a parameter and modifies its contents so that the elements in the last half of the deque are rearranged to be in the front of the deque in the same order. After a call to your method, the element that used to be the first in the second half of the deque will be the first overall element in the deque. If the deque is of odd size, consider the last half of the deck to include the middle element. Do not use any other collections or arrays as auxiliary storage.

For example, if passed the deque `[1, 2, 3, 4, 5, 6, 7, 8]`, your method should change it to `[5, 6, 7, 8, 1, 2, 3, 4]`. This method should run in O(*N*) time.

4. `stutterK`

Write a method named `stutterK` that accepts a `Deque` of strings and an integer *k* as a parameter and modifies its contents so that the first *k* elements in the deque each occur twice consecutively. The relative ordering of the elements should be retained. Do not use any other collections or arrays as auxiliary storage.

For example, if passed the deque `[a, b, c, d, e, f, g, h]`, and a *k* of 3, your method would change the deque to `[a, a, b, b, c, c, d, e, f, g, h]`. If you are passed a value of *k* that is negative or larger than the deque size, throw an `IllegalArgumentException`. This method should be O(*N*).

5. `evensBeforeOdds`

Write a method named `evensBeforeOdds` that accepts a `List` of integers as a parameter and modifies its contents so that all elements with even values occur before all elements with odd values. The relative ordering of the evens and the odds does not matter as long as all evens occur first. Use a single `Deque` as auxiliary storage to help you. For example, if passed the list `[2, 5, 3, 4, 7, 10, 9, 8]`, your method might change the list to `[8, 10, 4, 2, 5, 3, 7, 9]`. This method should run in O(*N*) time.

Iterators

6. `ArrayStackIterator`

The following code <u>incorrectly</u> implements an iterator over an array stack in top-to-bottom order. What is wrong with the code? What changes must be made in order for it to work properly?

```
public class ArrayStack<E> implements Stack<E> {
    private E[] elements;   // index 0 = bottom;  (size-1) = top
    private int size;
    ...
    private class ArrayStackIterator implements Iterator<E> {
        private int index;
        public ArrayStackIterator() {
            index = 0;
        }
        public boolean hasNext() {
            return index < size;
        }
        public E next() {
            E result = elements[index];
            index++;
            return result;
        }
        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
}
```

CSE 373 Section Handout #4

Iterators, continued

7. `RangeIterator`

Suppose we have a class called `IntegerRange` representing a range of integers, implemented as follows:

```
public class IntegerRange {
    private int min;
    private int max;   // both ends are inclusive
    ...
```

The author of this class wants to make it possible to use integer ranges as the target of a for-each loop:

```
IntegerRange range = new IntegerRange(7, 15);
for (int n : range) {
    System.out.print(n + " ");   // 7 8 9 10 11 12 13 14 15
}
```

Write a class named `RangeIterator` that would be usable as an inner class to implement an iterator over the integers in an `IntegerRange`. Implement the `hasNext` and `next` operations; when the `remove` method is called, you can throw an `UnsupportedOperationException`. Your iterator should not construct any internal data structures.

(What other changes must be made to the `IntegerRange` class for it to work in a for-each loop?)

8. `FibonacciIterator`

Write a class named `FibonacciIterator` that would be usable as a stand-alone class to implement an iterator over the Fibonacci integers. Recall that the first two Fibonacci numbers are 0 and 1, and each following Fibonacci number is the sum of the prior two. The first several Fibonacci numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

```
FibonacciIterator itr = new FibonacciIterator();
while (itr.hasNext()) {
    System.out.print(itr.next() + " ");   // 0 1 1 2 3 5 8 13 21 34 55 89 144 233 ...
}
```

Implement the `hasNext` and `next` operations; when the `remove` method is called, you can throw an `UnsupportedOperationException`. The Fibonacci numbers are an infinite sequence, so your `hasNext` method can always return `true`. Your iterator should not construct any internal data structures.

9. `CharacterIterator`

Write a class named `CharacterIterator` that would be usable as a stand-alone class to implement an iterator over the characters of a string. Your iterator's constructor should accept the string over which to iterate, and a `boolean` value named `reverse` to indicate whether to iterate over the characters in forward (`false`) or backward (`true`) order.

```
CharacterIterator itr = new CharacterIterator("HELLO", false);
while (itr.hasNext()) {
    System.out.print(itr.next() + "! ");   // H! E! L! L! O!
}
CharacterIterator itr2 = new CharacterIterator("GOODBYE", true);
while (itr2.hasNext()) {
    System.out.print(itr2.next() + "! ");   // E! Y! B! D! O! O! G!
}
```

Implement the `hasNext` and `next` operations; when the `remove` method is called, you can throw an `UnsupportedOperationException`. Your iterator should not construct any internal data structures.

CSE 373 Section Handout #4
Solutions

1.

a) `[foe, fie, fee, fum]`

b) `[Conor, Janette, Sara, Jake, Dana, Mikey, April]`

c) `[l, k, i, h, f, e, c, b, a, d, g, j, m]`

2.

a)
```
   0    1    2    3    4    5    6    7    8    9
[ 10,  20,  30,  40,  50,   _,   _,   _,   _,   _]    front = 0
```

b)
```
   0    1    2    3    4    5    6    7    8    9
[  _,   _,   _,  40,  50,   _,   _,   _,   _,   _]    front = 3
```

c)
```
   0    1    2    3    4    5    6    7    8    9
[  _,  80,  70,  40,  50,  60,   _,   _,   _,   _]    front = 1
```

d)
```
   0    1    2    3    4    5    6    7    8    9
[ 90,  80,  70,  40,  50,  60,   _,   _, 200, 100]    front = 8
```

e)
```
   0    1    2    3    4    5    6    7    8    9
[  5,   6,   _,   _,  50,  60,   1,   2,   3,   4]    front = 4
```

f)
```
  0   1  2  3  4  5  6  7    8    9   10   11  12  13  14  15  16  17   18   19
[50, 60, 1, 2, 3, 4, 5, 6, 300, 400, 500, 600,  _,  _,  _,  _,  _,  _, 800, 700]
                                                                front = 18
```

3.

```
public static void wrapHalf(Deque<Integer> deque) {
    for (int i = deque.size() / 2; i < deque.size(); i++) {
        int last = deque.removeLast();
        deque.addFirst(last);
    }
}
```

4.

```
public static void stutterK(Deque<String> deque, int k) {
    if (k < 0 || k > deque.size()) {
        throw new IllegalArgumentException();
    }
    for (int i = 0; i < k; i++) {
        String first = deque.removeFirst();   // stutter, temporarily place at back
        deque.addLast(first);
        deque.addLast(first);
    }
    for (int i = 0; i < 2 * k; i++) {
        deque.addFirst(deque.removeLast());   // move stuttered from back to front
    }
}
```

5.

```
public static void evensBeforeOdds(List<Integer> list) {
    Deque<Integer> deque = new LinkedList<Integer>();
    for (int n : list) {
        if (n % 2 == 0) {
            deque.addFirst(n);
        } else {
            deque.addLast(n);
        }
    }
    list.clear();
    list.addAll(deque);
}
```

CSE 373 Section Handout #4
Solutions, continued

6. The iterator goes in the opposite of the intended order (bottom to top). It also does not check `hasNext` in the code for `next`. Here is a correct version:

```
private class ArrayStackIterator implements Iterator<E> {
    private int index;

    public ArrayStackIterator() {
        index = size - 1;
    }

    public boolean hasNext() {
        return index >= 0;
    }

    public E next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        E result = elements[index];
        index--;
        return result;
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

7.

```
public class IntegerRange implements Iterable<Integer> {
    private int min;
    private int max;
    ...
    public Iterator<Integer> iterator() {
        return new RangeIterator();
    }
    ...

    private class RangeIterator implements Iterator<Integer> {
        private int current;

        public RangeIterator() {
            current = min;
        }

        public boolean hasNext() {
            return current <= max;
        }

        public Integer next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            int result = current;
            current++;
            return result;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
}
```

CSE 373 Section Handout #4
Solutions, continued

8.

```
public class FibonacciIterator implements Iterator<Integer> {
    private int prev1;
    private int prev2;
    private int count;

    public FibonacciIterator() {
        prev1 = 0;
        prev2 = 1;
        count = 0;
    }
    public boolean hasNext() {
        return true;
    }
    public Integer next() {
        count++;
        if (count == 1) {
            return prev1;
        } else if (count == 2) {
            return prev2;
        } else {
            int result = prev1 + prev2;
            prev1 = prev2;
            prev2 = result;
            return result;
        }
    }
    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

9.

```
public class CharacterIterator implements Iterator<Character> {
    private String str;
    private int index;
    private boolean reverse;

    public CharacterIterator(String str, boolean reverse) {
        this.str = str;
        this.reverse = reverse;
        this.index = reverse ? (str.length() - 1) : 0;
    }

    public boolean hasNext() {
        return index != (reverse ? -1 : str.length());
    }

    public Character next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        char chr = str.charAt(index);
        index += (reverse ? -1 : 1);
        return chr;
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

This document and its contents are copyright © University of Washington. All rights reserved.