Data Structures in Java

Lecture 7: Queues.

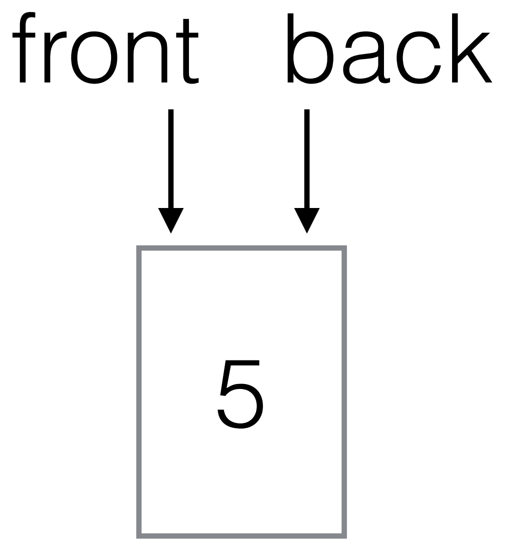
9/30/2015

Daniel Bauer

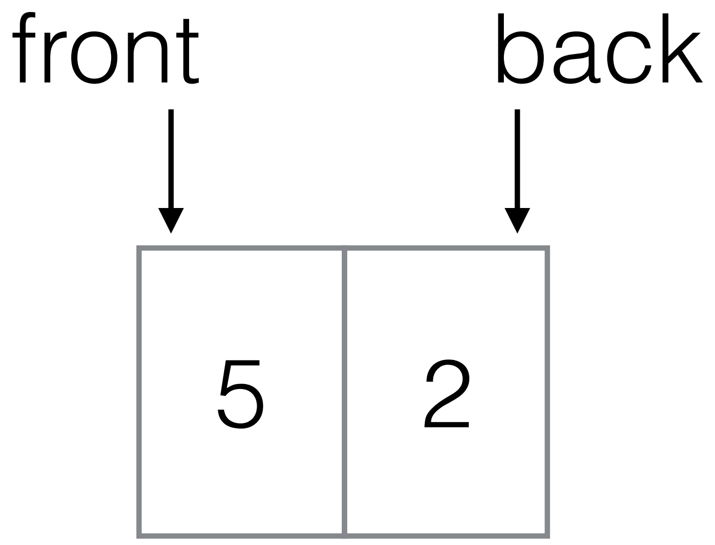
The Queue ADT

- A Queue *Q* is a sequence of *N* objects
  $A_0, A_1, A_2, \ldots, A_{N-1}$
- A_0 is called the front of Q, A_{N-1} is called the back of Q.
- A queue has two operations:
  - `void enqueue(x)` - append element *x* to the back of Q.
  - `Object dequeue()` - remove and return the front of Q.
- Queues are also known as **F**irst **I**n **F**irst **O**ut (FIFO) storage.

Queue Example

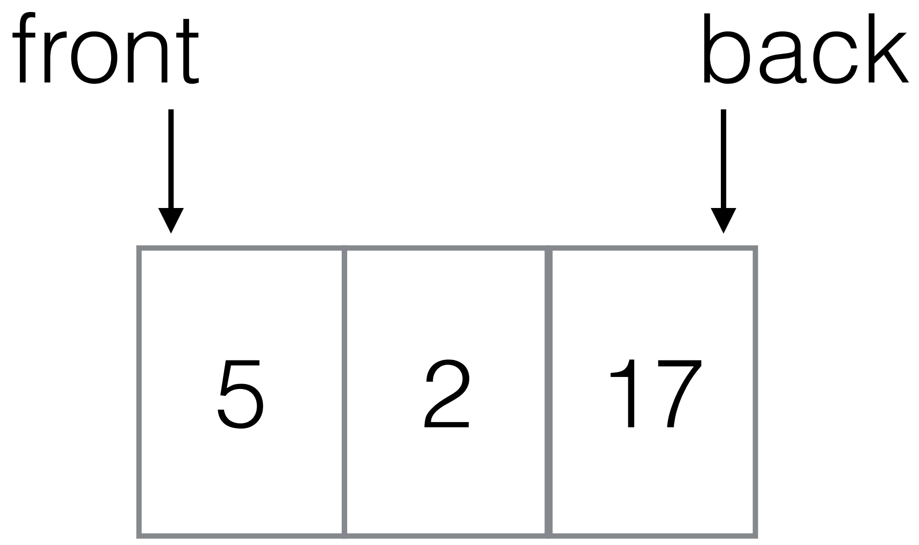


Queue Example



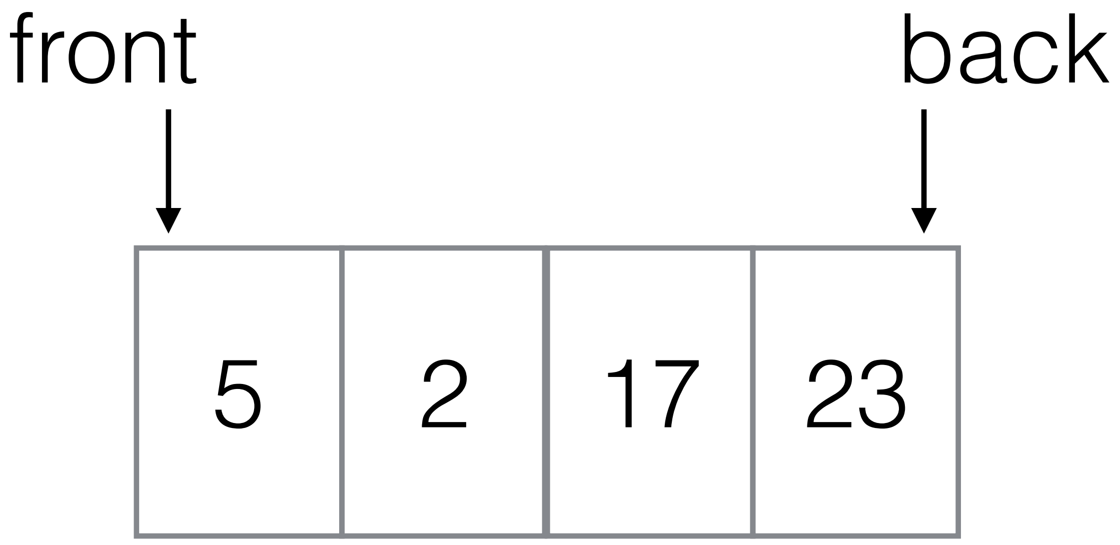
```
enqueue(2)
```

Queue Example



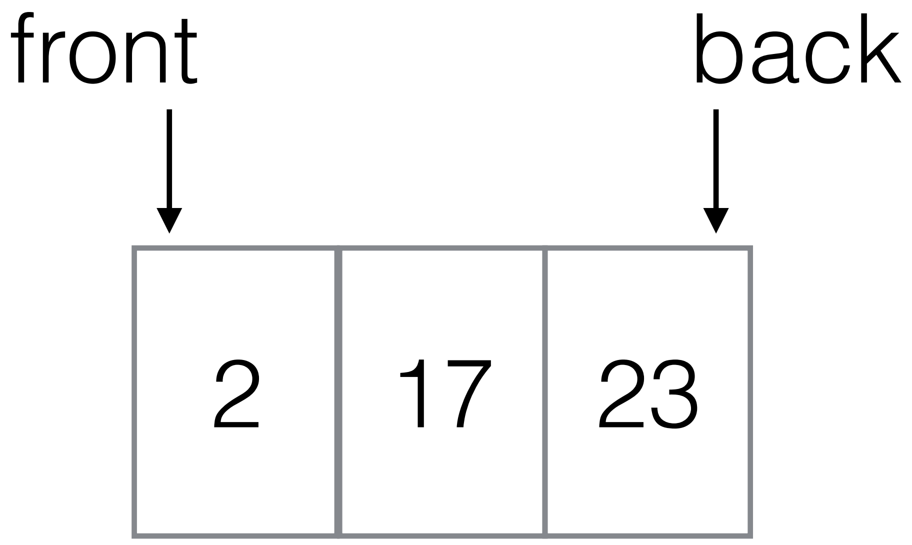
```
enqueue(2)  enqueue(17)
```

Queue Example



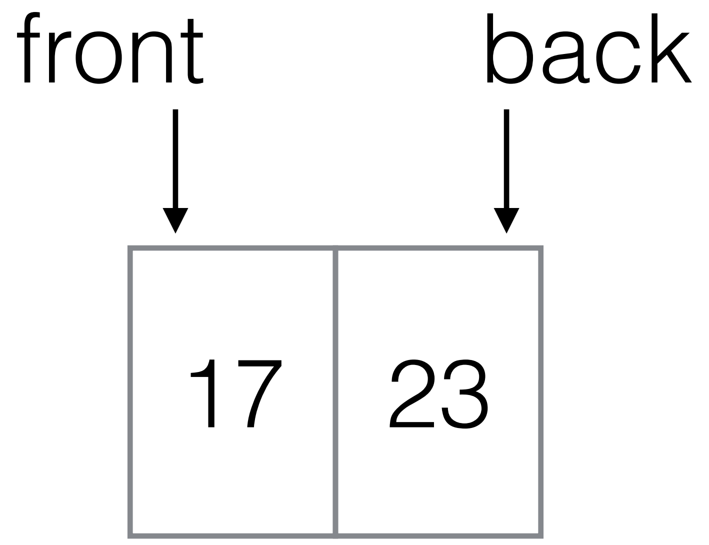
`enqueue(2)` `enqueue(17)` `enqueue(23)`

Queue Example



```
enqueue(2)  enqueue(17)      enqueue(23)

    dequeue() -> 5
```

Queue Example



```
enqueue(2)  enqueue(17)     enqueue(23)

    dequeue() -> 5    dequeue() -> 2
```

Implementing Queues

- Think of a Queue as a specialized List:
  - `enqueue`: Inserts only allowed at the end of the list.
  - `dequeue`: Remove only allowed at the beginning of the list.
- Can implement Queue using LinkedList implementation or using arrays.
  - enqueue and dequeue run in O(1) time with LinkedList.
  - What happens during dequeue in an Array?

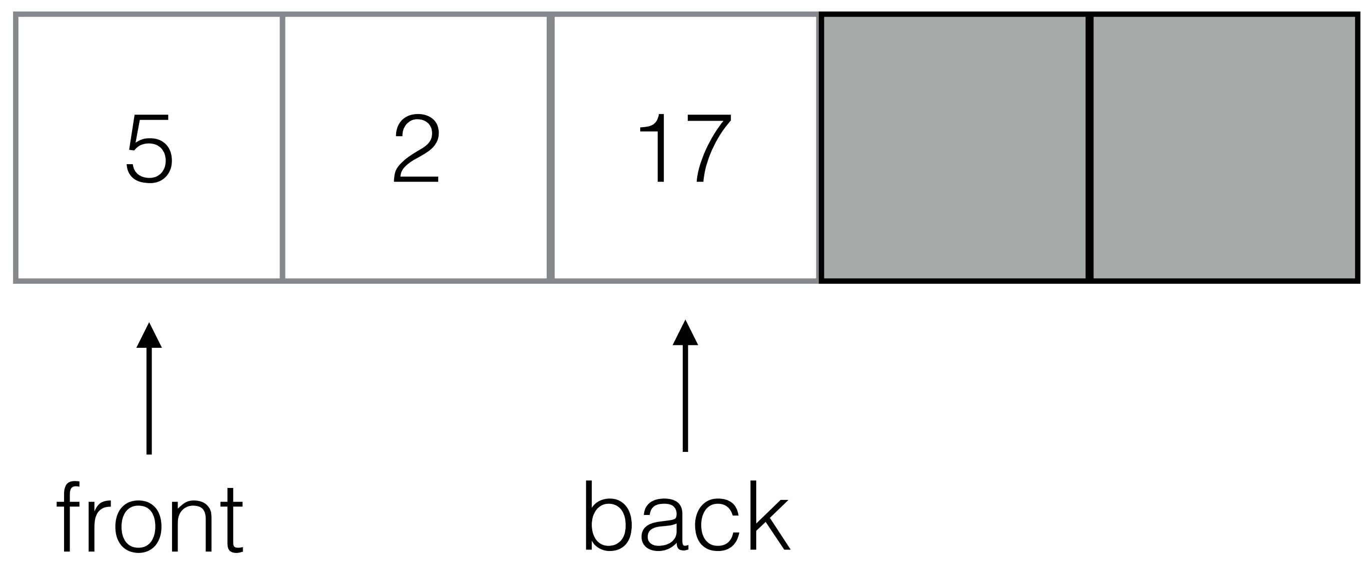
A Queue Interface

```
interface Queue<T> {
    /**
     * Insert a new item at the back of the queue
     */
    public void enqueue(T x);
    /**
     * Remove and return the next item from the
     * front of the queue.
     */
    public T dequeue();
    /**
     * Return the next item from the
     * front of the queue but do not remove it.
     */
    public T getFront();

}
```

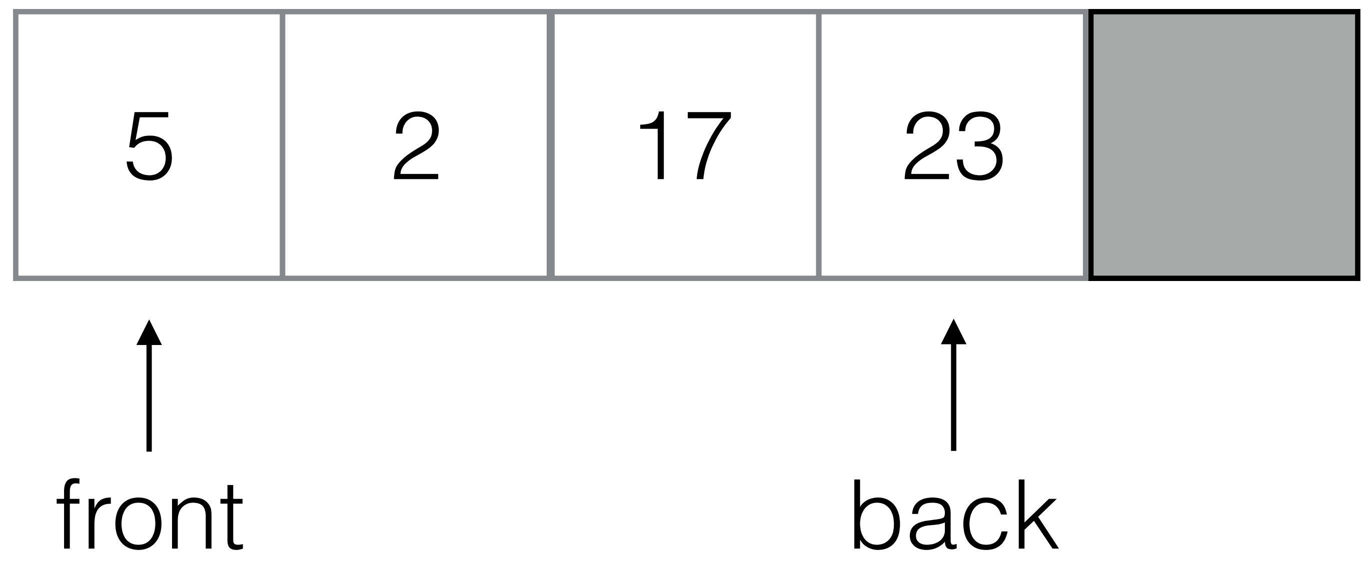
Using MyLinkedList to implement Queue

```
public class LinkedListQueue<T> extends MyLinkedList<T>
                                implements Queue<T> {

    public void enqueue(T x) {
        add(size(), x);
    }

    public T dequeue() {
        return remove(0);
    }
}
```

Dequeue on ArrayLists

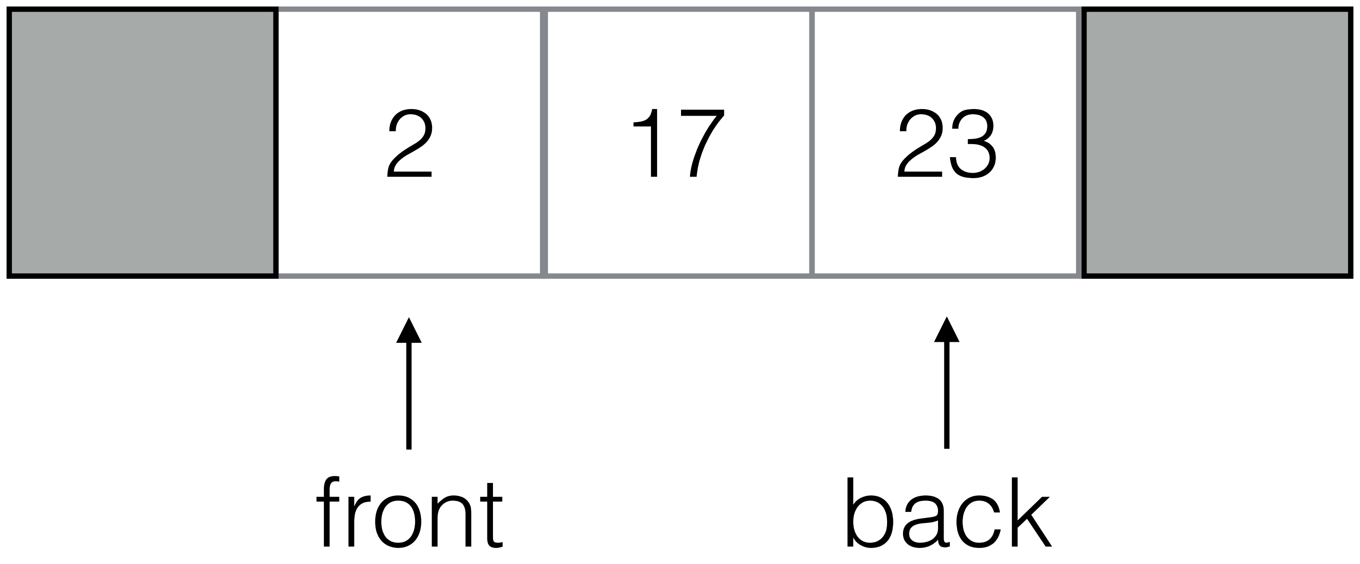


Dequeue on ArrayLists



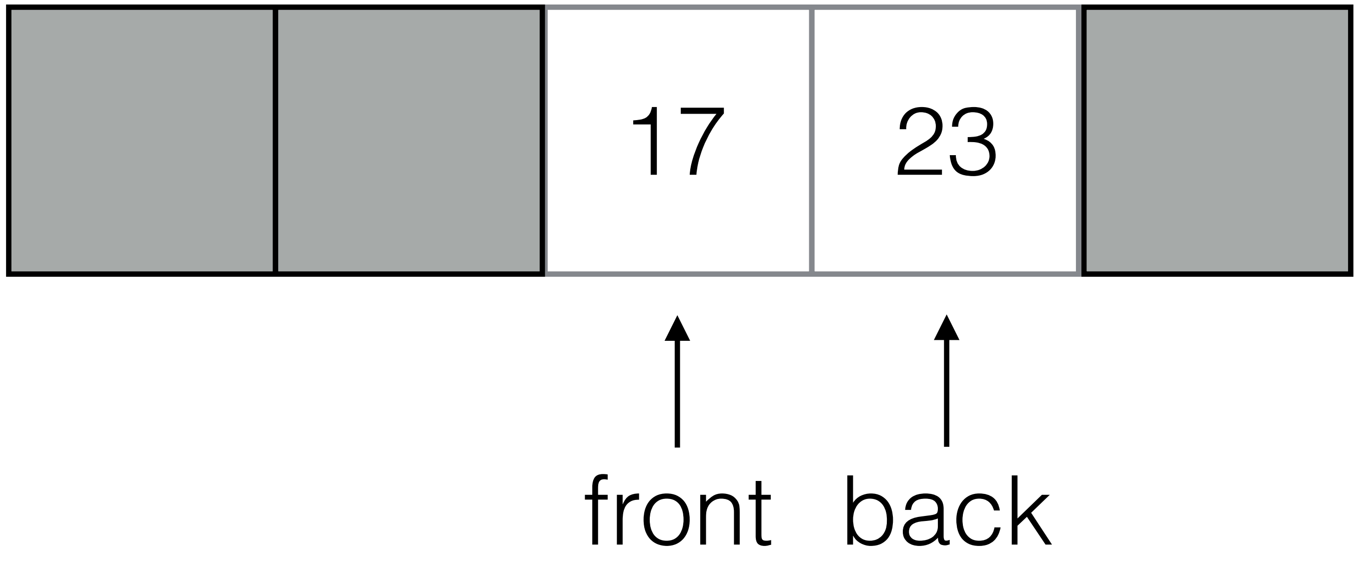
```
enqueue(23)
```

Dequeue on ArrayLists



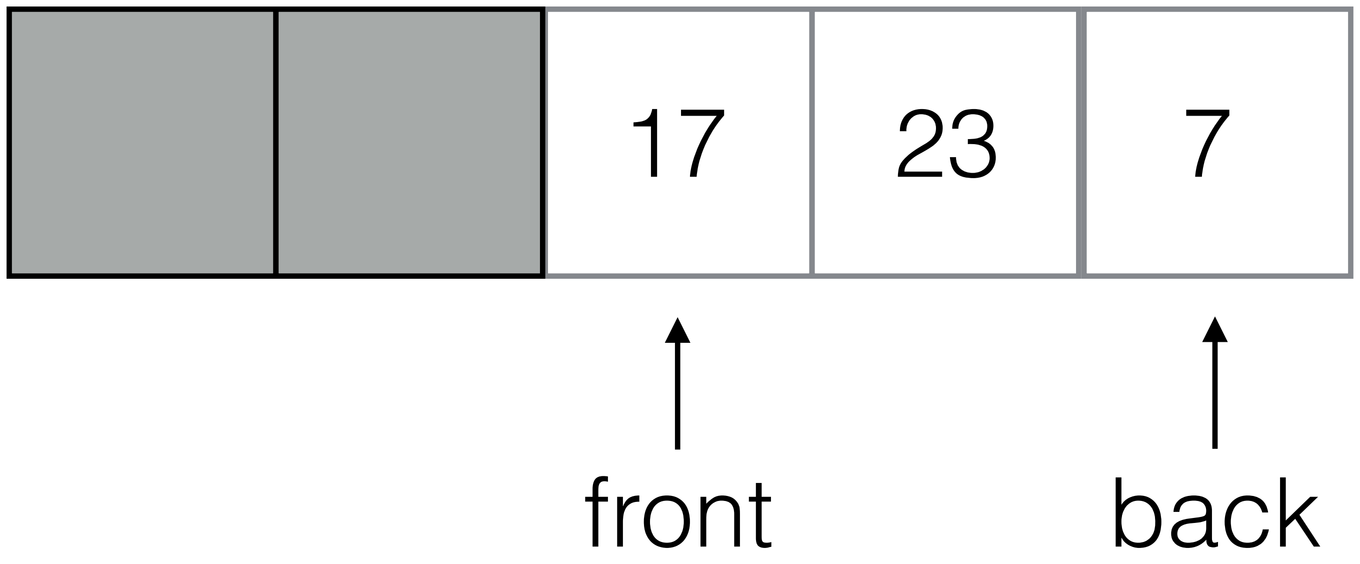
```
enqueue(23)   dequeue() -> 5
```

Dequeue on ArrayLists



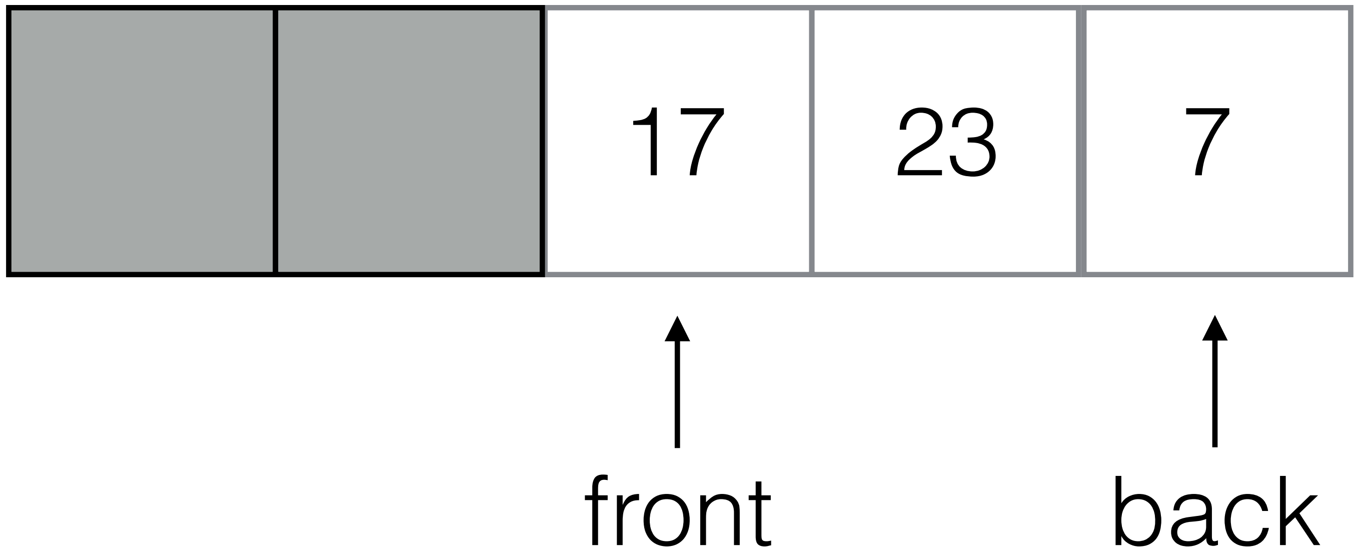
```
enqueue(23)   dequeue() -> 5   dequeue() -> 2
```

Dequeue on ArrayLists



```
enqueue(23)   dequeue() -> 5   dequeue() -> 2

          enqueue(7)
```

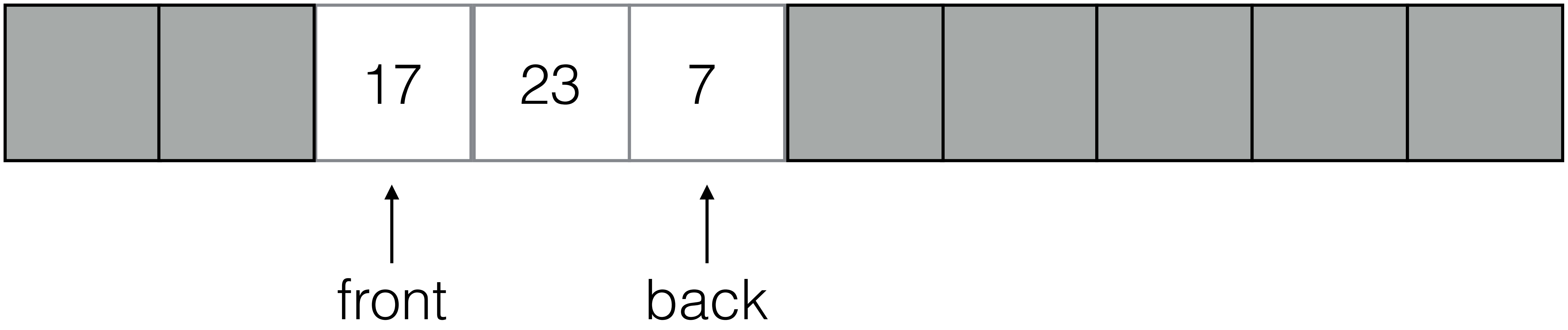
Dequeue on ArrayLists



```
enqueue(23)   dequeue() -> 5   dequeue() -> 2

        enqueue(7)    enqueue(42)
```

Dequeue on ArrayLists

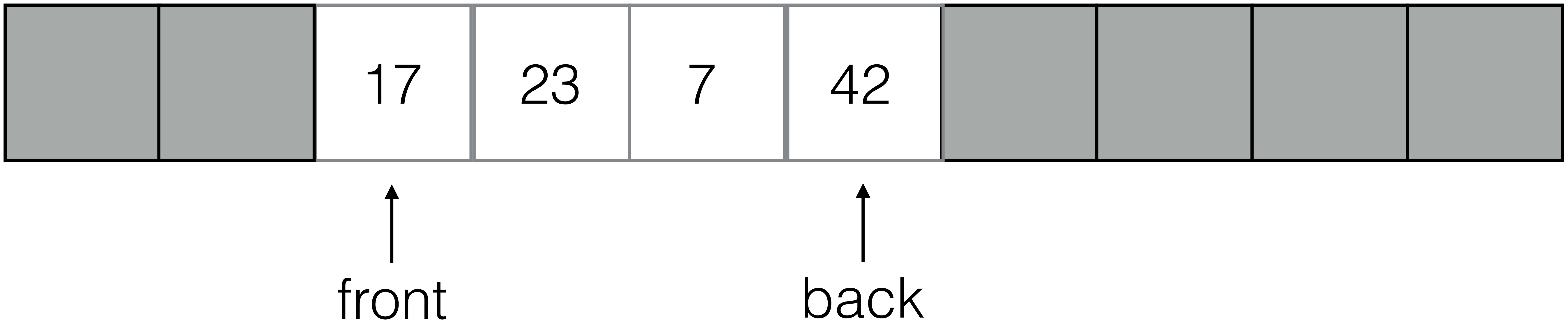
Need to reserve larger array, even though there is plenty of space at the beginning of the array.



```
enqueue(42)
```

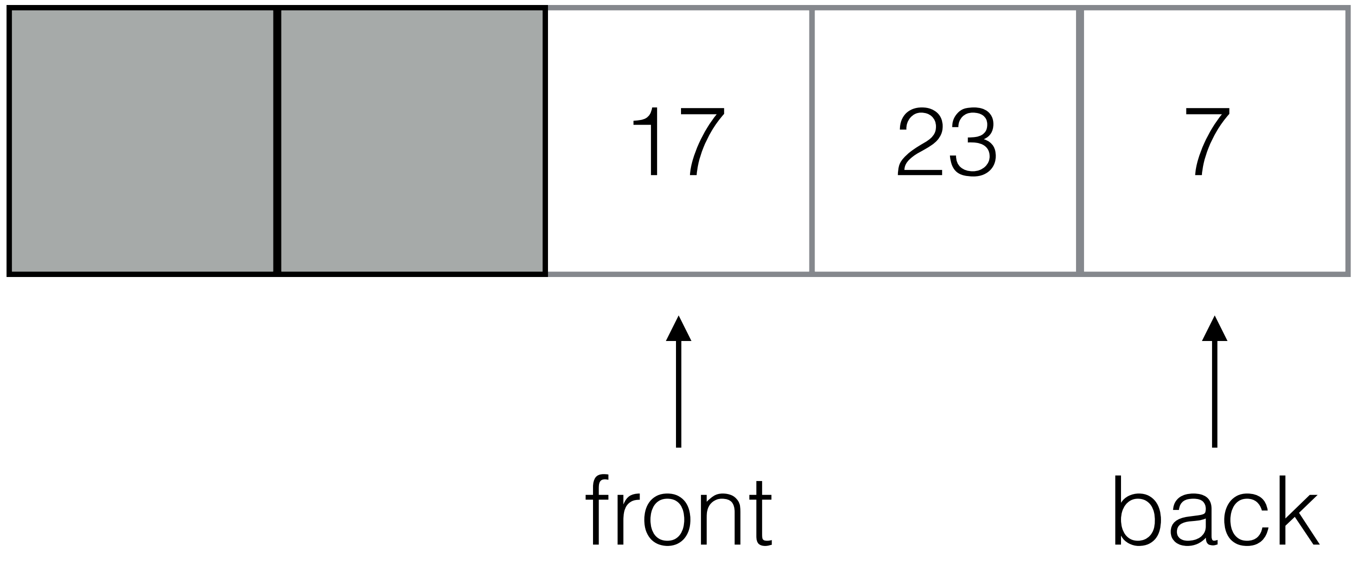
Dequeue on ArrayLists

Need to reserve larger array, even though there is plenty of space at the beginning of the array.



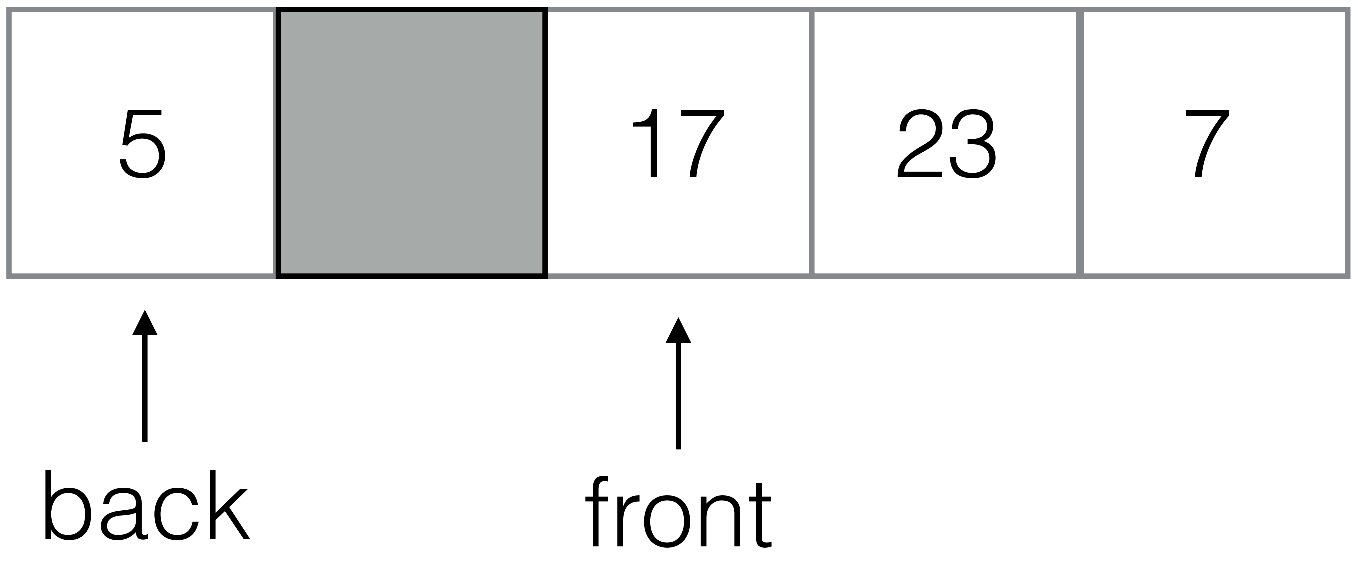
`enqueue(42)`

Circular Array



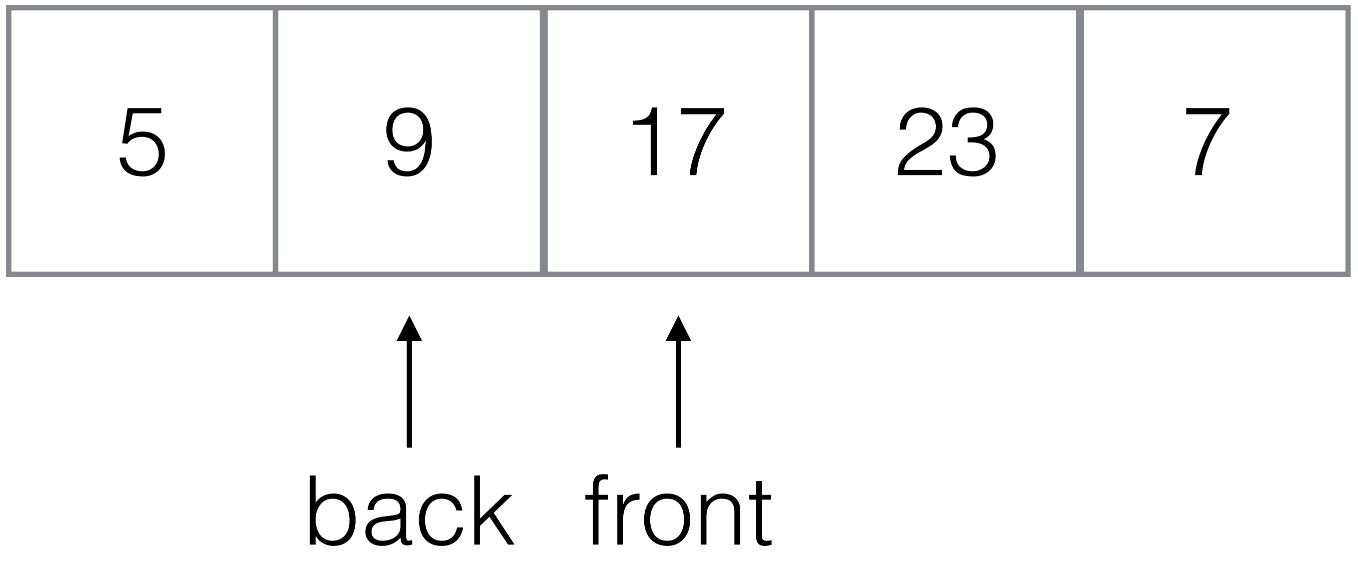
```
enqueue(42)          enqueue(9)
```

Circular Array



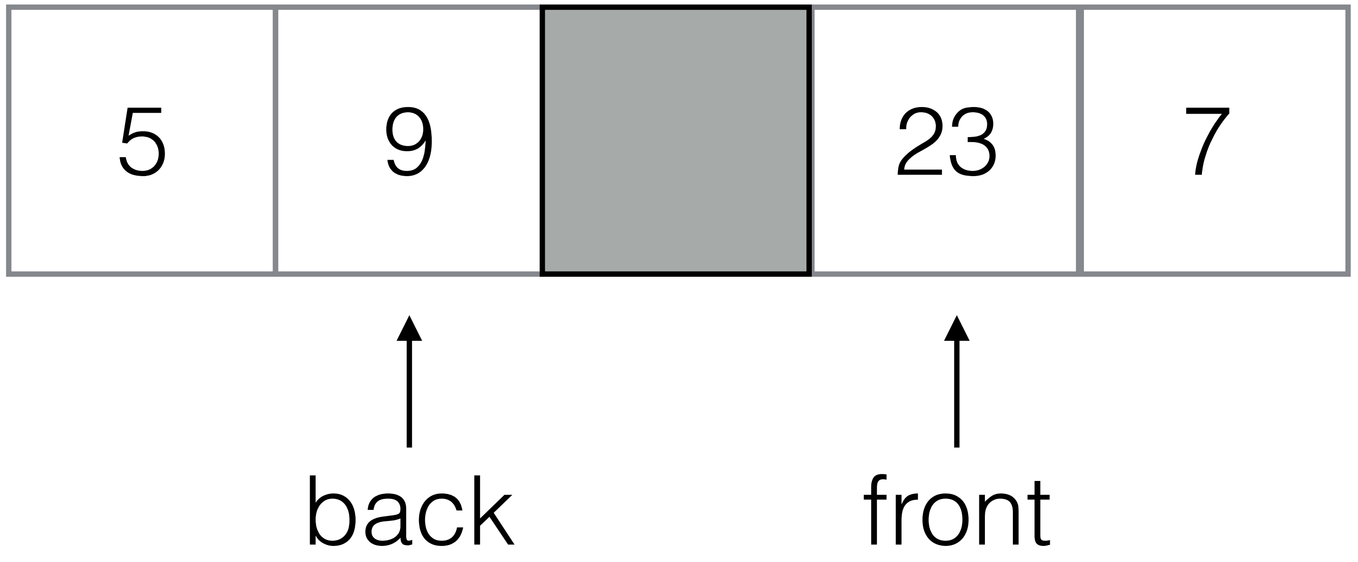
```
enqueue(42)          enqueue(9)
```

Circular Array



`enqueue(42)` `enqueue(9)`

Circular Array

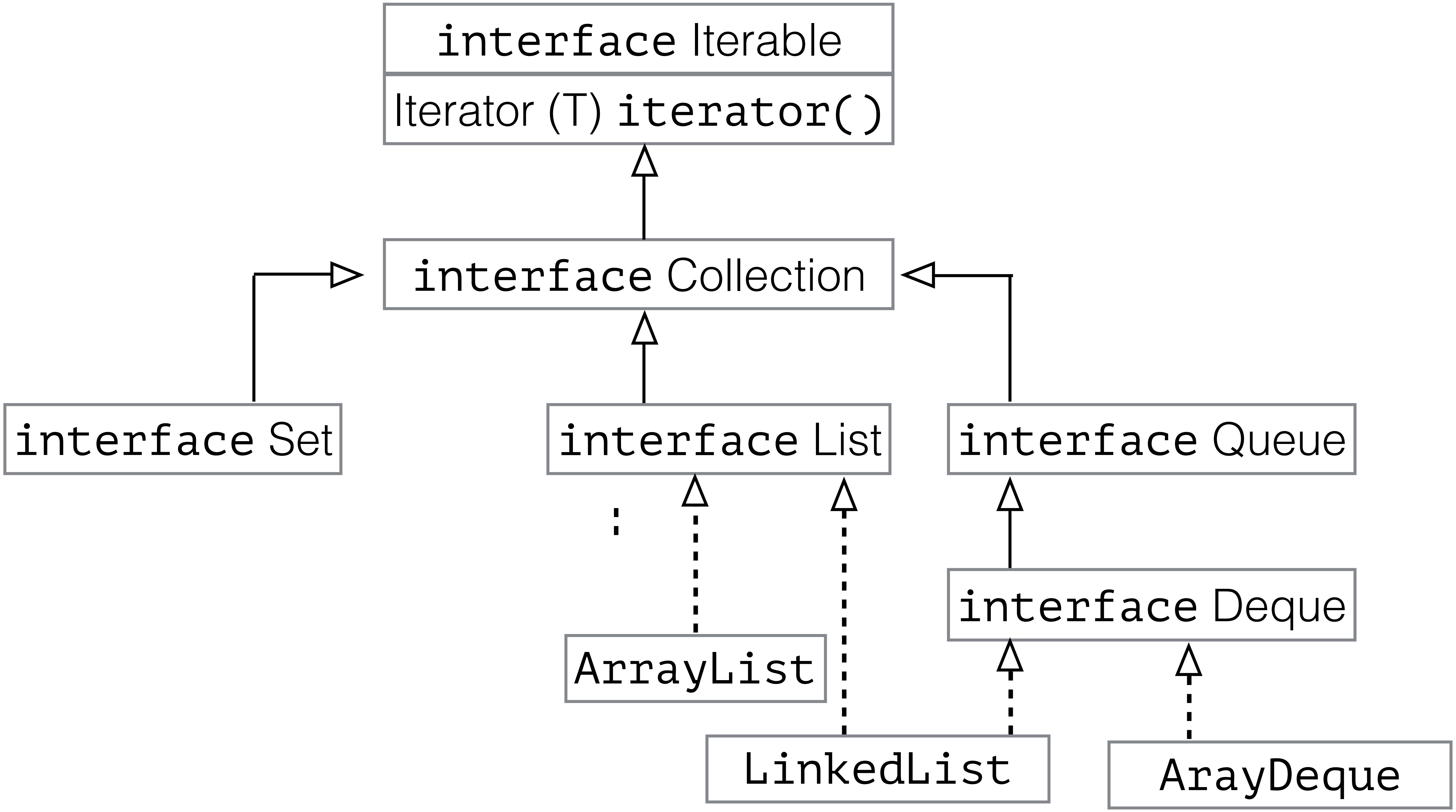


```
enqueue(42)          enqueue(9)

          dequeue() -> 17
```

Implementing Queue with a Circular Array

(example code)

Java Collections API



http://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

The Java Collection API

```
package java.util;

interface Collection<E> extends Iterable<E> {
    boolean add(E e);
    boolean addAll(Collection<? extends E> c);
    void clear();
    boolean contains(Object o);
    boolean containsAll(Collection<?> c);
    boolean isEmpty();
    Iterator<E> iterator(); // via Iterable
    boolean remove(Object o);
    boolean removeAll(Collection<?> c);
    boolean retainAll(Collection<?> c);
    int size();
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html

Java API List Interface

```
package java.util;

interface List<E> extends Collection<E> {
    E get(int index);
    int indexOf(Object o);
    int lastIndexOf(Object o);
    E remove(int index);
    E set(int index, E element);
    List<E> subList(int fromIndex, int toIndex)
}
```

http://docs.oracle.com/javase/7/docs/api/java/util/List.html

Java Queue Interface

```
package java.util;

interface Queue<E> extends Collection<E> {
    /* These methods throw exception on failure */
    boolean add(E e); // enqueue
    E remove(); // dequeue
    E element(); // Retrieve, but do not remove, front
    /* These methods return null on failure */
    boolean offer(E e); //enqueue
    E poll(); // dequeue
    E peek();
}
```

http://docs.oracle.com/javase/7/docs/api/java/util/Queue.html

Java Deque Interface

A linear collection that supports element insertion and removal at both ends. The name deque is short for "double ended queue" and is usually pronounced "deck"

```
package java.util;

interface Deque<E> extends Collection<E> {
    /* These methods throw exception on failure */
    boolean addFirst(E e);
    boolean addLast(E e);
    E removeFirst(); // dequeue
    E removeLast();  // dequeue
    E getFirst();
    E getLast();
   /* These methods return null on failure */
    …
}
```

http://docs.oracle.com/javase/7/docs/api/java/util/Dequeue.html

Deques can be Queues or Stacks

- **Stack view:**
  addFirst(E e) ~ push(E e)
  E removeFirst() ~ E pop()
  E getFirst() ~ E peek() / top()

- **Queue view:**
  addLast(E e) ~ enqueue(E e) / add(E e)
  E removeFirst() ~ dequeue() / remove()
  E getFirst() ~ element()