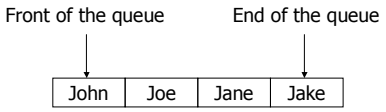
CS203 Programming with Data Structures

Queues and Stacks

Chengyu Sun
California State University, Los Angeles

Queue



- Queue properties
  - Always insert at the end of the queue (`enqueue`)
  - Always remove from the front of the queue (`dequeue`)
  - First-In-First-Out (FIFO)

The `Queue` Interface

- `void enqueue( Object o )`
- `Object dequeue()`
  - NoSuchElementException
  - Return and remove
- `Object front()`
  - Return but not remove
- `boolean isEmpty()`
- `void clear()`

Queue Example

```
// assume queue is empty

queue.enqueue( "John" );
queue.enqueue( "Joe" );

System.out.println( queue.front() );  // ??
System.out.println( queue.front() );  // ??
System.out.println( queue.front() );  // ??

System.out.println( queue.dequeue() );  // ??
System.out.println( queue.dequeue() );  // ??
System.out.println( queue.dequeue() );  // ??
```

Create an Exception Class

- Inherits from `Exception`
- Inherits from `RuntimeException`

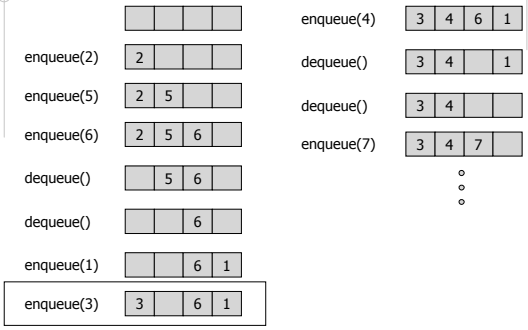
Queue Implementations

- Using LinkedList
- Using ArrayList
- What are the complexity of `enqueue()` and `dequeue()` in each case??

Queue Implementation Using Circular Array

- For simplicity, assume there will be *at most* N objects in queue at any time
- `enqueue()` and `dequeue()` do not require shifting all the elements in the array

Circular Array



Circular Array Implementation

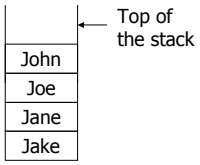
```
public class CircularArrayQueue implements Queue {

    Object    elements[];
    int       front, end;
    int       size;

    …
}
```

Stack

- Stack properties
  - Insertions (`push`) and removals (`pop`) happen only at the top of the stack
  - First-In-Last-Out (FILO)



The `Stack` Interface

- `void push( Object o )`
- `Object pop()`
  - NoSuchElementException
  - Return and remove
- `Object top()`
  - Return but not remove
- `boolean isEmpty()`
- `void clear()`

Stack Example

```
// assume stack is empty

stack.push( "John" );
stack.push( "Joe" );

System.out.println( stack.top() );  // ??
System.out.println( stack.top() );  // ??
System.out.println( stack.top() );  // ??

System.out.println( stack.pop() );  // ??
System.out.println( stack.pop() );  // ??
System.out.println( stack.pop() );  // ??
```

Stack Implementations

- Using list
- Using array

Stack Application – Balancing Symbols

- Given a program (as a text file or a string), check whether `()`, `[]`, and `{}` in the program are balanced

Stack Application – Evaluating Postfix Expressions