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 looking at:
  - Collections (as a generalization of arrays)
  - Generics
  - Focusing on ArrayList

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 Methods

- List has many useful methods:
  ```java
  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

Java Generics

- Note that the E in the List API is a **generic parameter** (or **type parameter**):
- E represents the type that is specified by the client when the List is declared and instantiated
- For example:

  ```java
  List<Account> accList;
  // E is Account
  
  List<String> strList;
  // E is String
  ```

- For the full API for this interface, see the online docs:
Example using List

```java
public class Bank {
    private List<Account> accounts;

    public Bank() {
        accounts = new ArrayList<Account>();
    }

    // Add new account at the end of List
    public void newAccount(double balance) {
        accounts.add(new Account(balance));
    }

    // Get number of accounts at Bank
    public int getNumAccounts() {
        return accounts.size();
    }
}
```

Example 2

• Complete the following method that counts the number of times a particular string is found in an `List`

```java
public static int count(List<String> list, String toFind) {
    int count = 0;
    for (int i = 0; i < list.size(); ++i) {
        if (list.get(i).equals(toFind)) {
            count += 1;
        }
    }
    return count;
}
```
Assignment with Arrays and subclasses

- Assume that `SavingsAccount` is a subclass of `Account`. Consider this:

  Account[] acc = new Account[10];
  SavingsAccount[] sAcc = new SavingsAccount[10];

- Is this allowed?

  acc[0] = new SavingsAccount();
  SavingsAccount sa = acc[0];

This diagram shows the relationship between `Account` and `SavingsAccount`:

```
Account
      △
  SavingsAccount
```

- What about this?

  acc = sAcc;
  acc[0] = new Account();
  // oops - just put an Account into an array of SavingsAccount objects

This does compile but can lead to problems if we then do

  acc[0] = new Account();
  // oops - just put an Account into an array of SavingsAccount objects

...problem isn't detected by compiler.

An exception is thrown when the program runs – nasty.
Assignment with Generics

• Consider this:

```java
List<Account> accounts = new ArrayList<Account>();
List<SavingsAccount> savingsAccounts = new ArrayList<SavingsAccount>();
```

• Is this allowed?

```java
accounts.add(0, new SavingsAccount());
SavingsAccount sa = accounts.get(0);
```

Assignment with Generics

• What about this?

```java
(accounts = savingsAccounts;)
```

• This code does not compile and so the problem illustrated with arrays earlier is avoided.
Assignment with Generics

• Assume we have the method:
  
  ```java
  public void myMethod(List<Account> list) {...}
  ```
  
  then the following client call will also not compile:
  
  ```java
  List<SavingsAccount> savAccs = new ArrayList<SavingsAccount>();
  myMethod( savAccs );
  ```

Motivating Wildcards

• Imagine that we want to add a method to Bank that will take a list of accounts and send a directed advertisement to their owners

  ```java
  public void spam(List<Account> targetAccounts) ...
  ```

• We have a problem. We may want to spam a list of SavingsAccount but we cannot write:

  ```java
  List<SavingsAccount> savingsAccounts = new ArrayList<SavingsAccount>();
  Bank b = new Bank();
  b.spam( savingsAccounts );  //not allowed
  ```
Bounded Wildcards

• In such cases we can use wildcards in the type parameter:

    public void spam(
        List<? extends Account> targetAccounts )
    {...}

• `<? extends Account>` indicates that we can pass a List of any type that is a subtype of Account

• So we can now pass a List of Account or SavingsAccount or any other type that’s a subtype of Account.

Bounded Wildcards - Question

• When we use a bounded wildcard, we can visit the items in the collection but we are not allowed to add an item to the collection.

    public void spam(List<? extends Account> targetAccounts )
    {
        targetAccounts.add( new Account() );
        //...
    }

• Why is this not allowed?
Lists and beyond...

• Suppose that we want to maintain a list of objects, but without allowing duplicates.
• Can we use a List for this purpose?
  
  Yes, but...

• It would be nice if there was another, similar class, that does not allow duplicates.

• Java library provides a family of such classes called the **Collection Classes**

---

Java Collections Framework

• explain the structure of the Java Collections framework.
• program to the generic Collection interface including reading and using the APIs
• program to the generic Iterator interface including reading and using the APIs
• read and write code that uses a for each loop to iterate over a collection
• determine when a for-each loop can be used and how to avoid concurrent modification of a collection

---

**Reading:**

Java Tutorial on Collections:
http://java.sun.com/docs/books/tutorial/collections/index.html

Lessons: Introduction and Interfaces
Java Collections Framework

- We have examined the use of one collection interface, List, and one collection class, ArrayList, and observed that we sometimes need other classes that support very similar operations (with some differences).
- We will now see how Java uses a hierarchy of interfaces to abstract the common behaviours that are shared by these classes.
- This hierarchy is called the *Java collections framework*.

Java Collections Framework

- The Collections Framework is in the java.util package.
- The interfaces and classes in this package provide
  - standardized interfaces with multiple implementations of most data structures (e.g., List, Set, etc.)
  - efficient, highly-optimized implementations of common data structures (e.g., ArrayList)
  - interoperability between programs by making it easier to exchange collections
Java Collections Framework

Consists of 3 components:

- **Interfaces**
  - provide specifications for the behaviour of the collections
  - form inheritance hierarchies
- **Implementations**
  - provide specific structures that store the elements and relevant operations on those structures
  - each interface may have multiple implementations that differ only by which optional operations they implement and by the efficiency of the operations
- **Algorithms**
  - polymorphic algorithms that manipulate data stored in collections
  - are not members of any collection

Some Collection Interfaces

```
<<interface>>           <<interface>>           <<interface>>
   Iterable             Iterator
                  <<interface>>
             Collection
             <<interface>>
               List
                   ArrayList
```