CS210 Discussion

Week 5

UMass Boston

Project 2 – Queues Galore

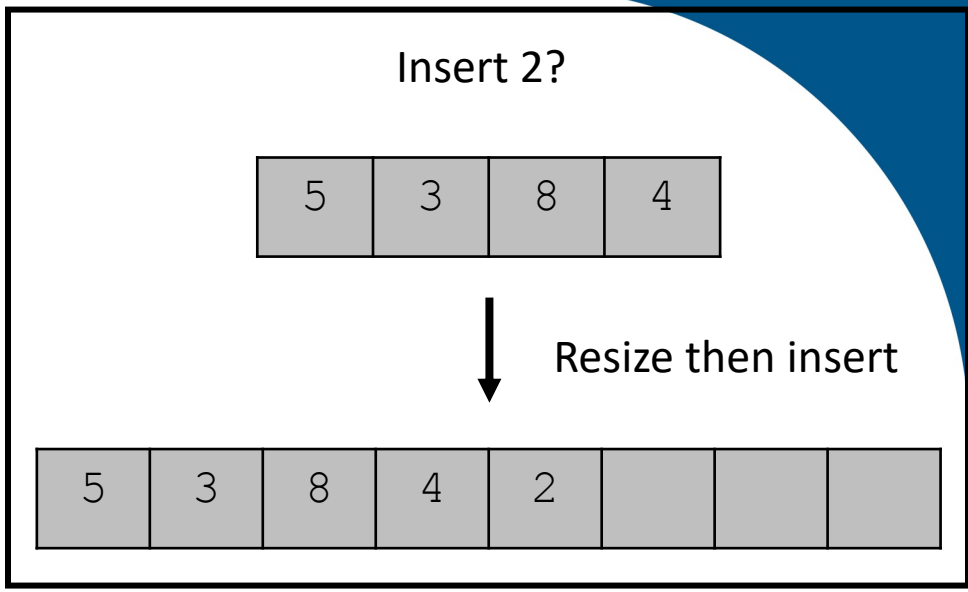
- Elementary data structures
- Generics
- Iterators

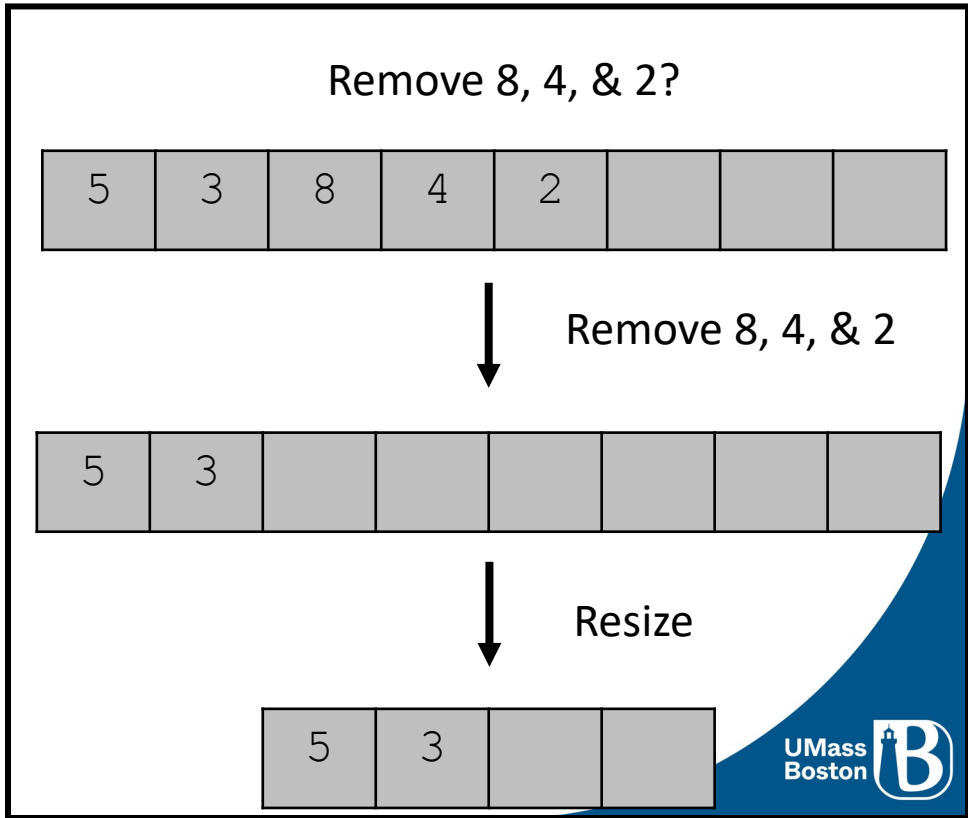
- Deque
  - **D**ouble ended **que**ue
- Randomized queue

UMass Boston

Resizing Arrays

- Another underlying data structure
  - Can be used in place of a linked list
- An array who's size can be changed as needed
  - Double when full
  - Half when ¼ full
- Make new array and copy elements over





Randomized Queue

- Like a normal queue but you get random elements out.
  - Similar methods
- Iterator again
  - Random items
- Use a resizing array instead of a linked list
  - Array starts with a size of 2
  - You manage the array
    - “resize” done for you

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

Sampling Integers

- Utilizes the Random Queue
- Gives random numbers from a range (inclusive)
  - [lo, hi] (range)
  - K (number of numbers)
- Mode (+/-)
  - With or without replacement
- T(k, n) ~ kn

Questions about the last two problems?

Reports

- Make sure to submit your report.txt file
- A sentence or two is probably not enough
- Edit what you write, make sure it’s clear & coherent
- Remove commented out code blocks

Approach

- Thoughtful consideration of the problem
- Don’t simply summarize the project instructions.
- Answer the Qs:
  - What were you thinking about when confronted with this problem
  - *Why* did you do things the way you did?

After reading through the project instructions, I realized that multiple methods relied on the isDog() method. I chose to build this method first so I would have everything I needed when working on the others. This way I could fully test those methods as I completed them, as opposed to testing them all at once after I finished isDog().

I also realized that many of the methods that call isDog() call bark() immediately afterwards. So, I wrote a helper function called barkIfDog() which makes an object bark if it’s a dog. This allowed me to reduce code duplication and centralize the bark logic to a single point in the codebase.

Issues and Resolution

- Be specific and detailed
- Talk about a specific issue/error:
  - What was it?
  - What caused it?
  - What did you have to change in the code to fix it?
- If you can't resolve the issue:
  - Explain what you tried
  - Speculate as to what may be wrong or what the solution may look like

While working on the rollOver() method I kept getting DogNotLyingDown errors. At first, I was surprised to see this issue because I thought I'd been meticulous about the trick performing logic. What I missed ended up being a simple conditional mistake. In the rollOver() method, I had an 'if' statement that checked to make sure the dog was lying down before attempting to roll over. I'd written "if (dog.isLayingDown())" when I needed to write "if (!dog.isLayingDown())". After adding the '!' the program ran as expected and passed all tests.

Report Questions?

E4

Buffer

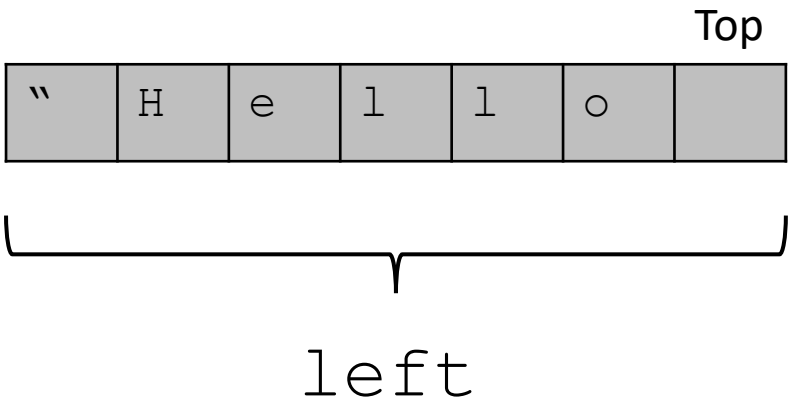
```
>_ ~/workspace/project2
$ java Buffer
|There is grandeur in this view of life, with its several powers,having been originally breathed by the
Creator into a few forms or into one; and that, whilst this planet has gone cycling on according to the
fixed law of gravity, from so simple a beginning endless forms most beautiful and most wonderful have
been, and are being, evolved. -- Charles Darwin, The Origin of Species
```

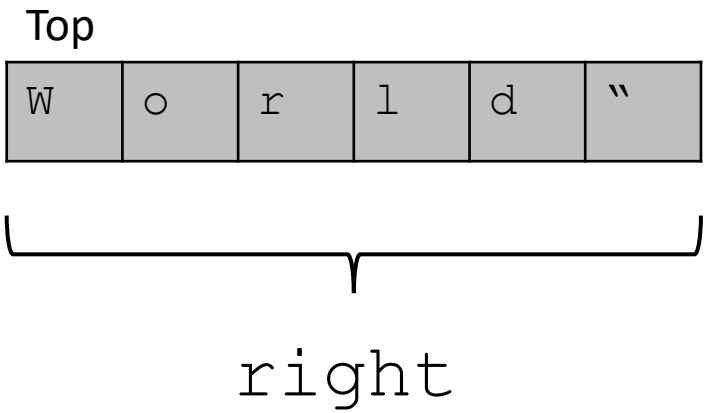
- The idea here is we're building a text editor
  - Buffer – data structure
  - "cursor"
  - Methods to move cursor and insert/remove characters
- Stacks are the underlying data structure
  - Two stacks!

E4

Buffer

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

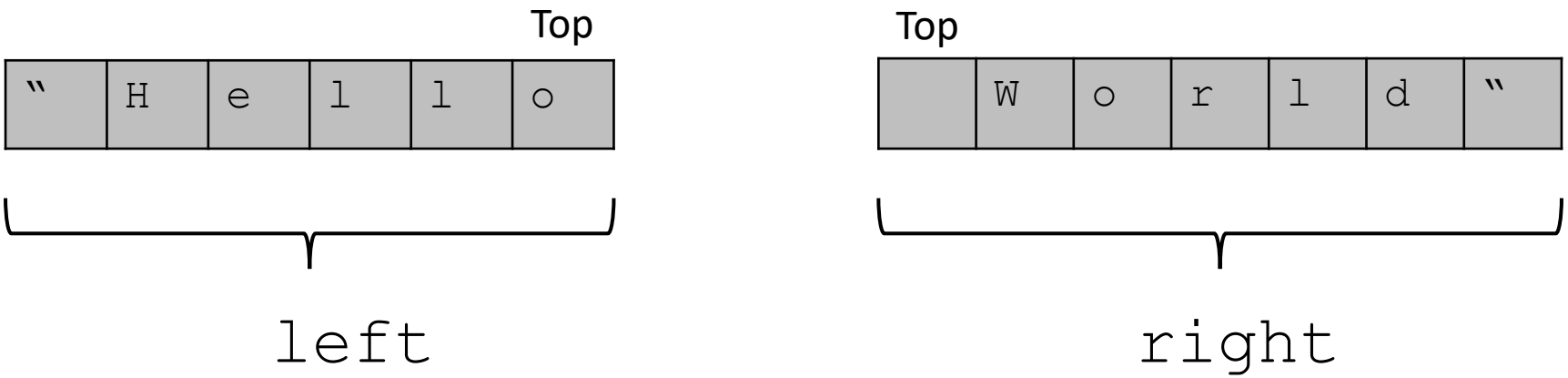




E4

Buffer

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



E4

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?

```
public Buffer() {

    ...

}
```

UMass Boston

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()

E4

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()
```

UMass Boston

E4

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()
```

UMass Boston

E4

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()
```

UMass Boston

E4

Buffer

- Work on the rest of the method's individually or in groups for ~ 10 minutes
- StringBuilder
  - sb.append() to add to the string you're building
  - sb.toString() to get the string you've built.

```
dsa.Stack<Item> extends java

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

UMass Boston

E4

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()
```

UMass Boston

E4

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()
```

UMass Boston

E4

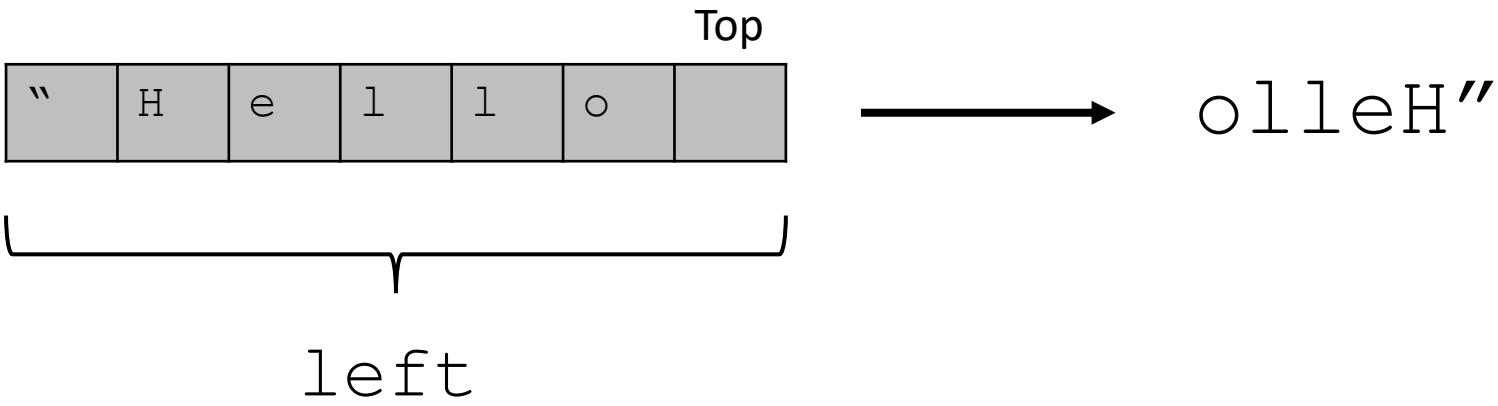
Buffer

- 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?

```java
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.
    ...
}
```

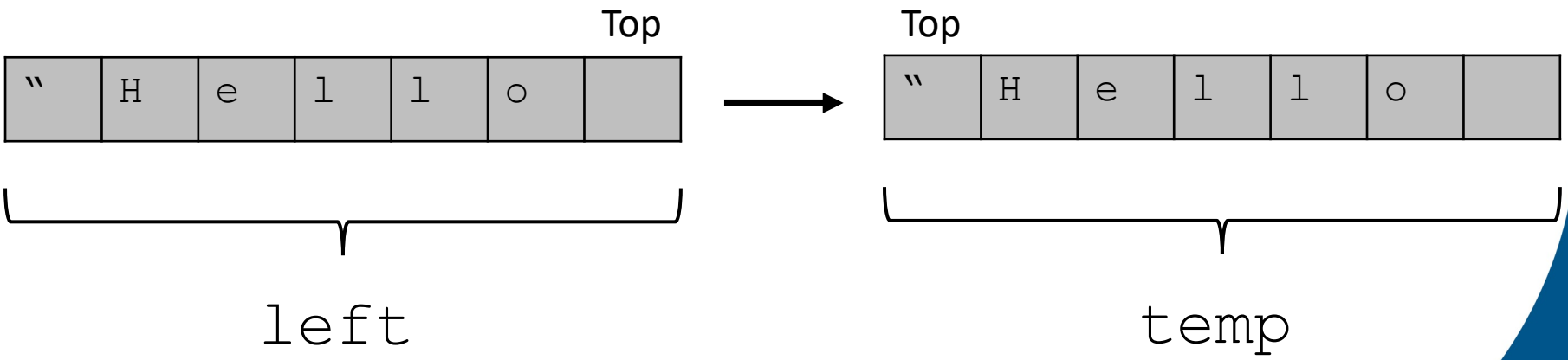
E4

Buffer



E4

Buffer



E4

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.
    ...
}
```

E5

Josephus

- Using a queue
- Go through a circle of elements and keep removing the ’mth’ element
- Elements are numbers 1 to n
- Work individually or in groups for 10 minutes

```
dsa.Queue<Item> extends java

boolean isEmpty()
int size()
void enqueue(Item item)
Item peek()
Item dequeue()
Iterator<Item> iterator()
```

UMass Boston

E5

Josephus

```
public class Josephus {
    // Entry point.
    public static void main(String[] args) {
        // Accept n (int) and m (int) as command-line arguments.
        ...

        // Create a queue q and enqueue integers 1, 2, ..., n.
        ...

        // Set i to 0. As long as q is not empty: increment i; dequeue an element (say pos); if m
        // divides i, write pos to standard output, otherwise enqueue pos to q.
        ...
    }
}
```

UMass Boston

Questions?