CS 367 - Introduction to Data Structures
Week 1

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

Instructors
- Lec 2 & 1: Deb Deppeler, 5376 CS, deppeler@cs.wisc.edu
  website: http://pages.cs.wisc.edu/~cs367-1/
- Lec 3: Alexi Brooks, 4369 CS, albrooks@cs.wisc.edu

Course Website: https://canvas.wisc.edu/
If Canvas is down, we post updates on: http://pages.cs.wisc.edu/~cs367-1/
See modules links for online readings and lecture outlines (no textbook)
zyBooks? Online interactive text recommended for students seeking more practice
Homework 0 available, complete as soon as possible

Read: Introduction, Course Info pages, Lists
This Week in Lecture:
  What makes software good?
  Collections
    - bag intro
  Abstract Data Types and data structures (ds)
  Designing an Integer Bag ADT – Java interfaces
  Using the Integer Bag ADT – review of autoboxing
  Generalizing the Integer Bag ADT – Java Object
  Implementing a general Bag ADT
    - casting when using Object
  Making a generic Bag using Java Generics
  ListADT (the list operations you need to know)
    - designing the List ADT
    - coding the List ADT as a Java interface
    - using lists via the List ADT
    - implementing the List ADT using an array (class SimpleArrayList)

Next Week
  Java API Lists
  Iterators
    - concept
    - iterators and the Java API
    - using iterators
What makes software good?

What makes code reusable?
Collections

→ What is a *collection*?

Definition:

Related terms:

→ What operations can you do on a collection?

Place a star by the operations which are the most fundamental?
Example: Bags

Concept

Operations

Problems

→ What problems might occur when doing Bag operations?
ADTs - Abstract Data Types

ADTs vs. Data Structures

Abstract Data Type (ADT)

Data Structure (DS)
Designing an Integer Bag ADT

Conceptual Description

Public Interface

Coding Issues

Example 1: Using the Integer Bag ADT

→ Write a code fragment
to put the numbers 0 through 99 into an Integer Bag ADT named bag.

    IntegerBagADT bag = new ...;  //assume bag has been instantiated here
Example 2: Using the Integer Bag ADT

→ Complete the printBag method so that it prints the contents of the parameter bag.

Challenge: Implement your printBag method so that it doesn’t change the bag’s contents.

    public static void printBag( IntegerBagADT bag ) {

Generalizing the Integer Bag ADT

→ What class in Java can be used to reference any Java object?

→ What modifications are needed to generalize our Integer Bag ADT?

```
import java.util.*;

public interface IntegerBagADT {
    void add(Integer item);
    Integer remove() throws NoSuchElementException;
    boolean isEmpty();
}
```

Comments:
Implementing BagADT Using an Array of Object References

public class ArrayBag

    //instance variables

    //constructor

    //BagADT methods

    // could add other methods specific to the array implementation
}
Recall the Bag ADT

WHY?
WHAT?
HOW?

1. Bag ADT Design & Interface

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

```java
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 implementation

Steps 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?**

```java
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?

```java
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?

```java
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 pos is less than 0
     * @throws IndexOutOfBoundsException if pos is 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);