CS367 Announcements
Tuesday, June 18, 2013

- Register on Piazza
- Email bgibson@cs.wisc.edu about conflicts, etc.

New to Linux?

- Linux orientation?
- See "New to Linux?" link on website

Last Time

- course info handout (extras at front)
- course intro
  - logistics
  - goal: create good software
  - focus: reusability (ease of understanding, general, modifiable) and efficiency
- abstract data type (ADT)
- Box ADT
- Java Interfaces
- using Object class for generality

Today

- Review ADT, Interfaces, Object class
- autoboxing/casting
- using Java generics for generality
- the ListADT

Recall the Box ADT

Conceptual Picture

Public Interface / operations

```
void add(Object item);
Object remove() throws NoSuchElementException;
boolean isEmpty();
```

Using the Box ADT: Example 2

Assume `Die` is a class representing dice and `box` is a `BoxADT`.
Why doesn't the following code compile?

```
for ( int i = 0; i < 10; i++ ) {
    box.add( new Die() );
}

while ( !box.isEmpty() ) {
    Die myDie = box.remove();
    myDie.roll();
}
```

Making the Box ADT General using Java Generics

```
import java.util.*;

public interface BoxADT {

    void add(Object item);

    Object remove() throws NoSuchElementException;

    boolean isEmpty();

}
```

Options for Implementing the Generic Box ADT

Option 1: a non-generic implementation

Option 2: a generic implementation

Using a Java Generic Implementation (assume option 2)

Write code to make a Box ADT storing `String`s and another storing `Die` objects

Now write code to make a Box ADT store `Object`s

The List ADT

Conceptual picture

Operations

- void add(E item)
- void add(int pos, E time)
- boolean contains(E item)
- int size()
- boolean isEmpty()
- E get(int pos)
- E remove(int pos)

List ADT as a Java Interface
(code and generated Javadoc will be posted on syllabus page)

```
/**
 * A List is an ordered collection of items.
 */
public interface ListADT<E> {

    /**
     * Add item to the end of the List.
     *
     * @param item the item to add
     */
    void add(E item);

    /**
     * Add item at position pos in the List, moving the items
     * originally in positions pos through size()- 1 one place
     * to the right to make room.
     *
     * @param pos the position at which to add the item
     * @param item the item to add
     * @throws IndexOutOfBoundsException if pos is less than 0
     * or greater than size()
     */
    void add(int pos, E item);

    /**
     * Return true iff item is in the List (i.e., there is an
     * item x in the List such that x.equals(item))
     *
     * @param item the item to check
     * @return true if item is in the List, false otherwise
     */
    boolean contains(E item);
```

List ADT as a Java Interface (cont.)

```
    /**
     * Return the number of items in the List.
     *
     * @return the number of items in the List
     */
    int size();

    /**
     * Return true iff the List is empty.
     *
     * @return true if the List is empty, false otherwise
     */
    boolean isEmpty();

    /**
     * Return the item at position pos in the List.
     *
     * @param pos the position of the item to return
     * @return the item at position pos
     * @throws IndexOutOfBoundsException if pos is less than 0
     * or greater than or equal to size()
     */
    E get(int pos);

    /**
     * Remove and return the item at position pos in the List,
     * moving the items originally in positions pos+1 through
     * size() one place to the left to fill in the gap.
     *
     * @param pos the position at which to remove the item
     * @return the item at position pos
     * @throws IndexOutOfBoundsException if pos is less than 0
     * or greater than or equal to size()
     */
    E remove(int pos);

}
```

Using the List ADT

Given a `ListADT` named `myList`, write code to reverse `myList` without using an additional `ListADT` or any other data structure (not even an array).

Implementing the List ADT using `SimpleArrayList` (option 1)

```
public class SimpleArrayList implements ListADT<Object> {

    private Object[] items; // the items in the List
    private int numItems;   // the # of items in the List

    public SimpleArrayList() {



    }

    //*** required ListADT methods ***

    public void add(Object item) { ... }

    public void add(int pos, Object item) { ... }

    public Object remove(int pos) { ... }

    public Object get (int pos) { ... }

    public boolean contains (Object item) { ... }

    public int size() { ... }

    public boolean isEmpty() { ... }

}
```

List ADT

Conceptual picture

Operations

```
public interface ListADT<E> {
    void add(E item);
    void add(int pos, E item);
    boolean contains(E item);
    int size();
    boolean isEmpty();
    E get(int pos);
    E remove(int pos);
}
```

Implementing ListADT using an array (option 1)

```
public class SimpleArrayList implements ListADT<Object>{

    private Object[] items; // items in the List
    private int numItems;   // # of items in the List

    public SimpleArrayList() { ... }

    //*** required ListADT methods ***

    public void add(Object item) { ... }
    public void add(int pos, Object item) { ... }
    public Object remove(int pos) { ... }
    public Object get (int pos) { ... }
    public boolean contains (Object item) { ... }
    public int size() { ... }
    public boolean isEmpty() { ... }

    // *** private methods ***

}
```

Using the List ADT

Given a `ListADT` named `myList`, write code to reverse `myList` without using an additional `ListADT` or any other data structure (not even an array).

One solution:

```
for ( int i = 1; i < myList.size(); i++ )
    myList.add(0,myList.remove(i));
```

Another solution:

```
for ( int i = 0; i < myList.size(); i++ )
    myList.add(myList.remove(myList.size() - 1 - i));
```

Yet another solution:

```
for ( int i = myList.size() - 1; i >= 0; i-- )
    myList.add(myList.remove(i));
```