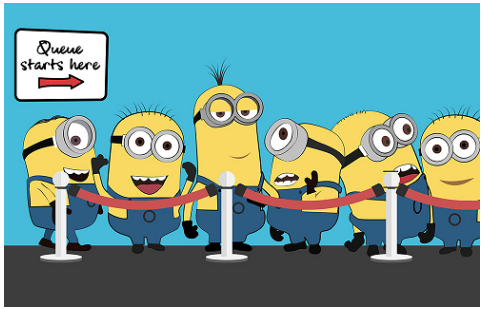
CSE214 Data Structures

Queues

YoungMin Kwon

SUNY Korea

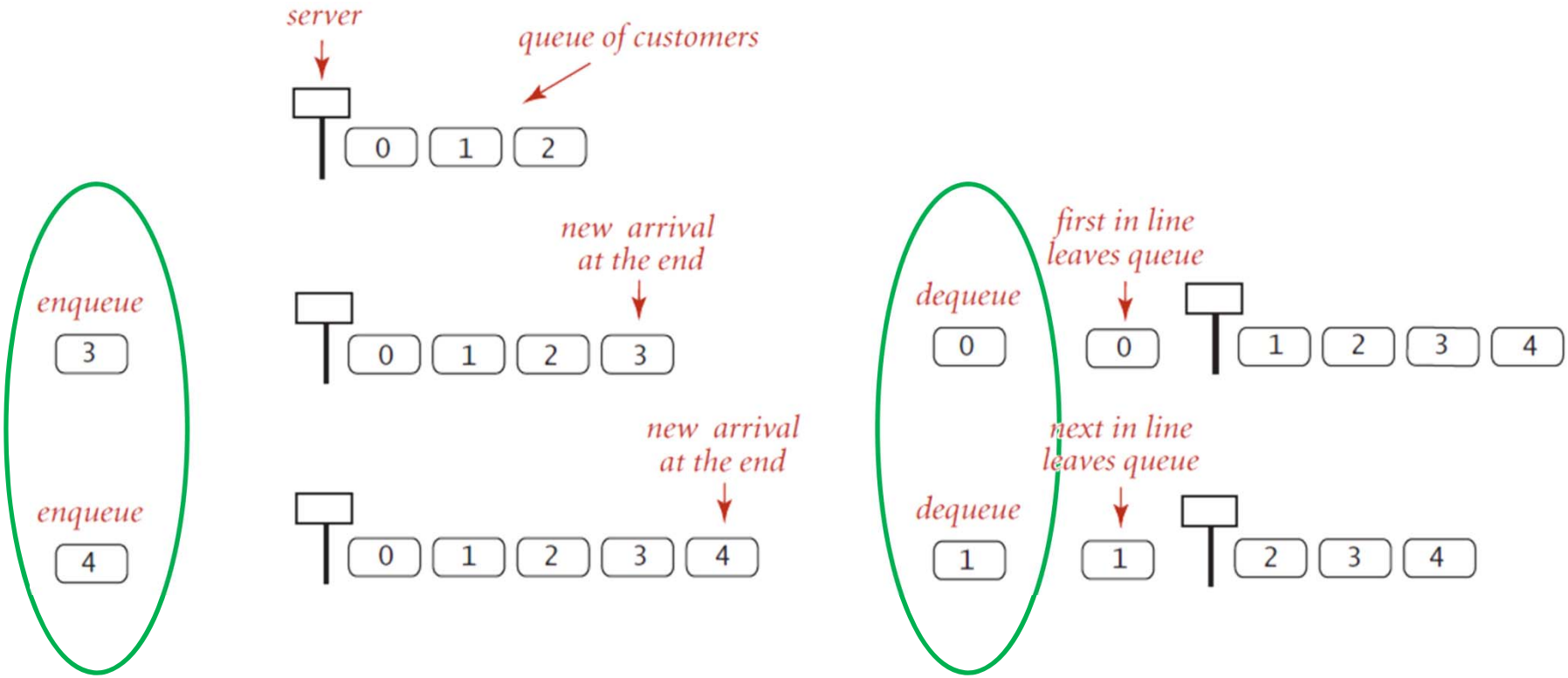
Queues



- Queue
  - Queue is a collection of objects that are inserted and removed according to the First-In, First-Out (FIFO) principle
- Main operations on queues
  - enqueue: adding an element to a queue
  - dequeue: removing the element that has been in the queue the longest

SUNY Korea

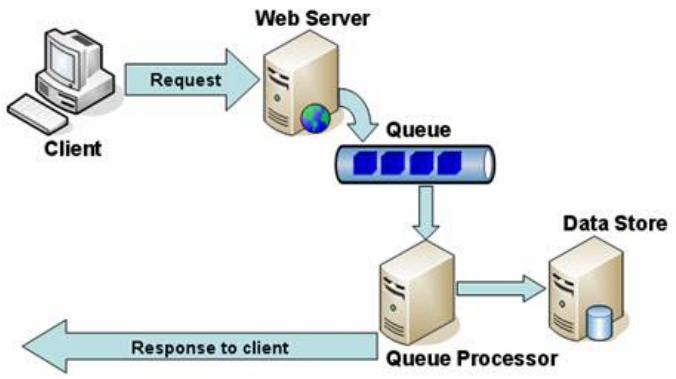
Queue Operations



Examples of Queues

- A line of people waiting to get in to a store

- A call service system at a customer service center
- Web server
- Networked printer …



SUNY Korea

Queue Abstract Data Type

enqueue(e): Adds element e to the back of queue.

dequeue( ): Removes and returns the first element from the queue (or null if the queue is empty).

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.

SUNY Korea

Queue Operation Example

Method	Return Value	first $\leftarrow Q \leftarrow$ last
enqueue(5)	–	(5)
enqueue(3)	–	(5, 3)
size( )	2	(5, 3)
dequeue( )	5	(3)
isEmpty( )	false	(3)
dequeue( )	3	( )
isEmpty( )	true	( )
dequeue( )	null	( )
enqueue(7)	–	(7)
enqueue(9)	–	(7, 9)
first( )	7	(7, 9)
enqueue(4)	–	(7, 9, 4)

SUNY Korea

Interface Queue

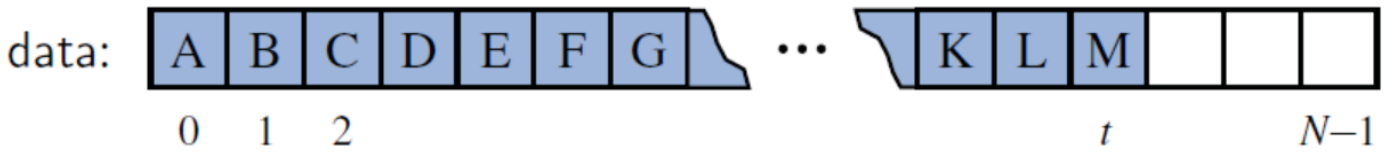
```
public interface Queue<E> {
    public int size();
    public boolean isEmpty();
    public void enqueue(E e);
    public E dequeue();
    public E first();
}
```

Our Queue ADT	Interface java.util.Queue	
	throws exceptions	returns special value
enqueue(e)	add(e)	offer(e)
dequeue( )	remove( )	poll( )
first( )	element( )	peek( )
size( )	size( )	
isEmpty( )	isEmpty( )	

SUNY Korea

Array-Based Queue Implementation

- Implementation 1
  - Enqueue: store elements at the end of existing elements (O(1))
  - Dequeue: remove the first element and shift the rest (O(n))



SUNY Korea

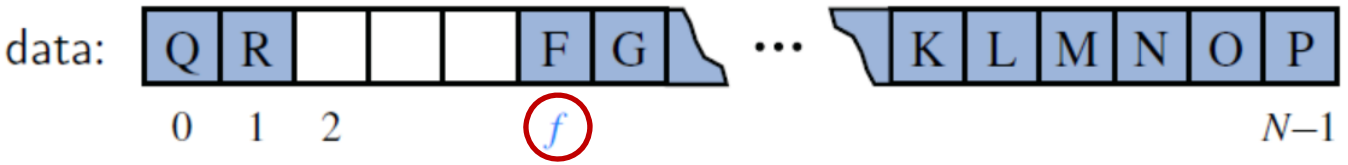
Array-Based Queue Implementation

- Implementation 2
  - Enqueue: store elements at the end of existing elements (O(1))
  - Dequeue: remove the element at index *f* and advance *f* by 1 (O(1))
  - *f* will drift backward and the back of the queue would eventually reach the end of the array

SUNY Korea

Array-Based Queue Implementation

- Implementation 3
  - Using an array circularly
  - The contents of the queue wraps around the end of the array as necessary
    - New elements to be placed at index *N*, *N*+1, … are placed at 0, 1, …
    - Easy to implement: $f = (f + 1) \% N$



SUNY Korea

Array-Based Queue Implementation

- Dequeue
  - Remove the element at index *f*
  - *f* = (*f* + 1) % *N*
  - Decrease size
- Enqueue
  - Add element to (*f* + *size*) % *N*
  - Increase size

SUNY Korea

```java
@SuppressWarnings("unchecked")
public class ArrayQueue<E> implements Queue<E> {
    protected static final int CAPACITY = 16;
    protected E[] data;
    protected int size;
    protected int f;

    public ArrayQueue()              { this(CAPACITY); }
    public ArrayQueue(int capacity) { data = (E[])new Object[capacity]; }

    //
    // Interface Queue
    //
    public int size()         { return size; }
    public boolean isEmpty() { return size == 0; }
    public E first()  {
        if(isEmpty())
            return null;
        return data[f];
    }
```

SUNY Korea

```
public void enqueue(E e) {
    if(size == data.length)
        throw new IllegalStateException("Queue is full");
    int avail = (f + size) % data.length;
    data[avail] = e;
    size++;
}

public E dequeue() {
    if(isEmpty())
        throw new IllegalStateException("Queue is empty");
    E ret = data[f];
    data[f] = null;
    f = (f + 1) % data.length;
    size--;
    return ret;
}
```

SUNY Korea

```
    protected static void onFalseThrow(boolean b) {
        if(!b)
            throw new RuntimeException("Error: unexpected");

    }

    public static void main(String[] args) {
        ArrayQueue<Integer> queue = new ArrayQueue<Integer>();

        queue.enqueue(1);
        queue.enqueue(2);
        queue.enqueue(3);
        queue.enqueue(4);

        onFalseThrow(queue.dequeue() == 1);
        onFalseThrow(queue.dequeue() == 2);
        onFalseThrow(queue.dequeue() == 3);
        onFalseThrow(queue.dequeue() == 4);

        System.out.println("Success!");
    }
}
```

SUNY Korea

Big-Oh of Array-Based Queue

Method	Running Time
size	$O(1)$
isEmpty	$O(1)$
first	$O(1)$
enqueue	$O(1)$
dequeue	$O(1)$

SUNY Korea

List-Based Queue Implementation

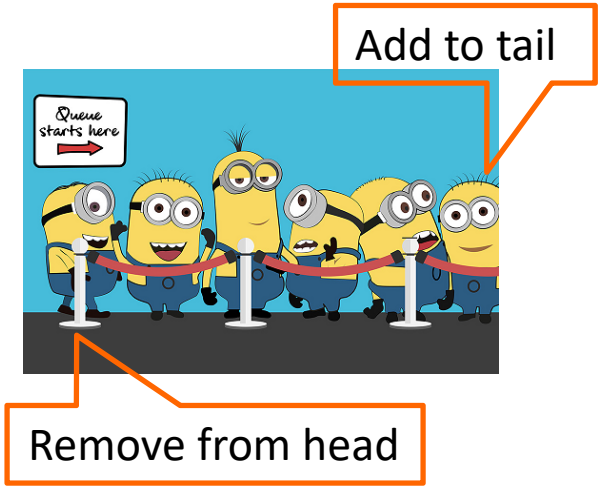
- Use a singly linked list
  - No limit in the queue capacity
  - O(1) enqueue and dequeue operations

SUNY Korea

```
public class ListQueue<E> implements Queue<E> {
    //Node for singly linked list
    private static class Node<E> {
        public Node<E> next;
        public E e;
        public Node() {}
        public Node(E e, Node<E>next) { this.e = e; this.next = next; }
    }

    //sentinel
    private Node<E> head,   //head is a sentinel
                    tail;   //tail points to the last node or
                            //head when empty
    private int size;

    public ListQueue() {
        head = tail = new Node<E>();
        size = 0;
    }

    // Interface Queue
    //
    public int size()         { return size; }
    public boolean isEmpty() { return size == 0; }
```

SUNY Korea

```java
//add e after tail
public void enqueue(E e) {
    tail.next = new Node<E>(e, null);
    tail = tail.next;
    size++;
}

//remove the node after head
public E dequeue() {
    if(isEmpty())
        return null;
    Node<E> f = head.next;
    head.next = f.next;
    size--;
    //if tail is removed, reset tail to head
    if(size == 0)
        tail = head;
    return f.e;
}

public E first() {
    if(isEmpty())
        return null;
    return head.next.e;
}
```



SUNY Korea

Josephus Problem (Hot Potato Game)

- Given a number k and n players
  - Each player is passing a potato to the next player k times
  - The person received the potato at the k^{th} step is out of the game
  - The play repeats until only 1 play is left
  - The last player left is the winner
- We can simulate the Josephus problem using a queue

SUNY Korea

```
public class Josephus {

    public static <E> E Play(Queue<E> queue, int k) {
        if(queue.isEmpty())
            return null;

        //find victims until only 1 player is left
        while(queue.size() > 1) {

            //rotate the potato k-1 times
            for(int i = 0; i < k-1; i++)
                queue.enqueue(queue.dequeue());

            //evict the victim (k-th player)
            E e = queue.dequeue();
            System.out.println("    " + e + "is out");
        }

        //the last player is the winner
        return queue.dequeue();
    }
```

SUNY Korea

```
    public static <E> Queue<E> buildQueue(E a[]) {
        Queue<E> queue = new ListQueue<E>();

        //add all elements in a to queue
        for(E e : a)
            queue.enqueue(e);

        //return the queue
        return queue;
    }

    public static void main(String[] args) {
        String[] a1 = {"Alice", "Bob", "Cindy", "Doug", "Ed", "Fred"};
        String[] a2 = {"Gene", "Hope", "Irene", "Jack", "Kim", "Lance"};
        String[] a3 = {"Mike", "Roberto"};

        System.out.println("First  winnder is " + Play(buildQueue(a1), 3));
        System.out.println("Second winnder is " + Play(buildQueue(a2), 10));
        System.out.println("Third  winnder is " + Play(buildQueue(a3), 7));
    }
}
```

SUNY Korea

Josephus Problem

Expected output

```
> java Josephus

    Cindyis out
    Fredis out
    Dougis out
    Bobis out
    Edis out
First  winnder is Alice
    Jackis out
    Ireneis out
    Lanceis out
    Geneis out
    Kimis out
Second winnder is Hope
    Mikeis out
Third  winnder is Roberto
```

SUNY Korea

Double-Ended Queues

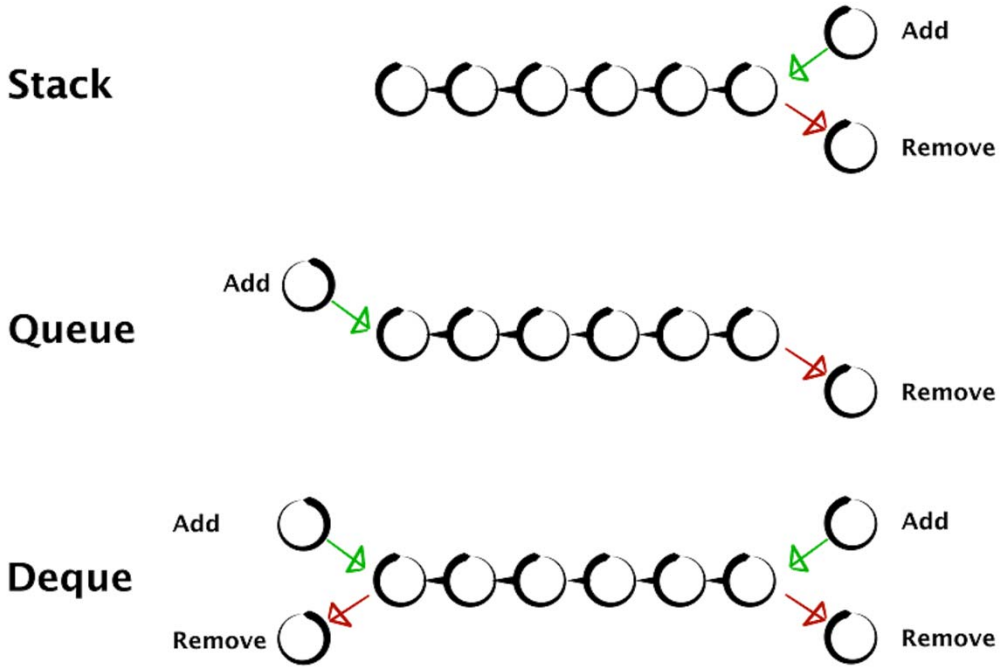
- Deque (Double-Ended Queue)
  - Supports insertion and deletion at both the front and the back of the queue

- Example
  - Guests removed from the head of the waitlist
    - They don't like the table and want to be relisted back to the head of the queue
  - Guests at the tail of the waitlist queue grow impatient and left

SUNY Korea

Deque

- Stack vs Queue vs Deque



SUNY Korea

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).

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.

SUNY Korea

Deque Operation Example

Method	Return Value	D
addLast(5)	–	(5)
addFirst(3)	–	(3, 5)
addFirst(7)	–	(7, 3, 5)
first( )	7	(7, 3, 5)
removeLast( )	5	(7, 3)
size( )	2	(7, 3)
removeLast( )	3	(7)
removeFirst( )	7	( )
addFirst(6)	–	(6)
last( )	6	(6)
addFirst(8)	–	(8, 6)
isEmpty( )	false	(8, 6)
last( )	6	(8, 6)

SUNY Korea

Interface Deque

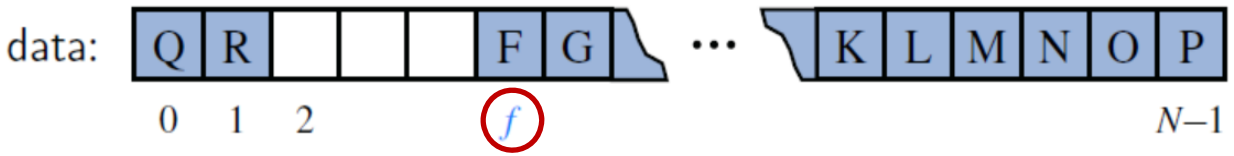
```
public interface Deque<E> {
    public int size();
    public boolean isEmpty();
    public E first();
    public E last();
    public void addFirst(E e);
    public void addLast(E e);
    public E removeFirst();
    public E removeLast();
}
```

SUNY Korea

Implementing a Deque

- First element
  - Index: f
  - After add: $f = (f - 1 + N) \% N$, $size{+}{+}$
  - After remove: $f = (f + 1) \% N$, $size{-}{-}$
- Last element
  - Index: $(f + size - 1 + N) \% N$
  - After add: $(f + size) \% N$, $size{+}{+}$
  - After remove: $(f + size - 2 + N) \% N$, $size{-}{-}$

Index of last element before size change



SUNY Korea

```java
@SuppressWarnings("unchecked")
public class ArrayDeque<E> implements Deque<E> {
    protected static final int CAPACITY = 16;
    protected E[] data;
    protected int size;
    protected int f;

    public ArrayDeque()              { this(CAPACITY); }
    public ArrayDeque(int capacity) { data = (E[])new Object[capacity]; }

    //Interface Deque
    public int size()        { return size; }
    public boolean isEmpty() { return size == 0; }

    private int index(int i) {  //take care of the wrapping
        return (i + data.length) % data.length;
    }

    private int lastIndex() {
        return index(f + size - 1);
    }
```

SUNY Korea

```
public E first()  {
    if(isEmpty())
        throw new IllegalStateException("Deque is empty");
    return data[f];
}
public E last()  {
    if(isEmpty())
        throw new IllegalStateException("Deque is empty");
    return data[lastIndex()];
}

public void addFirst(E e) {
    if(size == data.length)
        throw new IllegalStateException("Deque is full");
    f = index(f - 1);
    data[f] = e;
    size++;
}
public void addLast(E e) {
    if(size == data.length)
        throw new IllegalStateException("Deque is full");
    int i = index(lastIndex() + 1);
    data[i] = e;
    size++;
}
```

SUNY Korea

```
public E removeFirst() {
    if(isEmpty())
        throw new IllegalStateException("Deque is empty");
    E ret = data[f];
    data[f] = null;
    f = index(f + 1);
    size--;
    return ret;
}

public E removeLast() {
    if(isEmpty())
        throw new IllegalStateException("Deque is empty");
    int i = lastIndex();
    E ret = data[i];
    data[i] = null;
    size--;
    return ret;
}
```

SUNY Korea

```
    protected static void onFalseThrow(boolean b) {
        if(!b)
            throw new RuntimeException("Error: unexpected");
    }

    public static void main(String[] args) {
        Deque<Integer> deque = new ArrayDeque<Integer>();

        deque.addFirst(1);
        deque.addLast(2);
        deque.addFirst(3);
        deque.addLast(4);

        onFalseThrow(deque.size() == 4);
        onFalseThrow(deque.first() == 3);
        onFalseThrow(deque.last() == 4);
        onFalseThrow(deque.removeFirst() == 3);
        onFalseThrow(deque.removeLast() == 4);
        onFalseThrow(deque.removeLast() == 2);
        onFalseThrow(deque.removeLast() == 1);
        onFalseThrow(deque.size() == 0);

        System.out.println("Success!");
    }
}
```

SUNY Korea

Deque in Java

- `java.util.ArrayDeque, java.util.LinkedList`

Our Deque ADT	Interface java.util.Deque	
	throws exceptions	returns special value
first( )	getFirst( )	peekFirst( )
last( )	getLast( )	peekLast( )
addFirst(e)	addFirst(e)	offerFirst(e)
addLast(e)	addLast(e)	offerLast(e)
removeFirst( )	removeFirst( )	pollFirst( )
removeLast( )	removeLast( )	pollLast( )
size( )	size( )	
isEmpty( )	isEmpty( )	

SUNY Korea

Priority Queues

- Priority queue
  - A collection of prioritized elements
  - Arbitrary element insertion
  - Removal of the first priority element
- Key
  - Elements in priority queues are associated with a key
  - Elements with the minimal key will be the next to be removed
  - Keys are comparable (total order)

SUNY Korea

The Priority Queue ADT

insert(k, v): Creates an entry with key k and value v in the priority queue.

min( ): Returns (but does not remove) a priority queue entry (k,v) having minimal key; returns null if the priority queue is empty.

removeMin( ): Removes and returns an entry (k,v) having minimal key from the priority queue; returns null if the priority queue is empty.

size( ): Returns the number of entries in the priority queue.

isEmpty( ): Returns a boolean indicating whether the priority queue is empty.

SUNY Korea

Priority Queue Operation Example

Method	Return Value	Priority Queue Contents
insert(5,A)		{ (5,A) }
insert(9,C)		{ (5,A), (9,C) }
insert(3,B)		{ (3,B), (5,A), (9,C) }
min( )	(3,B)	{ (3,B), (5,A), (9,C) }
removeMin( )	(3,B)	{ (5,A), (9,C) }
insert(7,D)		{ (5,A), (7,D), (9,C) }
removeMin( )	(5,A)	{ (7,D), (9,C) }
removeMin( )	(7,D)	{ (9,C) }
removeMin( )	(9,C)	{ }
removeMin( )	null	{ }
isEmpty( )	true	{ }

SUNY Korea

Implementing a Priority Queue

- The entry composite

```
/** Interface for a key-value pair. */
public interface Entry<K,V> {
  K getKey( );   // returns the key stored in this entry
  V getValue( ); // returns the value stored in this entry
}
```

SUNY Korea

Priority Queue Interface

```
public interface PriorityQueue<K, V> {
    int size();
    boolean isEmpty();
    Entry<K, V> insert(K key, V value)
                throws IllegalArgumentException;
    Entry<K, V> min();
    Entry<K, V> removeMin();
}
```

SUNY Korea

Comparing Keys (Total Order)

- Partial order $\leq$
  - **Reflexive property**: $k \leq k$.
  - **Antisymmetric property**: if $k_1 \leq k_2$ and $k_2 \leq k_1$, then $k_1 = k_2$.
  - **Transitive property**: if $k_1 \leq k_2$ and $k_2 \leq k_3$, then $k_1 \leq k_3$.
- Total order
  - A partial order $\leq$ is a total order if for any keys k_1 and k_2
    - **Comparability property:** $k_1 \leq k_2$ or $k_2 \leq k_1$.

SUNY Korea

Interface java.lang.Comparable

- Define the natural ordering of instances
  - E.g. orders among numbers
  - E.g. the lexicographic ordering of strings

- `a.compareTo(b)` must return an integer *i* such that
  - $i < 0$ designates that $a < b$.
  - $i = 0$ designates that $a = b$.
  - $i > 0$ designates that $a > b$.

SUNY Korea

Interface java.util.Comparator

- Provides an order other than the natural ordering
  - E.g. order strings by their length
- External to the class
- `compare(a, b)`
  - Returns an integer with a similar meaning to `compareTo`

SUNY Korea

Interface java.util.Comparator

- Example

```
public class StringLengthComparator implements Comparator<String> {
  /** Compares two strings according to their lengths. */
  public int compare(String a, String b) {
    if (a.length( ) < b.length( )) return −1;
    else if (a.length( ) == b.length( )) return 0;
    else return 1;
  }
}
```

SUNY Korea

Comparators and Priority Queue ADT

- Priority Queue ADT
  - Allows users to choose key types
  - Allows users to use custom key Comparators
  - Makes the priority queue general and reusable

SUNY Korea

Comparators and Priority Queue ADT

- Default comparator for the natural ordering

```
import java.util.Comparator;

public class DefaultComparator<E extends Comparable<E>>
             implements Comparator<E> {

    public int compare(E a, E b) throws ClassCastException {
        return ((Comparable<E>)a).compareTo(b);
    }

}
```

SUNY Korea

AbstractPriorityQueue Base Class

```
import java.util.Comparator;

public abstract class AbstractPriorityQueue<K extends Comparable<K>, V>
                      implements PriorityQueue<K, V> {

    protected static class PQEntry<K extends Comparable<K>, V>
                           implements Entry<K, V> {
        private K k;
        private V v;

        public PQEntry(K key, V value)   { k = key; v = value; }
        public K getKey()                { return k; }
        public V getValue()              { return v; }

        protected void setKey(K key)     { k = key; }
        protected void setValue(V value) { v = value; }
    }
```

SUNY Korea

```
    private Comparator<K> comp;

    protected AbstractPriorityQueue() {
        this(new DefaultComparator<K>());
    }
    protected AbstractPriorityQueue(Comparator<K> c) {
        comp = c;
    }

    protected int compare(Entry<K,V> a, Entry<K,V> b) {
        return comp.compare(a.getKey(), b.getKey());
    }

    protected boolean checkKey(K key) throws IllegalArgumentException {
        try {
            return comp.compare(key, key) == 0;
        } catch(ClassCastException e) {
            throw new IllegalArgumentException("Incompatible key");
        }
    }

    public boolean isEmpty() { return size() == 0; }
}
```

SUNY Korea

Priority Queue with an Unsorted List

```
import java.util.Comparator;

public class UnsortedPriorityQueue<K extends Comparable<K>, V>
             extends AbstractPriorityQueue<K, V>{
    private PositionalList<Entry<K,V>> list =
                                new PositionalDblLinkedList<>();

    public UnsortedPriorityQueue() { super(); }
    public UnsortedPriorityQueue(Comparator<K> comp) {
        super(comp);
    }

    private Position<Entry<K,V>> findMin() {
        Position<Entry<K,V>> small = list.first();
        for(Position<Entry<K,V>> walk : list) {
            if(compare(walk.getElement(), small.getElement()) < 0)
                small = walk;
        }
        return small;
    }
```

SUNY Korea

```
public int size() {
    return list.size();
}

public Entry<K,V> insert(K key, V value)
                    throws IllegalArgumentException {
    checkKey(key);
    Entry<K,V> newest = new PQEntry<K,V>(key, value);
    list.addLast(newest);
    return newest;
}

public Entry<K,V> min() {
    if(list.isEmpty())
        return null;
    return findMin().getElement();
}

public Entry<K,V> removeMin() {
    if(list.isEmpty())
        return null;
    return list.remove(findMin());
}
```

SUNY Korea

```
    private static void onFalseThrow(boolean b) {
        if(!b)
            throw new RuntimeException("Error: unexpected");
    }

    public static void main(String[] args) {
        PriorityQueue<Integer, Integer> pq =
                        new UnsortedPriorityQueue<Integer, Integer>();
        pq.insert(3, 3);
        pq.insert(2, 2);
        pq.insert(1, 1);
        pq.insert(4, 4);
        pq.insert(5, 5);

        for(int i = 1; i <= 5; i++) {
            onFalseThrow(pq.min().getValue() == i);
            onFalseThrow(pq.removeMin().getValue() == i);
        }
        System.out.println("Success!");
    }
}
```

SUNY Korea

Priority Queue with an Unsorted List

- Running time of `UnsortedPriorityQueue`

Method	Running Time
size	$O(1)$
isEmpty	$O(1)$
insert	$O(1)$
min	$O(n)$
removeMin	$O(n)$

SUNY Korea

Priority Queue with a Sorted List

```
import java.util.Comparator;

public class SortedPriorityQueue<K extends Comparable<K>, V>
    extends AbstractPriorityQueue<K, V>{
    private PositionalList<Entry<K,V>> list =
                          new PositionalDblLinkedList<>();

    public SortedPriorityQueue()                    { super(); }
    public SortedPriorityQueue(Comparator<K> comp) { super(comp); }
    public int size() { return list.size(); }

    public Entry<K,V> min() {
        if(list.isEmpty())
            return null;
        return list.first().getElement();
    }
    public Entry<K,V> removeMin() {
        if(list.isEmpty())
            return null;
        return list.remove(list.first());
    }
```

SUNY Korea

```
public Entry<K,V> insert(K key, V value)
                    throws IllegalArgumentException {
    checkKey(key);
    //new Entry
    Entry<K,V> newest = new PQEntry<K,V>(key, value);

    //find where newest should be inserted
    Position<Entry<K,V>> pos = null;
    for(Position<Entry<K,V>> walk : list) {
        if(compare(newest, walk.getElement()) < 0) {
            pos = walk;
            break;
        }
    }

    //add it to the last position or insert it before pos
    if(pos == null)
        list.addLast(newest);       //new key is the largest
    else
        list.addBefore(pos, newest);//new key goes before pos

    return newest;
}
```

SUNY Korea

```
    private static void onFalseThrow(boolean b) {
        if(!b)
            throw new RuntimeException("Error: unexpected");
    }

    public static void main(String[] args) {
        PriorityQueue<Integer, Integer> pq =
                        new SortedPriorityQueue<Integer, Integer>();
        pq.insert(3, 3);
        pq.insert(2, 2);
        pq.insert(1, 1);
        pq.insert(4, 4);
        pq.insert(5, 5);

        for(int i = 1; i <= 5; i++) {
            onFalseThrow(pq.min().getValue() == i);
            onFalseThrow(pq.removeMin().getValue() == i);
        }
        System.out.println("Success!");
    }
}
```

SUNY Korea

Priority Queue with a Sorted List

- Running time comparison

Method	Unsorted List	Sorted List
size	$O(1)$	$O(1)$
isEmpty	$O(1)$	$O(1)$
insert	$O(1)$	$O(n)$
min	$O(n)$	$O(1)$
removeMin	$O(n)$	$O(1)$

SUNY Korea

Assignment 5

- We will implement a "Snake Bite" game in this assignment
  - Implement the code marked with TODO tags
  - Submit modified source files in a single zip file

- Due date: 4/21/2022

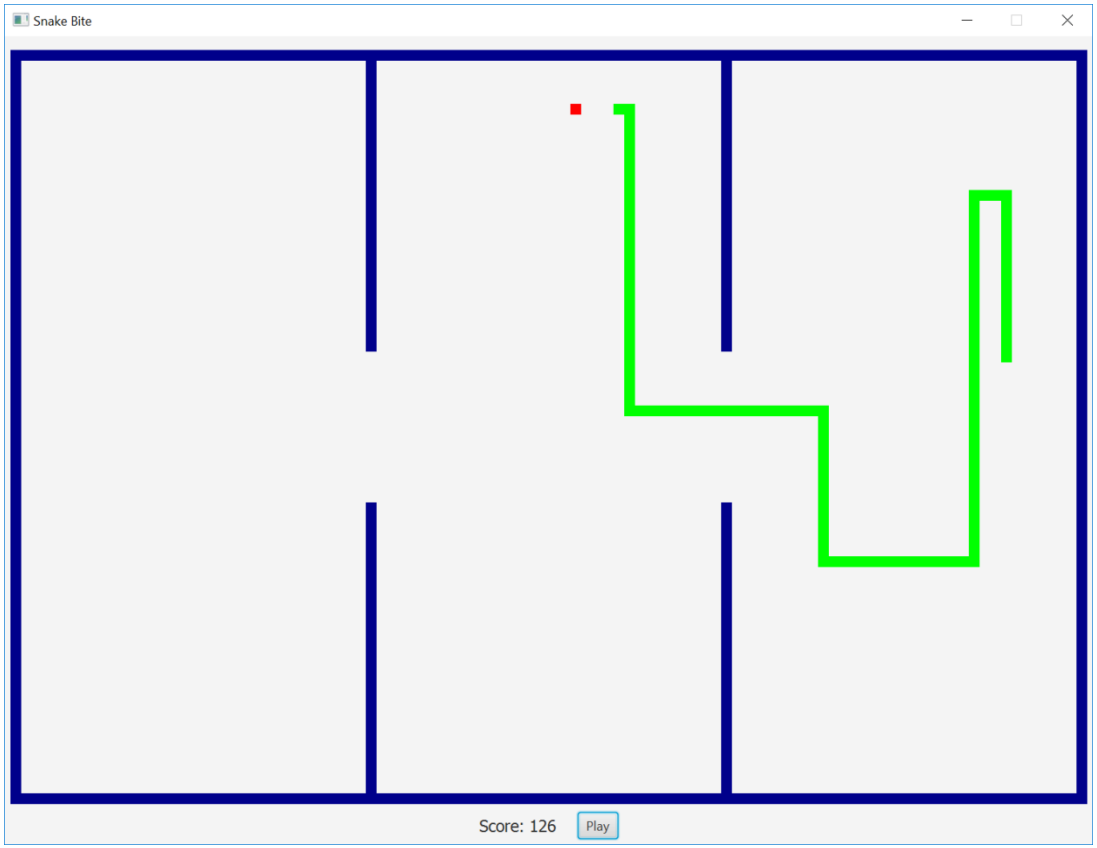
SUNY Korea

Assignment 5

- Implement `DynamicArrayDeque`
  - Finish implementing the TODOs
  - You should pass all unit tests in its `main`
- Implement `SnakeWorld`
  - Finish implementing the TODOs

SUNY Korea

- When all TODOs are implemented correctly, you can play the “Snake Bite” game



SUNY Korea