Events this week
Resume & Cover Letter Drop-In Session
Date: Wed., Mar 3
Time: 12 – 3 pm (20 mins. sessions)
Location: Rm 255, ICICS/CS

Find a Job Fast! Info Session
Date: Thurs., Mar 4
Time: 12:30 – 1:45 pm
Location: DMP 201
Registration: Email dianejoh@cs.ubc.ca

Townhall Meeting – 1st Year CS Students
Date: Thurs., Mar 4
Time: 12:30 - 2 pm
Location: DMP 310
Lunch will be provided!

Faculty Talk – Son Vuong
Title: Mobile Learning via LIVES
Date: Thurs., Mar 4
Time: 12:30 – 1:45 pm
Location: DMP 201

Events next week
Townhall Meeting – Combined Majors/Honours, BA, B.Comm in CS
Date: Thurs., Mar 11
Time: 12:30 – 2 pm
Location: DMP 310
Lunch will be provided!

CS Distinguished Lecture Series – Featuring David Parkes
Title: Incentive Mechanism Engineering in the Internet Age
Date: Thurs., Mar 11
Time: 3:30 – 4:50 pm
Location: DMP 110

CSSS Movie Night – “Zombieland” & “Iron Man”
Date: Thurs., Mar 11
Time: 6 – 10 pm
Location: DMP 310
Free pop & popcorn!

Administrivia
• Lecture slides (day by day) are on the web:
  • http://www.cs.ubc.ca/~norm/211/2009W2
• Assignment #3 is out
  • Due Thursday March 18, 10:00pm
• Midterm exam
  • Wednesday March 10, 6:00pm
  • Math 100 (BCS and 202 sections)
  • If you have a conflict, talk to me today
• Final exam schedule is now posted
  • CPSC 211 - Apr 23, 7:00pm
  • No location yet announced
The LinkedList Class

We'll look at how a LinkedList is implemented later in the course. For now we simply focus on the differences between ArrayList and LinkedList. Let's consider the time complexity of some common operations in the worst case:

<table>
<thead>
<tr>
<th>Operation</th>
<th>LinkedList</th>
<th>ArrayList</th>
</tr>
</thead>
<tbody>
<tr>
<td>get( int index )</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>add( int i, E e )</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>add( E e )</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>remove( int index )</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>contains( Object o )</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>ListIterator -&gt; add</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>ListIterator -&gt; remove</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

List Example

Reverse a list

```java
public static <E> List<E> reverse(List<E> list) {
    List<E> newlist = new ArrayList<E>();
    Iterator<E> itr = list.iterator();
    while ( itr.hasNext() ) {
        newlist.add(0, itr.next() );
    }
    return newlist;
}
```

What is its complexity?
List Example

A better way to write this method:

```java
public static <E> List<E> reverse(List<E> list) {
```

The `Set` Interface

You will be expected to:

- program to the generic Set and SortedSet interfaces including read and use the API’s
- justify the use of a set vs. a list for a given problem
- compare and contrast the HashSet and TreeSet classes (benefits of using each, basic run time analysis)
- design and implement a class in such a way that it can be used with the Java collections framework
  - overrides equals and hashCode
  - implements the generic Comparable and Comparator interfaces to account for multiple sorting criteria

Reading:

- 3rd Ed: 14.8, 16.1, briefly: 16.3, 16.4
- 4th Ed: 14.8, 16.1, briefly: 16.3, 16.4
Using a Set

In a List,

- Each object has a position
- We can put the same object in the list multiple times

Sometimes, we need the functionality of a mathematical set

- No duplicates in the set
- Members do not have a position in the set

For example…

- MenuItems that appear in the Menu of a restaurant
- Songs that appear on a PlayList
- Student enrolled in a Course

In these cases we use a Set not a List

The Set interface

![Diagram showing the inheritance hierarchy of Set interfaces, including Iterable, Collection, List, Set, HashSet, TreeSet, and SortedSet.]
Methods of the `Set` interface

Note that the `Set` interface extends the `Collection` interface. An implementation of `Set` therefore supports the methods defined in the `Collection` interface:

- `add(o)` – add a specified element (if not already a member)
- `remove(o)` – remove the specified element
- `contains(o)` – is the specified element in the set?
- etc.

Note that the `add` method:

```java
public boolean add( E item );
```

adds the item only if it isn't already in the set. The method returns true if the item is added and false if it's already in the set.

Similarly the `addAll` method does not add duplicates.

Methods of the `Set` interface

We can use these methods to define known set operations:

- `c1.containsAll(c2)` - true if `c2` is a subset of `c1`
- `c1.addAll(c2)` - `c1` becomes union of `c1` and `c2`
- `c1.retainAll(c2)` - `c1` becomes intersection of `c1` and `c2`
- `c1.removeAll(c2)` - `c1` becomes set difference of `c1` and `c2`
Using Sets

Since Set is specified as an interface, to use it we have to pick a particular implementation (e.g., HashSet, or TreeSet).

Example:
```java
public class PlayList {
    private Set<Song> songs;

    public PlayList() {
        songs = new HashSet<Song>();
    }
}
```

The HashSet implementation

The HashSet implementation provides an efficient implementation of the Set interface that allows us to add or remove an item or check if the set contains an item in O(1) time provided certain conditions are met (more later).

That is, if
```java
    Set<...> s = new HashSet<...>();
s.add(o) is O(1)
s.remove(o) is O(1)
s.contains(o) is O(1)
```
The HashSet implementation

As mentioned, certain conditions must be met if we are to add, remove or determine if the set contains an item in O(1) time.

To understand these conditions, we must have a basic understanding of how the hash set works.

A hash set uses a hash table as the underlying structure in which data is stored.

A hash table is an array of linked lists…

The HashSet implementation

We add elements to the table using a hash code, an integer that represents the object

A hash set maintains a list of groups.
All members of the group at position i have a hashCode of i.

Let’s see an example…. 
Hash Table Example

Suppose we want to add integers to a hash table using the following hash code:

```java
hashCode = value%10;
```

What does the table look like after inserting:
243556,
329394,
3348,
436,
3234,
424

```
0
1
2
3
4
5
6
7
8
9
```

- 243556 - 3234 - 3234
- 329394
- 3348 - 436