University of Iowa Computing Conference 2017

February 24-25, 2017

Register by Feb. 14: acm.org.uiowa.edu/uicc

Awesome Speakers:
James Hendler, Lee Brinle, Alexander Stoytchev, Kyle Rector, Thomas Hornbeck

No-Tie Job Fair
Puzzle Competition
Laptop Orchestra Concert
Free Food and Swag!
Midterm 1

• Next Friday 2/17 in class, 50 minutes
• What is on it?
  • Anything you’ve practiced including HW3 (incl. quizzes, peer instructions, HWs, discussion work)
• Is there any practice/review?
  • We’ll put up at least 1 prior exam/solutions
  • Discussion on 2/13-2/15 will be used as a review (come ready to work on practice exam questions)
• Notes allowed?
  • 1 double-sided sheet of notes
CS 2230
CS II: Data structures

Meeting 9: List ADT, Brandon Myers
University of Iowa
Today’s big ideas

• Interfaces can be used to express *Abstract Data Types*, like List

• *Generics* allow us to build data structures that can hold elements of any type

• Learn about two more Abstract Data Types, Stack and Queue
public interface List {
    // add the element to the end of the list
    public void append(Object element);

    // check if the element exists in the list
    public boolean contains(Object element);

    // remove first element from the list and return it
    public Object removeFirst();

    // return true if the list is empty
    public boolean isEmpty();
}

class ArrayList implements List {
    private Object[] elements;
    private int numElements;
    @Override
    public void append(Object element) {
    }
    @Override
    public boolean contains(Object element) {
        }
    @Override
    public Object removeFirst() {
        }
    @Override
    public boolean isEmpty() {
        }
}

class LinkedList implements List {
    private ListNode header;
    @Override
    public void append(Object element) {
    }
    @Override
    public boolean contains(Object element) {
        }
    @Override
    public Object removeFirst() {
        }
    @Override
    public boolean isEmpty() {
        }
}
In **ArrayList**’s implementation of `List.append`, what should we do in this case?

- a) allocate a bigger array and copy all the elements to it
- b) return an error
- c) increase the length of the array to make room
- d) replace the last element with 900
- e) don’t change anything

```
myList.append(900)
```

https://b.socrative.com/login/student/
CS2230A ids 1000-4999
CS2230B ids 5000+
make a group of 3 students:
• manager (ensure everyone contributes)
• scribe (sits in the middle, doesn’t decide what to write)
• skeptic (question everything)

class ArrayList implements List {
    private Object[] elements;
    private int numElements;

    @Override
    public void append(Object ele) {
        // finish this method
    }
}

for your information, here is the relevant part of List

public interface List {
    // add the element to the end of the list
    public void append(Object ele);
}
Generics allow us to build data structures that can hold elements of any type
/*
Interface for a generic List ADT, an ordered collection of elements

GList<T> can store objects of some type T;
for example, a GList<String> can store Strings
and a GList<Cat> can store Cats
*/

public interface GList<T> {
    // add the element to the end of the list
    public void append(T element);

    // check if the element exists in the list
    public boolean contains(T element);

    // remove first element from the list and return it
    public T removeFirst();

    // return true if the list is empty
    public boolean isEmpty();
}
public interface Glist<T> {
    // add the element to the end of the list
    public void append(T element);

    // check if the element exists in the list
    public boolean contains(T element);

    // remove first element from the list and return it
    public T removeFirst();

    // return true if the list is empty
    public boolean isEmpty();
}

class LinkedList<T> implements Glist<T> {
    private ListNode header;

    @Override
    public void append(T element) {
        // implementation
    }

    @Override
    public boolean contains(T element) {
        // implementation
    }

    @Override
    public T removeFirst() {
        // implementation
    }

    @Override
    public boolean isEmpty() {
        // implementation
    }
}

unlimited number of LinkedList classes for the price of 1!!

LinkedList<Cat>

LinkedList<Integer>

LinkedList<String>
public interface GList<T> {
    // add the element to the end of the list
    public void append(T element);

    // check if the element exists in the list
    public boolean contains(T element);

    // remove first element from the list and return it
    public T removeFirst();

    // return true if the list is empty
    public boolean isEmpty();
}

class ArrayList<T> implements GList<T> {
    private T[] elements;
    private int numElements;
    @Override
    public void append(T element) {
    }

    @Override
    public boolean contains(T element) {
    }

    @Override
    public T removeFirst() {
    }

    @Override
    public boolean isEmpty() {
    }
}

class LinkedList<T> implements GList<T> {
    private ListNode header;
    @Override
    public void append(T element) {
    }

    @Override
    public boolean contains(T element) {
    }

    @Override
    public T removeFirst() {
    }

    @Override
    public boolean isEmpty() {
    }
}
Learn about two more Abstract Data Types, Stack and Queue
Queue ADT

public interface Queue<T> {
    /* Insert element at back of queue */
    public void enqueue(T ele);

    /* Remove element from front of queue and return it */
    public T dequeue();

    /* Return the element at the front of queue */
    public T peek();
}
/*
Interface for a generic List ADT, an ordered collection of elements

GList<T> can store objects of some type T;
for example, a GList<String> can store Strings
and a GList<Cat> can store Cats
*/

public interface GList<T> {
    // add the element to the end of the list
    public void append(T element);

    // check if the element exists in the list
    public boolean contains(T element);

    // remove first element from the list and return it
    public T removeFirst();

    // return true if the list is empty
    public boolean isEmpty();
}

public interface Queue<T> {
    /* Insert element at back of queue */
    public void enqueue(T ele);

    /* Remove element from front of queue
    and return it */
    public T dequeue();

    /* Return the element at the front of queue */
    public T peek();
}
Stack ADT
Real-life example where...

• a process/situation/etc behaves like a stack
• or, a specific application where a stack data structure would be useful

https://b.socrative.com/login/student/
CS2230A ids 1000-4999
CS2230B ids 5000+
public interface Stack<T> {
    /* put the element on the top of the stack */
    public void push(T ele);

    /* remove the element on top of the stack
    and return it */
    public T pop();

    /* return the element on top of the stack */
    public T peek();
}
/*
Interface for a generic List ADT, an ordered collection of elements

GList<T> can store objects of some type T; for example, a GList<String> can store Strings and a GList<Cat> can store Cats
*/

public interface GList<T> {
    /* add the element to the end of the list */
    public void append(T element);

    /* check if the element exists in the list */
    public boolean contains(T element);

    /* remove first element from the list and return it */
    public T removeFirst();

    /* return true if the list is empty */
    public boolean isEmpty();
}

public interface Stack<T> {
    /* put the element on the top of the stack */
    public void push(T ele);

    /* remove the element on top of the stack and return it */
    public T pop();

    /* return the element on top of the stack */
    public T peek();
}

True/False

Stack<T> could be implemented using a GList<T>
Today’s big ideas

• Interfaces can be used to express *Abstract Data Types*, like List

• **Generics** allow us to build data structures that can hold elements of any type

• Learn about two more Abstract Data Types, Stack and Queue