CS 91 USACO

Bronze Division

Unit 2: 1-D Data Structures

LECTURE 9: QUEUE

DR. ERIC CHOU

IEEE SENIOR MEMBER

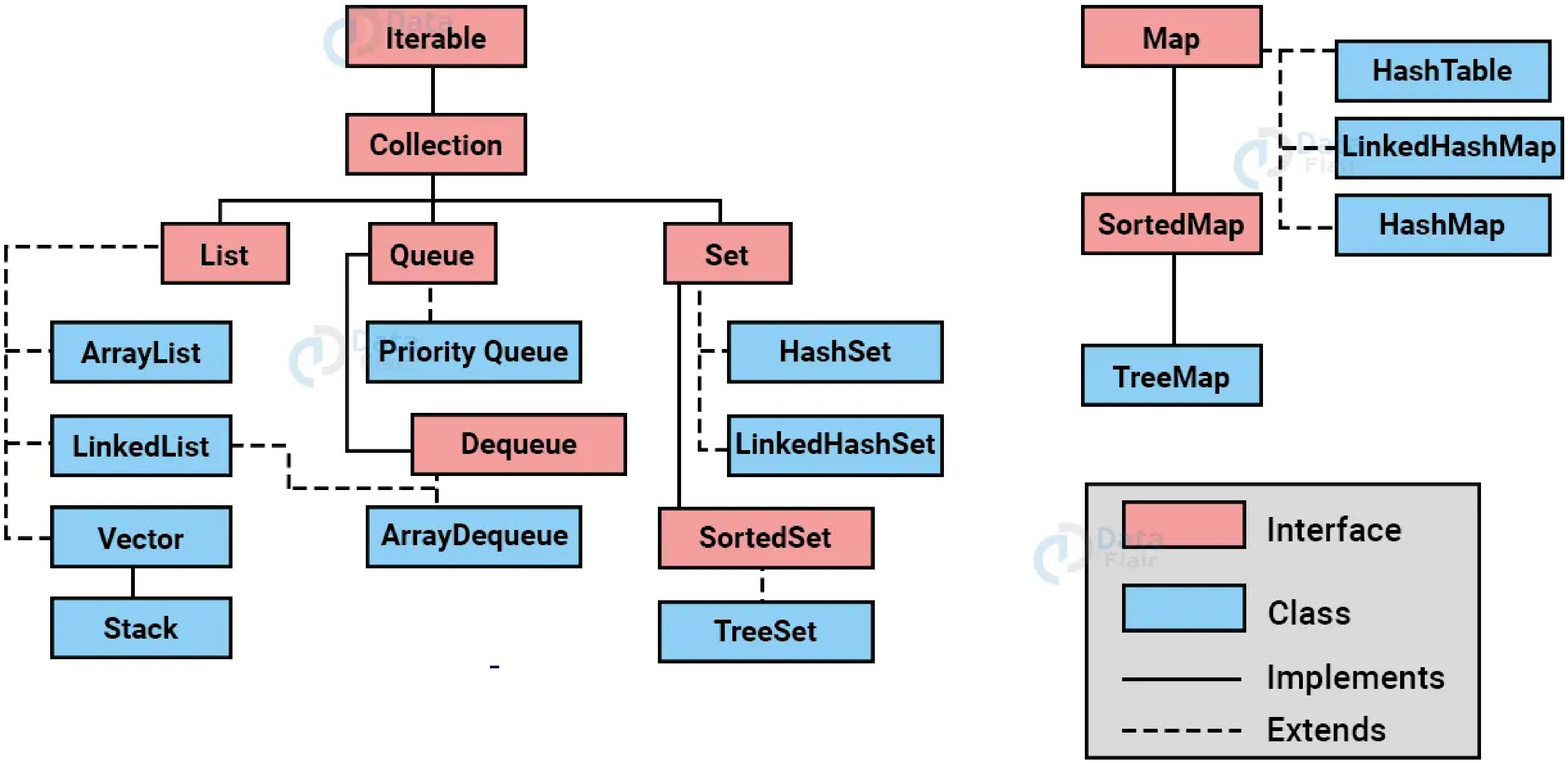
Objectives

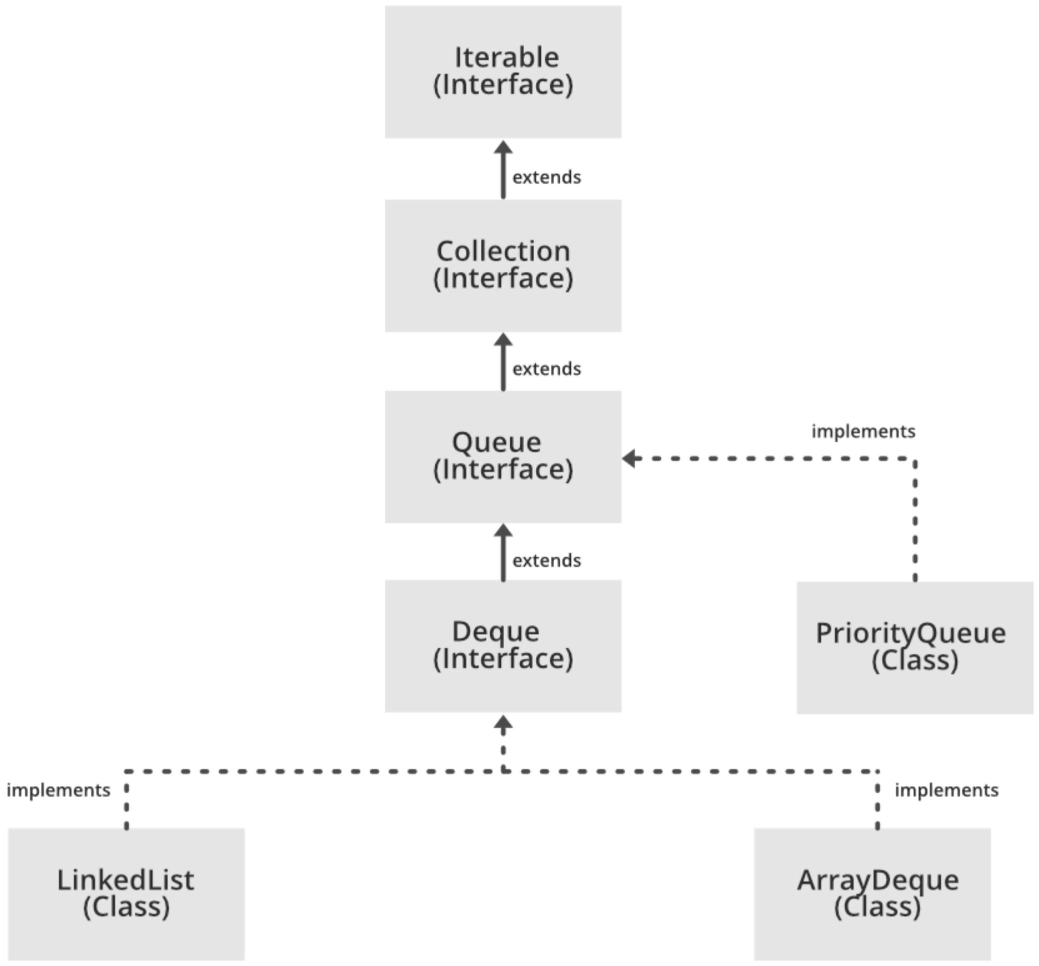
- Queue/Deque Interface
- ArrayQueue Class
- LinkedList Class
- Priority Queue Class

Queue

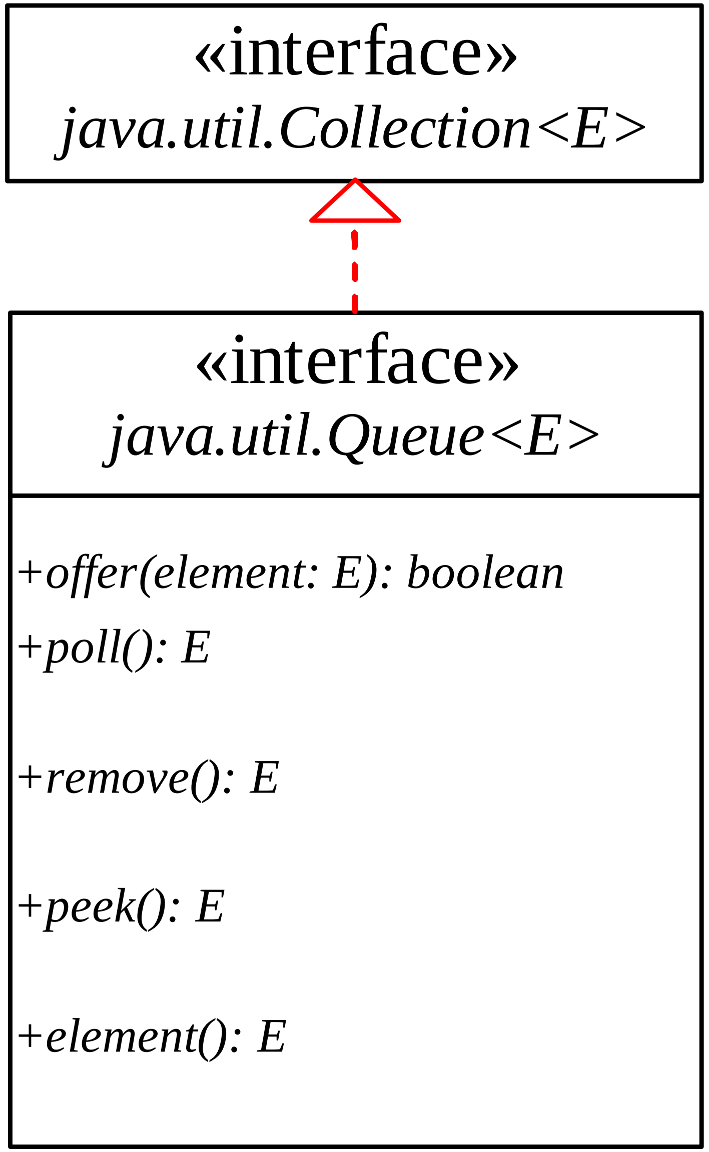
SECTION 1

Hierarchy of Collection Framework in Java




Iterable
(Interface)
extends
Collection
(Interface)
extends
Queue
(Interface)
implements
extends
Deque
(Interface)
PriorityQueue
(Class)
implements
implements
LinkedList
(Class)
ArrayDeque
(Class)

java.util.Queue



«interface»
java.util.Collection<E>

«interface»
java.util.Queue<E>

+offer(element: E): boolean

+poll(): E

+remove(): E

+peek(): E

+element(): E

Inserts an element to the queue.

Retrieves and removes the head of this queue, or null if this queue is empty.

Retrieves and removes the head of this queue and throws an exception if this queue is empty.

Retrieves, but does not remove, the head of this queue, returning null if this queue is empty.

Retrieves, but does not remove, the head of this queue, throwing an exception if this queue is empty.

The Queue Interface

java.Collection.Queue

- Being a Collection subtype all methods in the Collection interface are also available in the Queue interface.
- Since Queue is an interface you need to instantiate a concrete implementation of the interface in order to use it. You can choose between the following Queue implementations in the Java Collections API:
  - **java.util.LinkedList**
  - **java.util.PriorityQueue**

LinkedList (Concrete Class)
PriorityQueue (Concrete Class)

Both implements Queue

LinkedList is a pretty standard queue implementation.
PriorityQueue stores its elements internally according to their natural order (if they implement Comparable), or according to a Comparator passed to the **PriorityQueue**.
There are also **Queue** implementations in the **java.util.concurrent** package, but not our topic.

Here are a few examples of how to create a Queue instance:

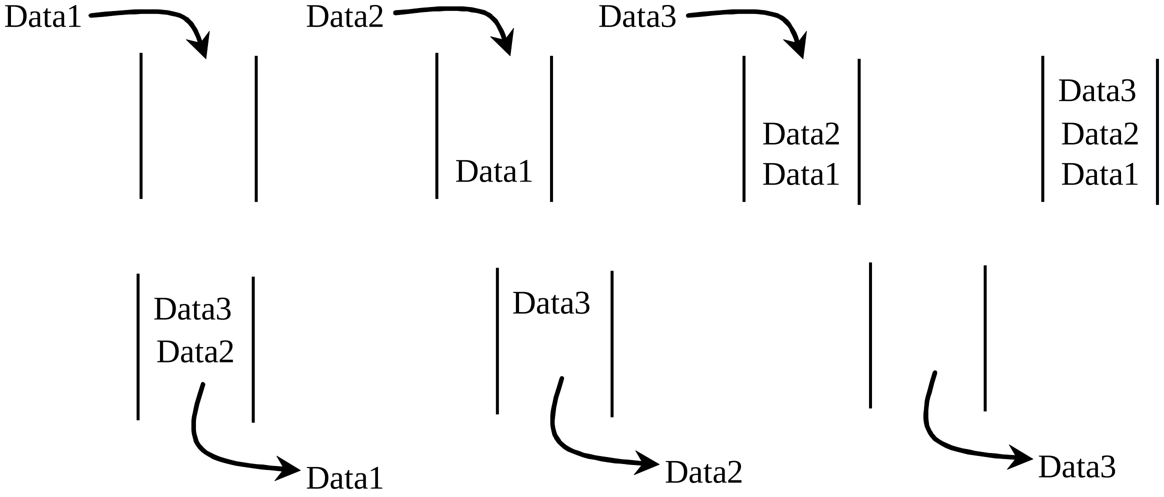
- Queue queueA = new LinkedList();
- Queue queueB = new PriorityQueue();

Demonstration Program

QUEUEDEMO.JAVA

Queue

A queue represents a waiting list. A queue can be viewed as a special type of list, where the elements are inserted into the end (tail) of the queue, and are accessed and deleted from the beginning (head) of the queue. (FIFO)



Applications for Queue

SECTION 2

Uses of Queues in Computing

- Printer queue
- Keyboard input buffer
- GUI event queue (click on buttons, menu items)

Using Queues: Coded Messages

- A **Caesar cipher** is a **substitution code** that encodes a message by shifting each letter in a message by a constant amount **k**
  - If **k** is **5**, **a** becomes **f**, **b** becomes **g**, etc.
    - **Example**: **n qtaj ofaf**
  - Used by Julius Caesar to encode military messages for his generals (around 50 BC)
  - This code is fairly easy to break.

Using Queues: Coded Messages

- **Modern version**: ROT13
  - Each letter is shifted by 13
  - “used in online forums as a means of hiding spoilers, punchlines, puzzle solutions, and offensive materials from the casual glance” (**Wikipedia**)

Using Queues: Coded Messages

- **An improvement**: change how much a letter is shifted depending on where the letter is in the message
- A **repeating key** is a sequence of integers that determine how much each character is shifted
  - Example: consider the repeating key

    3 1 7 4 2 5
  - The first character in the message is shifted by 3, the next by 1, the next by 7, and so on
  - When the key is exhausted, start over at the beginning of the key

Using Queues: Coded Messages

- A **repeating key** is a sequence of integers that determine by how much each character in a message is shifted. Consider the repeating key 3 1 7 4 2 5

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

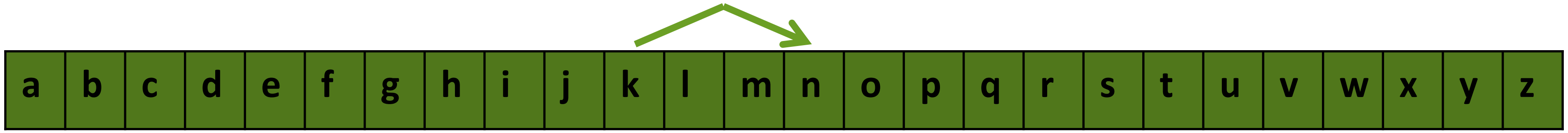
message: knowledge

encoded message:

queue: 3 1 7 4 2 5

Using Queues: Coded Messages

- A **repeating key** is a sequence of integers that determine by how much each character in a message is shifted. Consider the repeating key 3 1 7 4 2 5



a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

message: knowledge

Encoded: n

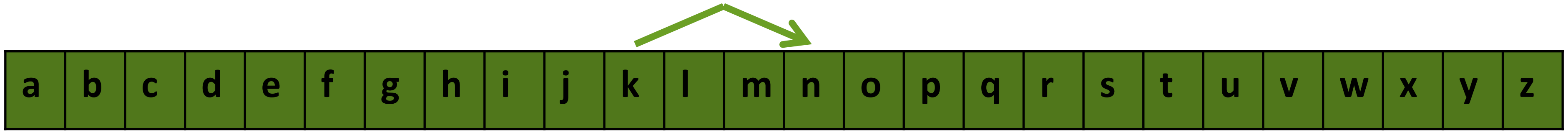
message:

dequeued: 3

queue: 1 7 4 2 5

Using Queues: Coded Messages

- A **repeating key** is a sequence of integers that determine by how much each character in a message is shifted. Consider the repeating key 3 1 7 4 2 5



a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

message: knowledge

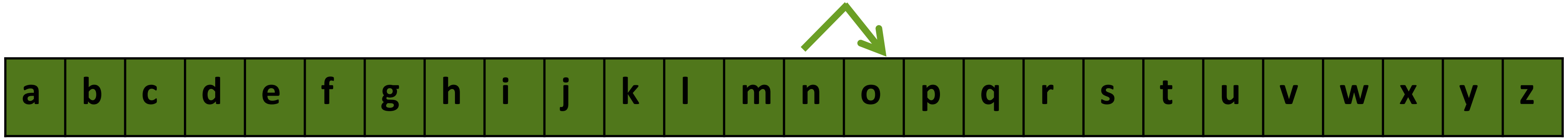
Encoded: n

message:

queue: 1 7 4 2 5 3

Using Queues: Coded Messages

- A **repeating key** is a sequence of integers that determine by how much each character in a message is shifted. Consider the repeating key 3 1 7 4 2 5



a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

message: knowledge

Encoded: no

message:

dequeued: 1

queue: 7 4 2 5 3

Using Queues: Coded Messages

- A **repeating key** is a sequence of integers that determine by how much each character in a message is shifted. Consider the repeating key 3 1 7 4 2 5

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

message: knowledge

Encoded: no

message:

queue: 7 4 2 5 3 1

Using Queues: Coded Messages

- A **repeating key** is a sequence of integers that determine by how much each character in a message is shifted. Consider the repeating key 3 1 7 4 2 5

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

message: knowledge

Encoded: novangjhl

message:

queue: 4 2 5 3 1 7

Using Queues: Coded Messages

- We can use a queue to store the values of the key
  - **dequeue** a key value when needed
  - After using it, **enqueue** it back onto the end of the queue
- So, the queue represents the constantly cycling values in the key

Using Queues: Coded Messages

- See ***Codes.java*** in the sample code page of the course's website
  - Note that there are *two* copies of the key, stored in two separate queues
    - The encoder has one copy
    - The decoder has a separate copy

    - Why?

Demonstration Program

REPEATINGKEY.JAVA

Queue Implementations

SECTION 3

The Queue Abstract Data Type

- enqueue(*e*): Adds element *e* to the back of queue.
- dequeue( ): Removes and returns the first element from the queue
- first( ): Returns the first element of the queue, without removing it (or null if the queue is empty).
- size( ): Returns the number of elements in the queue.
- isEmpty( ): Returns a boolean indicating whether the queue is empty.

Implementing a Queue with a Singly Linked List

```
/** Realization of a FIFO queue as an adaptation of a SinglyLinkedList. */
public class LinkedQueue<E> implements Queue<E> {
  private SinglyLinkedList<E> list = new SinglyLinkedList<>( ); // an empty list
  public LinkedQueue( ) { } // new queue relies on the initially empty list
  public int size( ) { return list.size( ); }
  public boolean isEmpty( ) { return list.isEmpty( ); }
  public void enqueue(E element) { list.addLast(element); }
  public E first( ) { return list.first( ); }
  public E dequeue( ) { return list.removeFirst( ); }
}
```

Double-Ended Queues

SECTION 4

The Deque Abstract Data Type

addFirst(*e*): Insert a new element *e* at the front of the deque.

addLast(*e*): Insert a new element *e* at the back of the deque.

removeFirst( ): Remove and return the first element of the deque (or null if the deque is empty).

removeLast( ): Remove and return the last element of the deque (or null if the deque is empty).

Additionally, the deque ADT will include the following accessors:

first( ): Returns the first element of the deque, without removing it (or null if the deque is empty).

last( ): Returns the last element of the deque, without removing it (or null if the deque is empty).

size( ): Returns the number of elements in the deque.

isEmpty( ): Returns a boolean indicating whether the deque is empty

ArrayDeque

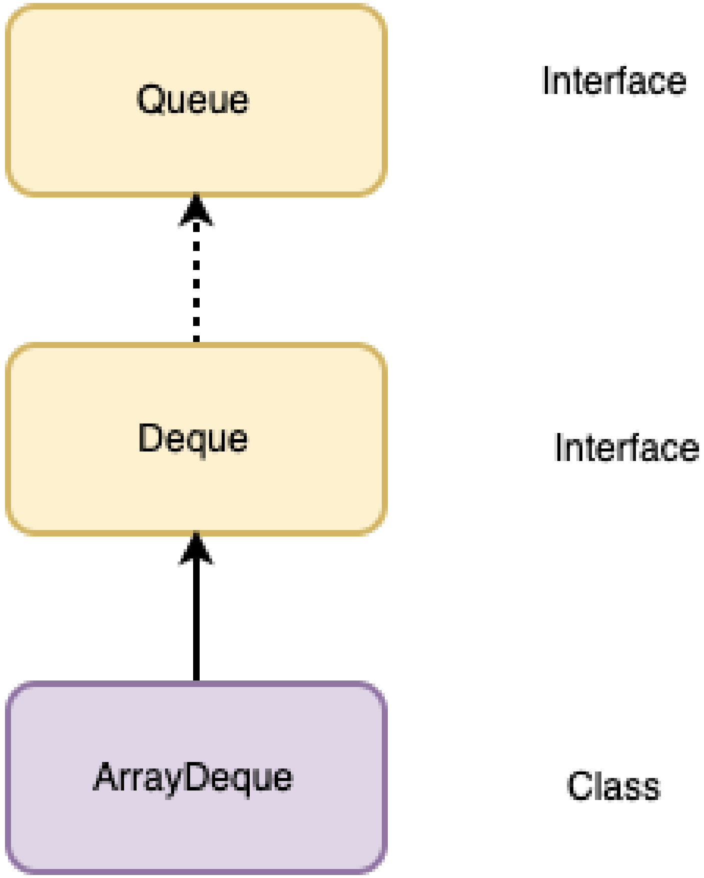
SECTION 5

ArrayDeque

- ArrayDeque is an array
- ArrayDeque is a queue
- ArrayDeque is a list
- ArrayDeque is a double-ended queue

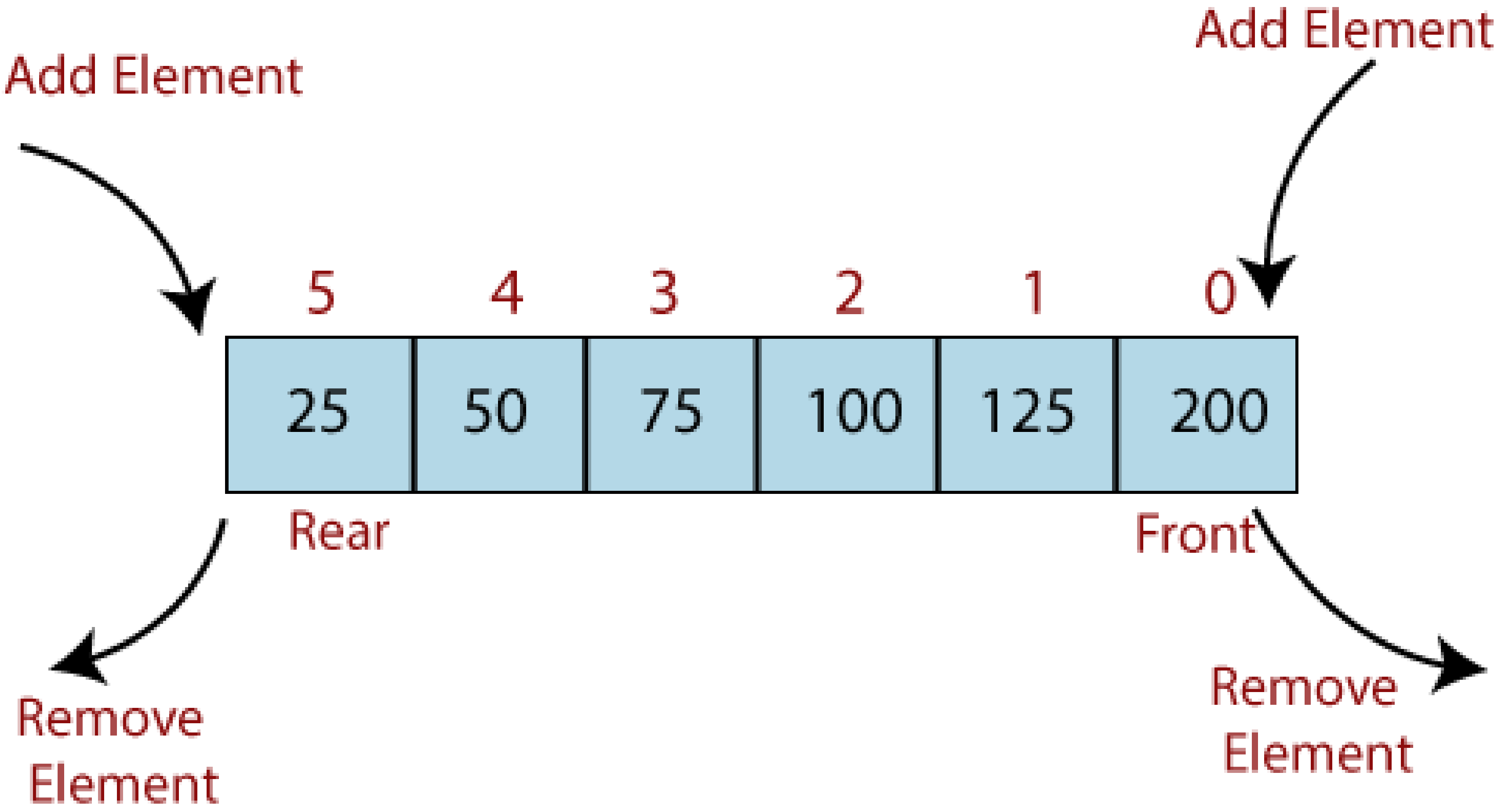
ArrayDeque

- Random Accessed
- Push and pop like a stack
- Enqueue and dequeue like a queue
- Push, pop, shift, unshift like JavaScript array (push_back, pop, removeFirst(), addFirst())



Disadvantage

- Slow Array Operations.


Add Element
Add Element
5
4
3
2
1
0
25
50
75
100
125
200
Rear
Front
Remove Element
Remove Element

Demonstration Program

USINGARRAYDEQUE.JAVA

Demonstration Program

SUBARRAY.JAVA

Demonstration Program

USEQUEUE.JAVA

LinkedList (ListQueue/Deque)

SECTION 6

LinkedList

- LinkedList by its nature is a list and is a double-ended queue
- Not efficient if too many random access.
- Works better if for Stack or Queue operations.

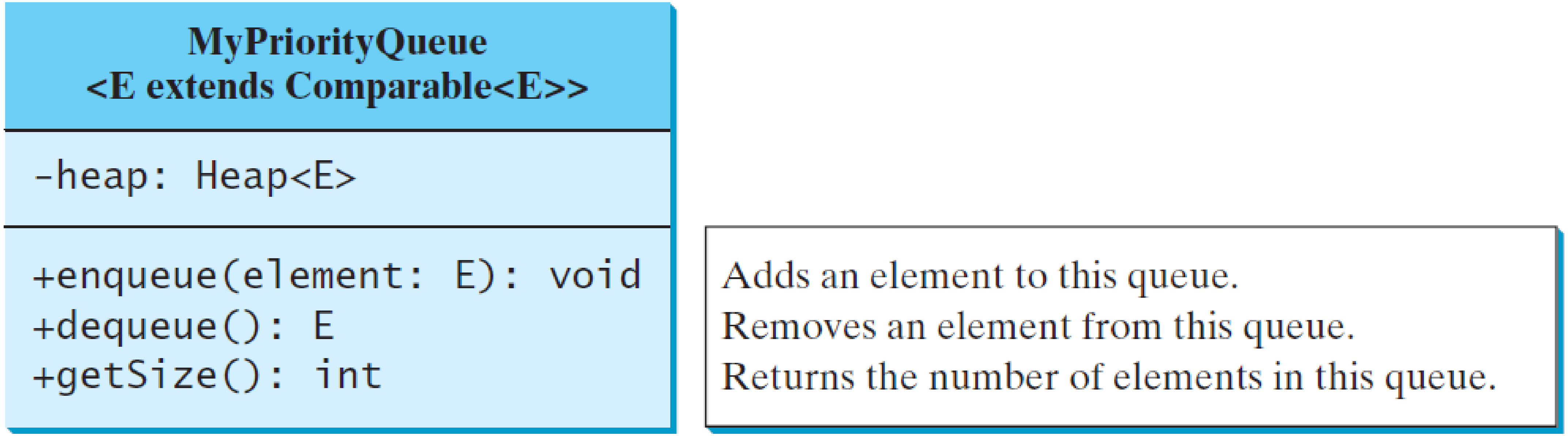
Priority Queue

SECTION 7

Priority Queue

- A regular queue is a first-in and first-out data structure. Elements are appended to the end of the queue and are removed from the beginning of the queue. In a priority queue, elements are assigned with priorities. When accessing elements, the element with the highest priority is removed first. A priority queue has a largest-in, first-out behavior.
- For example, the emergency room in a hospital assigns patients with priority numbers; the patient with the highest priority is treated first.

Demo Program:



MyPriorityQueue
<E extends Comparable<E>>

```
-heap: Heap<E>
```

```
+enqueue(element: E): void
+dequeue(): E
+getSize(): int
```

Adds an element to this queue.
Removes an element from this queue.
Returns the number of elements in this queue.

Demonstration Program

MYPRIORITYQUEUE.JAVA, TESTMYPRIORITYQUEUE.JAVA

PriorityQueue is based on Heap

(A Heap with Queue Interface, Heap has priority key.)

The **java.util.PriorityQueue** class is an unbounded priority queue based on a priority heap. Following are the important points about PriorityQueue:

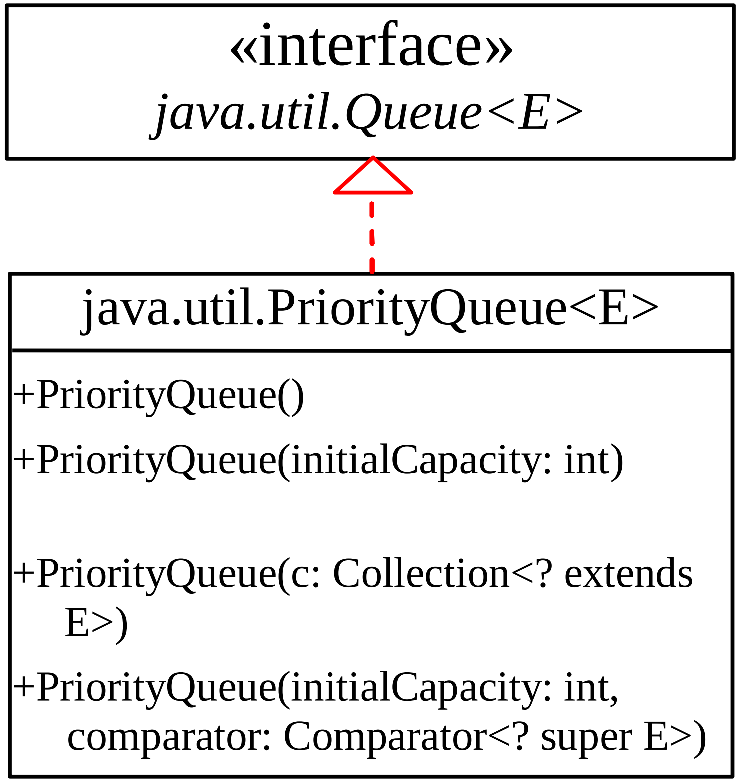
- The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.
- A priority queue does not permit null elements.
- A priority queue relying on natural ordering also does not permit insertion of non-comparable objects.

PriorityQueue Class

SECTION 8

PriorityQueue UML

PriorityQueue->Heap/Comparable/Queue->LinkedList



«interface»
java.util.Queue<E>

java.util.PriorityQueue<E>

+PriorityQueue()	Creates a default priority queue with initial capacity 11.
+PriorityQueue(initialCapacity: int)	Creates a default priority queue with the specified initial capacity.
+PriorityQueue(c: Collection<? extends E>)	Creates a priority queue with the specified collection.
+PriorityQueue(initialCapacity: int, comparator: Comparator<? super E>)	Creates a priority queue with the specified initial capacity and the comparator.

Good Data Structure for Priority Queue

- Add
- Remove
- Adjust

Priority Queue in JDK

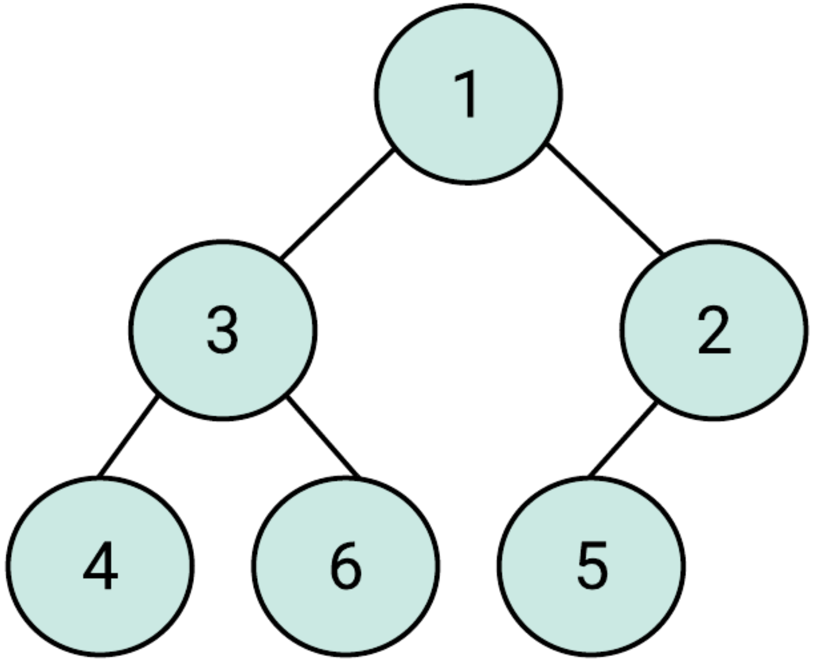
- As a conclusion, in java JDK we have **PriorityQueue**, an unbounded priority queue based on a priority heap.
- The elements of the priority queue are ordered according to their natural ordering, or by a comparator provided at queue construction time, depending on which constructor is used.

Demonstration Program

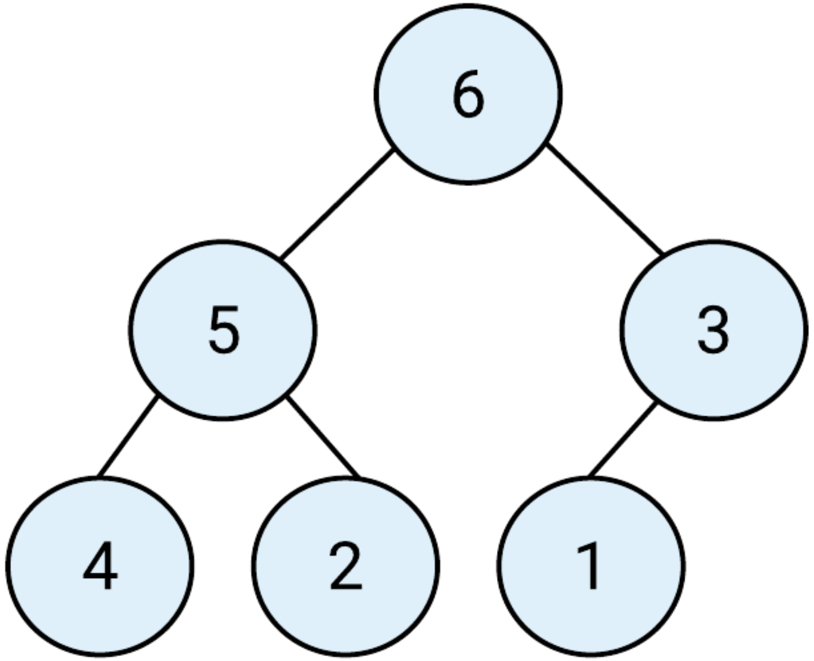
PRIORITYQUEUEDEMO.JAVA

Demonstration Program

USEPRIORITYQUEUE.JAVA



Min heap



Max Heap