Department of Computer Science
Undergraduate Events

Events this week

**Resume Editing Drop-In Session**
- **Date:** Mon., Feb 1
- **Time:** 11 am – 2 pm
- **Location:** Rm 255, ICICS/CS

**EADS Info Session**
- **Date:** Mon., Feb 1
- **Time:** 3:30 – 5:30 pm
- **Location:** CEME 1202

**Job Interview Practice Session (for non-coop students)**
- **Date:** Tues., Feb 2
- **Time:** 11 am – 1 pm
- **Location:** Rm 206, ICICS/CS

**RIM Info Session**
- **Date:** Thurs., Feb 4
- **Time:** 5:30 – 7 pm
- **Location:** DMP 110

Events next week

**Finding a Summer Job or Internship Info Session**
- **Date:** Wed., Feb 10
- **Time:** 12 pm
- **Location:** X836

**Masters of Digital Media Program Info Session**
- **Date:** Thurs., Feb 11
- **Time:** 12:30 – 1:30 pm
- **Location:** DMP 201

Administrivia

- Lecture slides (day by day) are on the web:
- Assignment #2 is out
  - Due Thursday February 11, 10:00pm
- Office hours
  - I only guarantee to stay in my office for the first 15 minutes
  - Mostly, you’ve been walking over with me so this works
  - If you want to come by later in the hour, just let me know (within the first 15 minutes!)
Where are we?

- We are wrapping up the portion of the course on design
- Moving on to becoming more effective Java programmers

How many errors can you find in this design?  

Note: The default multiplicities are 1
A Better Design

Introduction to Collections: the List interface

- compare and contrast the use of a List over an array
- know how and when to use a List data structure
- compare and contrast the use of generic data structures and arrays of type Object
- compare and contrast assignment with various generic collections under specific subclass scenarios
- use wildcards appropriately in generic type parameters to enable assignment in subclass scenarios

Reading
- 3rd and 4th Ed: Chapter 17; Skip 17.2
- 2nd Ed: Chapter 22; Skip 22.2

Exercises
- 4th Ed: P17.1, P17.2, P17.3, P17.14
- 3rd Ed: P17.1, P17.2, P17.3, P17.13
- 2nd Ed: P22.1, P22.2, P22.3, P22.13
Course Structure

- So far…
  - we’ve considered how to design and implement robust classes
- Now…
  - we’re going to look at how to represent collections of information (objects) so that we can build programs that do more
- Then…
  - we’re going to some programming concepts that will help you build even more interesting programs (Streams, GUI, Threads)

Why arrays aren't enough…

- Objects often have to store collections of references to other objects
  - e.g., a bank has a collection of accounts
- To this point, you have used arrays to store such collections

  e.g., public class Bank {
      private Account[] accounts; ...
  }

- But…
  - We have to decide the size of an array when we allocate it.
  - If the array fills, it doesn't expand automatically. We have to write code to create a bigger array and copy the data over.
Collections of Objects

- Sometimes we want to create objects that store a collection of other objects of an unspecified type.

- For example, we might want to create a list class that can store a list of any other type of object (e.g., a list of `String` or `Account` or `Point`).

- We can achieve this by storing the items in the collection in an array of type `Object`:

```java
public class MyList {
    private Object[] items;
    private int numItems;
    final int INIT_NUM=10;

    public MyList {
        items = new Object[INIT_NUM];
        numItems = 0;
    }

    public void add( Object item ){...}
    public Object get( int index ){...}
    public boolean isEmpty(){...}
    public int size(){...}
}
```
Some Problems with MyList

• As a user of such a list, we have no problem adding items to the list:

        myList.add( new Account() );

• …but we've got to be careful when we retrieve items from the list

        if ( myList.get( 0 ) instanceof Account )
            Account acc = (Account) myList.get( 0 );

More Problems with MyList

• The fact that we can add any type of object to our List can be problematic.

• Suppose we want to create a list of Account objects:

        MyList myList = new MyList();

        myList.add( new Account() );
        myList.add( new Account() );
        myList.add( new Account() );
        myList.add( new KitchenSink() );
        myList.add( new Account() );

• The compiler won't flag the fact that we've added a KitchenSink to our list of Account objects – ugh.
List

• A List is an interface defined in the Java libraries.
• An object of type List acts like an array except that it automatically grows and shrinks as needed.
• There are several kinds of List classes which differ in their performance characteristics
  • ArrayList, Vector, LinkedList, etc.
  • Details are described in CPSC 221
  • We will use an ArrayList for this lecture

List

• A List is an example of a generic interface/class.
• We specify the type of data to be stored in the list when a List is declared and instantiated:
  • List<Account> accts = new ArrayList<Account>(); // a list of Account objects
  • List<String> strings = new ArrayList<String>(); // a list of String objects
List

- The compiler will not allow us to add objects of the wrong type:
  - `List<Account> accts = new ArrayList<Account>();`
  - `accts.add( new Account() ); // OK`
  - `accts.add( new Account() ); // OK`
  - `accts.add( new KitchenSink() ); // won't compile`

- This is a good thing. The compiler will now check that we're adding the right type of object to our list.

List

- It's also easy to retrieve items from the list.
- Recall that when we retrieve an item from `MyList`, we have to cast to the appropriate type.
  The cast is not necessary when working with a generic `List`.
- Let's assume that we're working with the `List` declared on the previous page and that we've inserted a few `Account` objects into the list:
  - `Account myAccount = accts.get( 0 ); // Gets the account at position 0 in the list`

- No cast is necessary.
List

• Given:

\[
\text{List<Account> accts = new ArrayList<Account>();}
\]

• we can add objects of type Account
• we can also add objects that are a subtype of Account
• So, if SavingsAccount is a subclass of Account, we can do the following:

\[
\text{accts.add( new Account() );}
\]

\[
\text{accts.add( new SavingsAccount() );}
\]

List Methods

• List has many useful methods:

```
public interface List<E> {
...
public boolean add( E item )
    // add at end of list
public boolean add( int i, E item )
    // insert at specific position i
public boolean contains( Object item )
    // is item in the accounts collection
public E get( int i )
    // get item at position i
public E remove( int i )
    // remove account at position i
public int size()
    // gets number of elements in list
    // NOT current capacity of list
...
```

E is a generic parameter