CS210
Discussion

Week 4

UMass Boston

Project 2 – Queues Galore

- Elementary data structures
- Generics
- Iterators

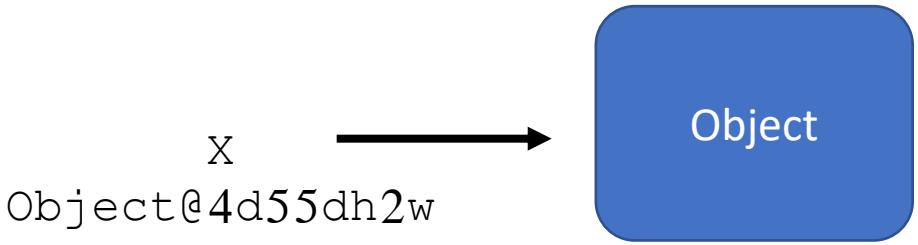
- Deque
  - **D**ouble ended **que**ue
- Randomized queue

UMass Boston

Pointers/references in Java

- Doesn't have explicit pointer data types like in C
- Object variables are essentially pointers
  - You can treat them like values/objects being passed around
  - But they are just references to data on the heap

```
Object x = new Object("Hello World");
```



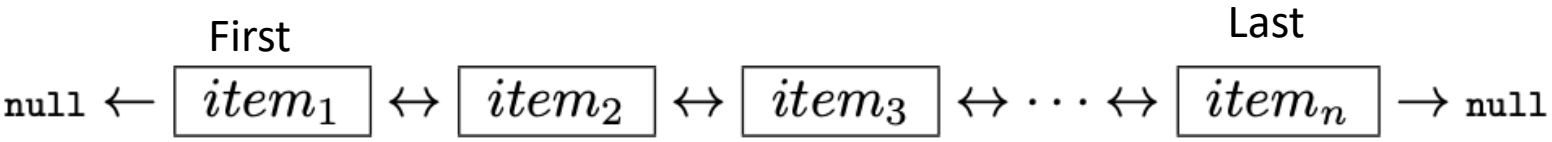
Generics

- Placeholder types
- Allows you to define classes/methods that operate on any type

```
LinkedQueue<String> some_queue = new LinkedQueue<String>();
```

```
public class LinkedDeque<Item> implements Iterable<Item> {
```

Doubly Linked List




```
// A data type to represent a doubly-linked list. Each node in the list stores a generic item
// and references to the next and previous nodes in the list.
2 usages
private class Node {
    private Item item;  // the item
    private Node next;  // the next node
    private Node prev;  // the previous node
}
```

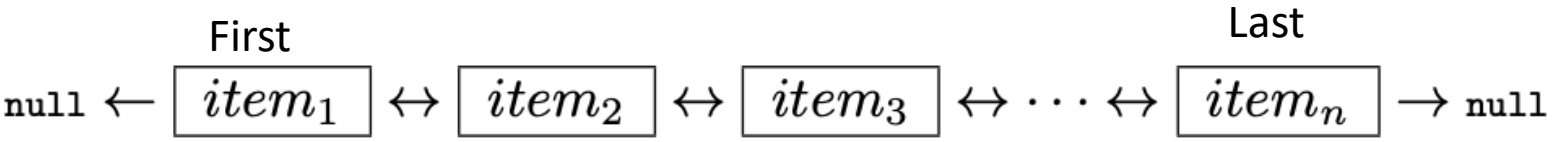
← "Pointers"

Which item does 'next' point to?

What about 'prev'?

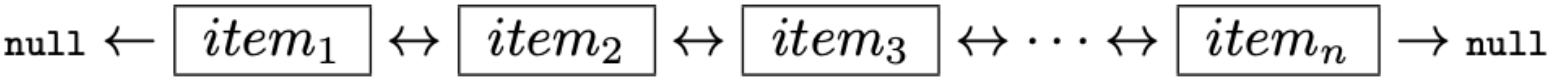
UMass Boston

Doubly Linked List




```
// Create item 1
Node item1 = new Node();

// Create item 2
Node item2 = new Node();
item1.next = item2;
item2.prev = item1;

// Create item 3
Node item3 = new Node();
item2.next = item3;
item3.prev = item2;
```

Deque - Double ended queue

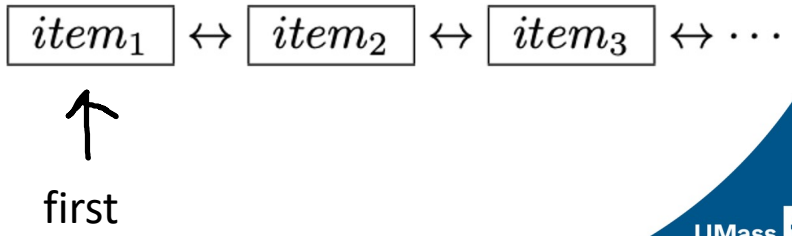


- Uses a doubly linked list as the underlying data structure
- Like a normal queue but with more methods

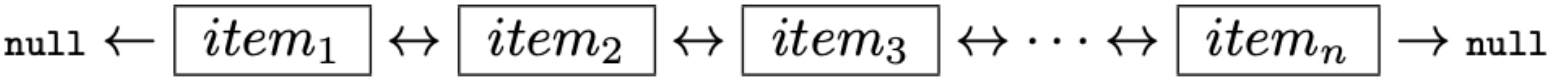
```
void addFirst(Item item)
void addLast(Item item)
```

The `add*()` methods

- Peek vs remove
  - Get the item
  - Get and remove the item



Deque - Double ended queue



`Iterator<Item> iterator()` returns an iterator to iterate over the items in this deque from front to back

- Return an object that 'implements' the Iterator interface
  - This isn't the same as the objects type
  - But we can treat it similarly

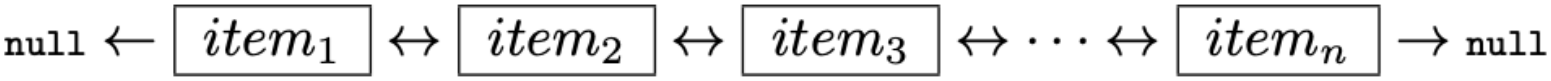
- An interface
  - Defines specific functions a class MUST implement
  - Can be used in place of the class name itself

Deque - Double ended queue

`Iterator<Item> iterator()`	returns an iterator to iterate over the items in this deque from front to back

```
// A deque iterator.
private class DequeIterator implements Iterator<Item> {
    ...

    // Constructs an iterator.
    public DequeIterator() {
        ...
    }

    // Returns true if there are more items to iterate, and false otherwise.
    public boolean hasNext() {
        ...
    }

    // Returns the next item.
    public Item next() {
        ...
    }
}
```

Deque - Double ended queue



- Performance T(n) ~ 1
  - For ALL constructors and ALL methods
  - Input size shouldn't affect runtime
  - Size of the deque itself shouldn't affect runtime

P2

Sorting Strings

- Uses the LinkedDeque data structure you built in problem 1
  - Make sure you compile LinkedDeque
- Take strings from StdIn and sort them into the deque
- Print the sorted strings out

UMass Boston

Questions about the first two problems?

Binary Strings

- Binary strings
  - Strings representing a binary number
  - For example, "`01010`"
- Iterate over binary strings w/ specified # of digits
- Iterators!

```
>_ ~/workspace/project2

$ java BinaryStrings 3
000
001
010
011
100
101
110
111
```

E1

Binary Strings

- Simple constructor
  - Set n
  - What's the syntax?
- "iterator" returns the iterator inner class

```
public class BinaryStrings implements Iterable<String> {
    1 usage
    private int n; // need all binary strings of length n

    // Constructs a BinaryStrings object given the length
    1 usage
    public BinaryStrings(int n) {
        ...
    }

    // Returns an iterator to iterate over binary strings
    public Iterator<String> iterator() {
        ...
    }
```

E1

Binary Strings

- Iterator inner class
  - Iterator interface
    - hasNext
    - next
  - Has access to outer class instance variables
    - n
  - Method to turn a number into a binary string is done for us.

```
// Binary strings iterator.
private class BinaryStringsIterator implements Iterator<String> {
    private int count; // number of binary strings returned so far
    private int p;     // current number in decimal

    // Constructs an iterator.
    public BinaryStringsIterator() {
        ...
    }

    // Returns true if there are anymore binary strings to be itera
    public boolean hasNext() {
        ...
    }

    // Returns the next binary string.
    public String next() {
        ...
    }
```

E1

Binary Strings

- Simple constructor
  - Set count and p to 0
- What about hasNext?
  - True if there are more binary strings
  - False otherwise
  - For a binary string with ”n” digits, what’s the largest integer our binary string can represent?

```
// Binary strings iterator.
private class BinaryStringsIterator implements Iterator<String> {
    private int count; // number of binary strings returned so far
    private int p;     // current number in decimal

    // Constructs an iterator.
    public BinaryStringsIterator() {
        ...
    }

    // Returns true if there are anymore binary strings to be itera
    public boolean hasNext() {
        ...
    }

    // Returns the next binary string.
    public String next() {
        ...
    }
```

3 digits: “000” = 0, “111” = 7

UMass Boston

E1

Binary Strings

- next
  - Gives the binary string representation of p
  - Use the given “binary” method
  - Increment p and count by 1
    - Readying the next binary string

```
// Binary strings iterator.
private class BinaryStringsIterator implements Iterator<String> {
    private int count; // number of binary strings returned so far
    private int p;     // current number in decimal

    // Constructs an iterator.
    public BinaryStringsIterator() {
        ...
    }

    // Returns true if there are anymore binary strings to be itera
    public boolean hasNext() {
        ...
    }

    // Returns the next binary string.
    public String next() {
        ...
    }
```

UMass Boston

Primes

- Like binary strings but you're iterating over primes
  - First n primes
- In this case, you iterate until you've found 10 primes →
- isPrime is done for us
- Iterating Integers instead of Strings

```
>_ ~/workspace/project2
$ java Primes 10
2
3
5
7
11
13
17
19
23
29
```

E2

Primes

- Debugging my version of primes
- Each method has 1 bug
- The constructor also has 1 bug
- Think about how we did Binary Strings
  - And how this program might be different

```
public class Primes implements Iterable<Integer> {
    1 usage
    private int n; // need first n primes

    // Constructs a Primes object given the number of primes needed.
    1 usage
    public Primes(int n) {
        n = n;
    }

    // Returns an iterator to iterate over the first n primes.
    public Iterator<Integer> iterator() {
        return PrimesIterator();
    }

    // Primes iterator.
    private class PrimesIterator implements Iterator<Integer> {
        2 usages
        private int count; // number of primes returned so far
        5 usages
        private int p;     // current prime

        // Constructs an iterator.
        public PrimesIterator() {
            count = 0;
            p = 0;
        }

        // Returns true if there are anymore primes to be iterated, and
        public boolean hasNext() {
            return p < n;
        }

        // Returns the next prime.
        public Integer next() {
            // Increment count by 1.
            count++;

            // As long as p is not prime, increment p by 1.
            while (isPrime(p)) {
                p++;
            }

            // Return current value of p and increment it by 1.
            return p++;
        }
```

MinMax

```
>_ ~/workspace/project2

$ java MinMax
min(first) == StdStats.min(items)? true
max(first) == StdStats.max(items)? true
```

- Linked lists! Traversal!
  - Singly linked list
- Go through the list and find the smallest and largest items

MinMax

```
protected static class Node {
    1 usage
    protected int item;  // the item
    1 usage
    protected Node next; // the next node
}
```

- These are the nodes in our linked list
  - Items here are just integers
- We have a reference to the first one in the list
  - The arguments to the functions in MinMax
- Then we use the “next” reference to get to the next node in the list

MinMax

```
// Set min to the largest integer.
```

```
// Set max to the smallest integer.
```

- This is a bit confusing
  - Here it doesn't mean the largest/smallest item in the list, which is what the overall goal is
  - It's saying, start our comparison with the largest/smallest integer possible in Java
  - `Integer.MAX_VALUE`
  - `Integer.MIN_VALUE`

MinMax

- Go item by item and keep tabs on what the smallest integer you've seen is
  - Using the `next` variable of each `Node`
  - This will be an odd `For` loop
- After all the items in the linked list, return the smallest integer we saw

```
public static int min(Node first) {
    // Set min to the largest integer.
    ....

    // Compare each element in linked list w
    ....

    // Return min.
    ....
}
```

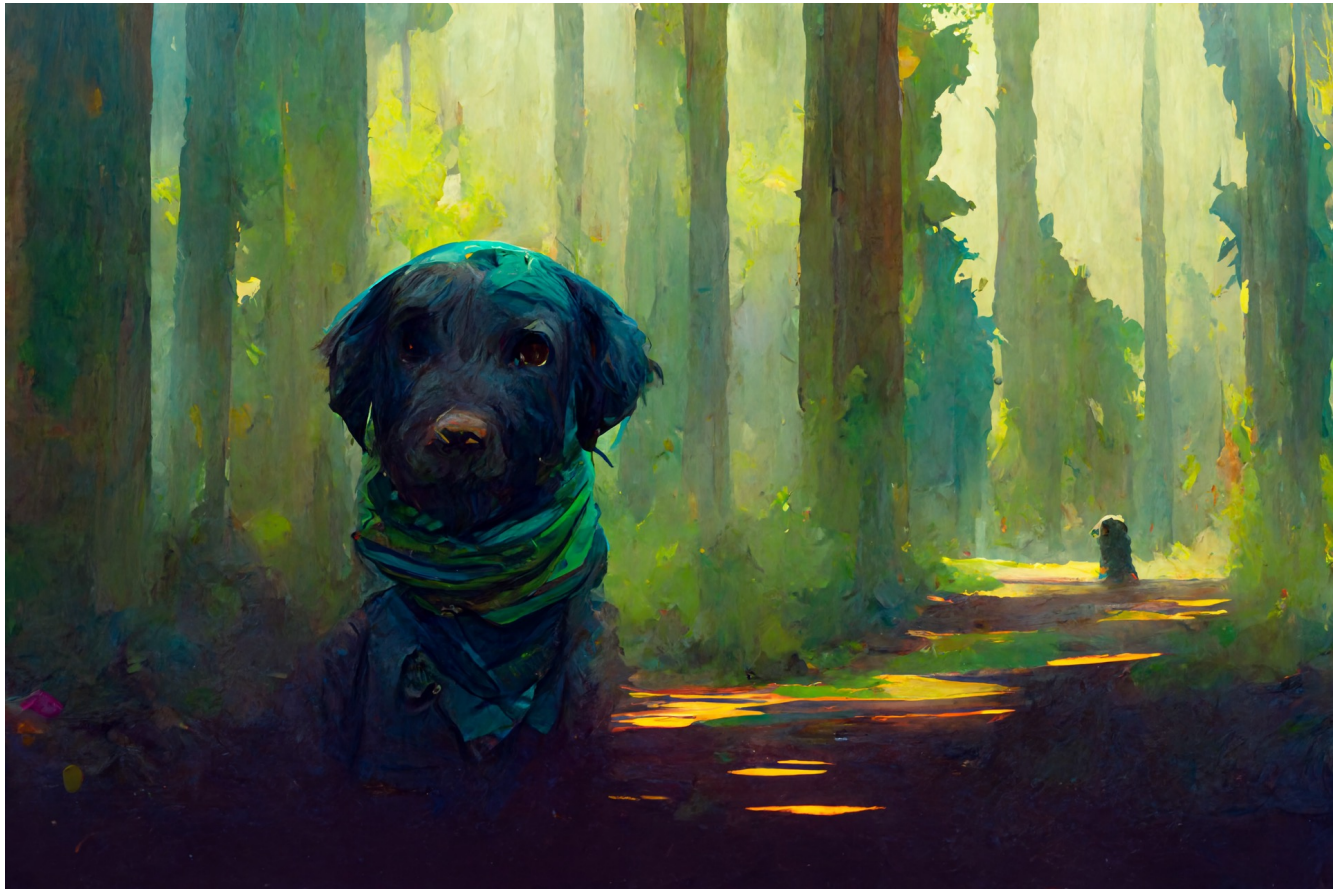
E3

MinMax

- Same as ”min” but keep tabs on the largest integer you’ve seen so far
- So where in the code will the two functions be different?
  - Two places
- Take a few minutes and write it out “max”
  - Talk to your neighbor

```
public static int max(Node first) {
    // Set max to the smallest intege
    ...

    // Compare each element in linked
    ...

    // Return max.
    ...
}
```

Questions?



UMass Boston

E4

Buffer

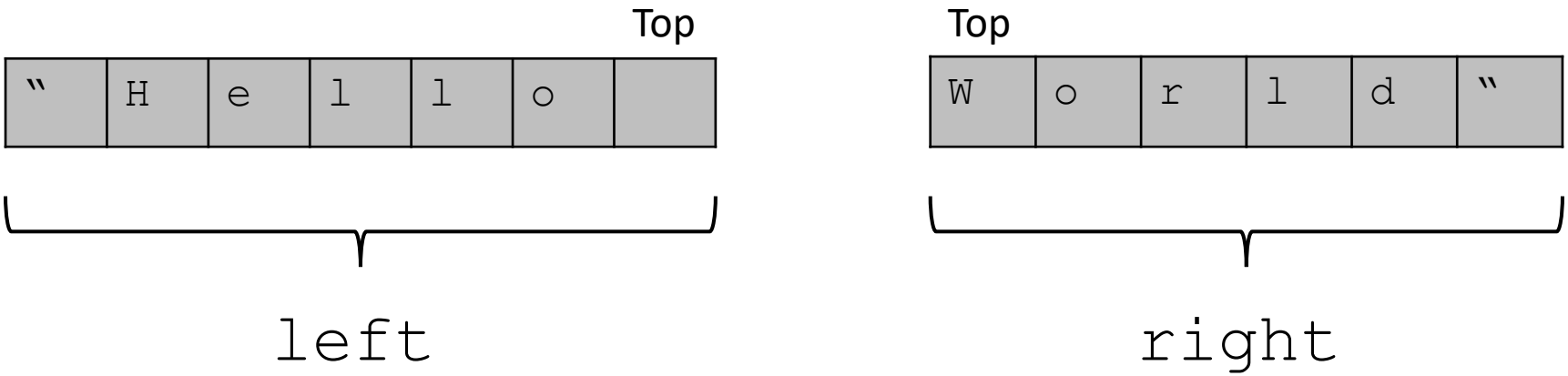
```
>_ ~/workspace/project2

$ java Buffer
|There is grandeur in this view of life, with its several powers,having been originally breathed by the
Creator into a few forms or into one; and that, whilst this planet has gone cycling on according to the
fixed law of gravity, from so simple a beginning endless forms most beautiful and most wonderful have
been, and are being, evolved. -- Charles Darwin, The Origin of Species
```

- The idea here is we're building a text editor
  - Buffer – data structure
  - "cursor"
  - Methods to move cursor and insert/remove characters
- Stacks are the underlying data structure
  - Two stacks!

UMass Boston

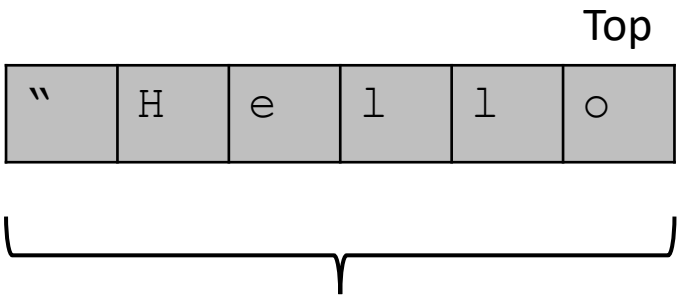
Buffer

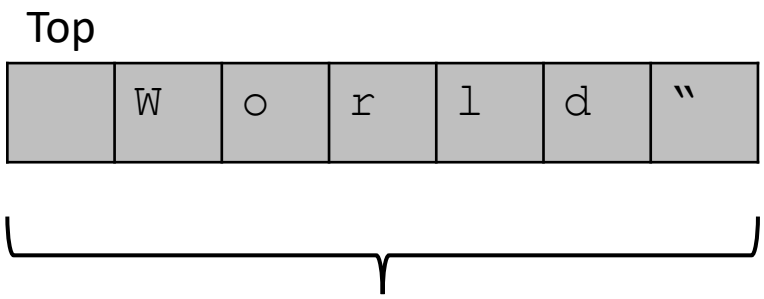
```
protected LinkedStack<Character> left;  // chars left of cursor
protected LinkedStack<Character> right; // chars right of cursor
```



Buffer

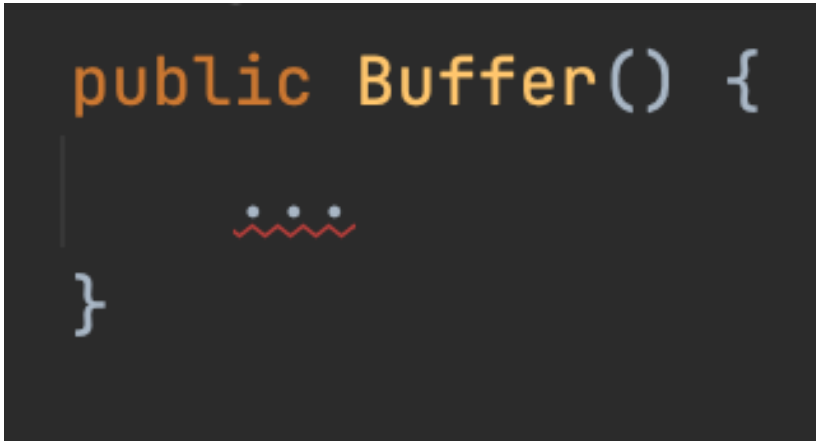
```
protected LinkedStack<Character> left;  // chars left of cursor
protected LinkedStack<Character> right; // chars right of cursor
```





Buffer

- Simple constructor
  - Set instance variables
- Left & right stacks
  - Buffer is empty to start, so nothing in the stacks
  - How do we create a new empty stack? What would we type for left?



E4

Buffer

- When we type, the text shows up to the left of the cursor
  - Same for “insert”
  - Left stack
- What method do you use to add something on top of a stack?
  - What would we type to add ”c” to the top of the left stack?

```
public void insert(char c) {
    ....
}
```

dsa.Stack<Item> extends java

```
boolean isEmpty()
int size()
void push(Item item)
Item peek()
Item pop()
Iterator<Item> iterator()
```

UMass Boston

Buffer

- Delete, not backspace
  - Delete the character to the right of the cursor
  - Right stack
- What method do we need to remove the item on top of a stack?
  - What's the difference between peek and pop?
  - What would we type?

```
public char delete() {
    ....
}
```

```
dsa.Stack<Item> extends java

boolean isEmpty()
int size()
void push(Item item)
Item peek()
Item pop()
Iterator<Item> iterator()
```

Buffer

- How is the cursor represented?
  - Implicit
  - Two stacks

```
public void left(int k) {
    ....
}
```

dsa.Stack<Item> extends java

```
boolean isEmpty()
int size()
void push(Item item)
Item peek()
Item pop()
Iterator<Item> iterator()
```

Buffer

- How is the cursor represented?
  - Implicit
  - Two stacks

- Moving k characters from one stack to the other
  - `For` loop
  - What stack methods do we use?

```
public void left(int k) {
    ....
}
```

dsa.Stack<Item> extends java

```
boolean isEmpty()
int size()
void push(Item item)
Item peek()
Item pop()
Iterator<Item> iterator()
```

Buffer

- Same as left but with one difference

- What is it?

```
public void right(int k) {
    ....
}
```

dsa.Stack<Item> extends java

```
boolean isEmpty()
int size()
void push(Item item)
Item peek()
Item pop()
Iterator<Item> iterator()
```

Buffer

- The total number of characters in our buffer
  - Left stack
  - Right stack
- What stack method do we need?
  - What do we type to get the total buffer size?
  - Oneliner

```
public int size() {
    ....
}
```

dsa.Stack<Item> extends java

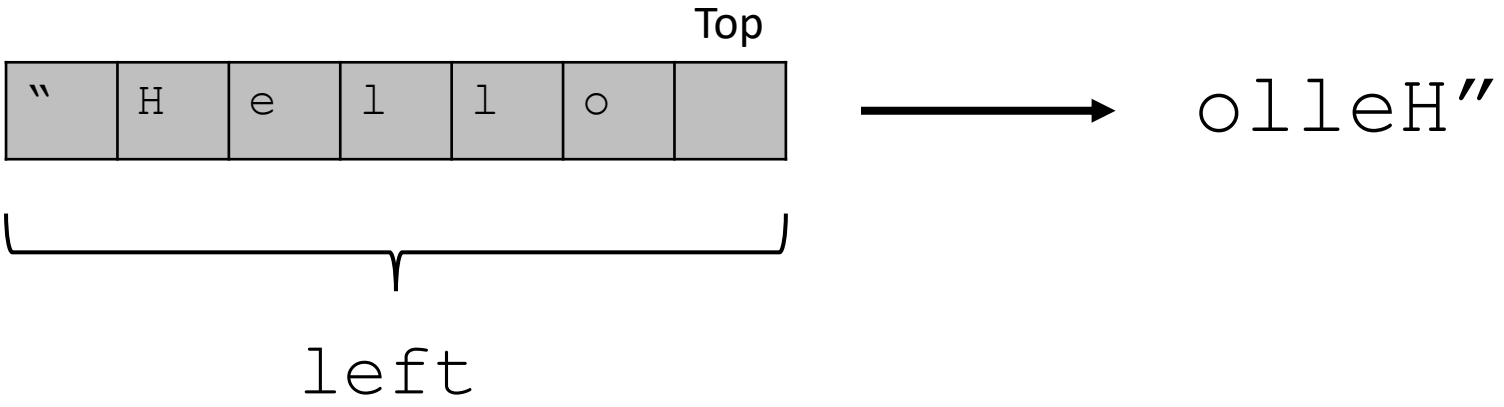
```
boolean isEmpty()
int size()
void push(Item item)
Item peek()
Item pop()
Iterator<Item> iterator()
```

Buffer

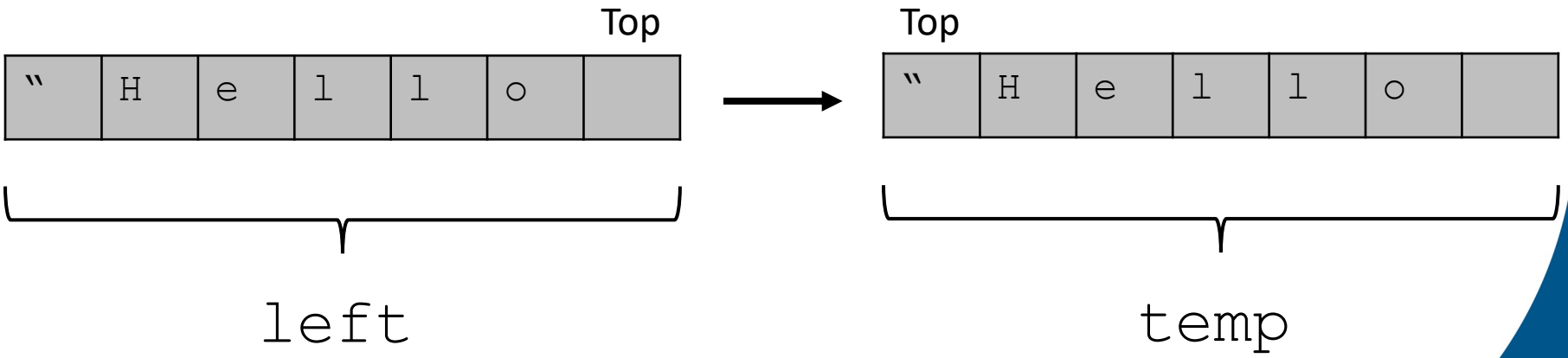
- The full text of the buffer with the cursor shown as the “|” character
- Step by step
  - Left
  - Cursor
  - Right
- Why do we use a temporary stack for left?

```
public String toString() {
    // A buffer to store the string representation.
    StringBuilder sb = new StringBuilder();

    // Push chars from left into a temporary stack.
    ...

    // Append chars from temporary stack to sb.
    ...

    // Append "|" to sb.
    ...

    // Append chars from right to sb.
    ...

    // Return the string from sb.
    ...
}
```

Buffer



Buffer



Buffer

Relevant StringBuilder methods

```
StringBuilder    append(char c)
```

```
String           toString()
```

- Stacks are iterable so we can use some nice syntactic sugar

```
public String toString() {
    // A buffer to store the string representation.
    StringBuilder sb = new StringBuilder();

    // Push chars from left into a temporary stack.
    ...

    // Append chars from temporary stack to sb.
    ...

    // Append "|" to sb.
    ...

    // Append chars from right to sb.
    ...

    // Return the string from sb.
    ...
}
```

Questions?

UMass Boston

Random Queue

- Uses a resizing array as the underlying data structure
  - You have to manage the array
  - ”resize” method done for you
- Difference from queue
  - “sample”
  - Random iterator

Random Queue

- Each iterator should be independent of the original random queue
- Maintains it’s own copy of the items in the random queue

```
// An iterator, doesn't implement remove() since it's optional.
private class RandomQueueIterator implements Iterator<Item> {
    ...

    // Constructs an iterator.
    public RandomQueueIterator() {
        ...
    }

    // Returns true if there are more items to iterate, and fal
    public boolean hasNext() {
        ...
    }

    // Returns the next item.
    public Item next() {
        ...
    }
}
```

Random Queue

- All methods in the random queue should run in constant time
- All methods in the iterator should run in constant time
  - Except the constructor, which should run in linear time