CS 367 - Introduction to Data Structures
Thursday, January 21, 2016

We assume that you are proficient at object-oriented programming in Java.

Instructors

- **Lec 3 & 1:** Jim Skrentny, 5379 CS, skrentny@cs.wisc.edu
- **website:** http://pages.cs.wisc.edu/~cs367-1/
- **Lec 2:** Deb Deppeler, 5376 CS, deppeler@cs.wisc.edu
- **website:** http://pages.cs.wisc.edu/~cs367-2/

See syllabus page for online readings and lecture outlines (no textbook).
Waitlisted? Please sign up on the yellow notepad. Continue attending.

Homework h1 assigned tomorrow night, due 10 pm Friday, January 29th

Last Time

Collections

- bag intro
- abstract data types and data structures
- designing an Integer Bag ADT – Java interfaces
- using the Integer Bag ADT – review of autoboxing

Characteristics of Good & Reusable Software
Generalizing the Integer Bag ADT – Java `Object`
Implementing a General Bag ADT

Today

Implementing the Bag ADT

- casting when using Object
- using Java generics for generality

List ADT

- designing the ListADT
- coding the ListADT as a Java interface

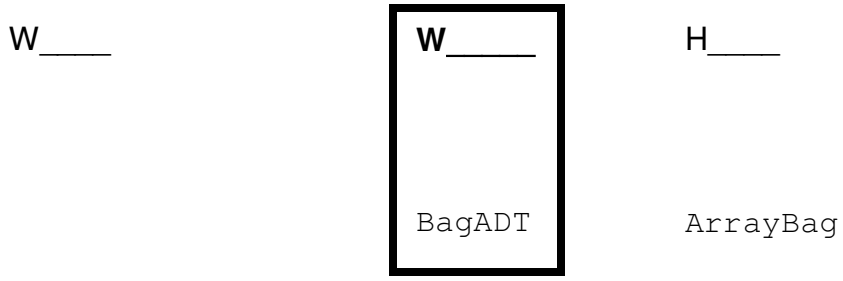
Next Time

Read: continue *Lists*
Lists

- using lists via the ListADT
- implementing the ListADT using an array (SimpleArrayList)

Java API Lists
Iterators Concept

Recall the Bag ADT



❋

1. Bag ADT Design & Interface

A Bag is a general unordered container of items where duplicates are allowed.

```
import java.util.*;

public interface BagADT {

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

→ Why are we using the `Object` class in our `BagADT` interface?

2. Bag ADT mplementation

```
public class ArrayBag implements BagADT {

   private Object[] items;
   private int numItems;

   public ArrayBag() {
      items = new Object[100];
      numItems = 0;
   }

    void add(Object item) { ... }
    Object remove() throws NoSuchElementException { ... }
    boolean isEmpty(){ return numItems == 0; }
}
```

Use - `BagADT` and Casting

Using a general ADT and its implementation to instantiate a container:

→ **Write a statement** that makes a Bag ADT container named `bag`.

→ Assume `Die` is a class representing dice and has a zero parameter constructor.
Write a code fragment that adds 6 dice to `bag`.

→ Assume the bag has had items added to it. **Why doesn't the following code compile?**

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

Java Generics - A Better Way to Make a General Bag ADT

What changes are needed to make the interface below generic?

```
import java.util.*;

public interface BagADT      {

   void add(Object item);

   Object remove() throws NoSuchElementException;

   boolean isEmpty();

}
```

1.

2.

Implementation - Generic `BagADT`

What changes are needed to make the implementation below generic?

```
public class ArrayBag    implements BagADT    {

   private Object[] items;

   private int numItems;


   public ArrayBag() {

      items = new Object[100];


      numItems = 0;
   }


    boolean isEmpty(){ return numItems == 0; }

    void add(Object item) { ... }

    Object remove() throws NoSuchElementException { ... }
}
```

1.

2.

Use - Generic Bag ADT

How do we use a generic interface and its generic implementation to make a container?

→ **Write a code fragment** to make one generic Bag ADT container storing `String` objects and another one storing `Die` objects.

→ **Write a statement** to add "cs367" into the appropriate container.

→ **What happens with…?**

→ Can we make a single generic Bag ADT container that can store both `String` and `Die` objects at the same time?

Design - List ADT

Concept

Operations

- add item at end of list

- add item at specified position
- get item at specified position
- remove item at specified position

- check if list contains a specified item
-
- get size of list (number of items it contains)
- check if list is empty

Issues

Null item – detect then signal with `IllegalArgumentException`

Bad position – detect then signal with `IndexOutOfBoundsException`

Empty list – handle as a bad position

Interface - Generic `ListADT`

```
/**
 * A List is a general container storing a contiguous collection
 * of items, that is position-oriented using zero-based indexing
 * and where duplicates are allowed.
*/
public interface ListADT <E> {

   /**
    * Add item to the end of the List.
    *
    * @param item the item to add
    * @throws IllegalArgumentException if item is null
    */
   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 IllegalArgumentException if item is null
    * @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);
```

Interface - Generic `ListADT` (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);
}
```