Announcements

- Attendance for this week’s recitation is mandatory!
- A2 is due Wednesday
- Get started on A3 – do one method a day

Overview

- Big Demo!
- Interfaces
- Abstract Classes
- Normal Classes vs. Abstract Classes vs. Interfaces

Implementing Interfaces

```java
/** A range of integers that always includes 0 */
public class IntRange implements Collection<Integer> {
    private int min = 0; // Represents the range min..max
    private int max = 0; // min <= max
    /** Return true if elem is an integer in the range. */
    public boolean contains(Object elem) {...}
    /** Minimally extend the range to include elem. */
    public boolean add(Integer elem) {...}
}
```

Provides implementations for interface methods

Using Interfaces

```java
/** Returns whether the collection contains every */
/** integer between and including min and max. */
/** Precondition: ints is not null */
public static boolean containsRange(
    Collection<Integer> ints, int min, int max) {
    for (int i = min; i <= max; i++)
        if (!ints.contains(i))
            return false;
    return true;
}
```

Works on any Collection!!
This includes your own DLinkedList from A3!

Because ints has type Collection<Integer>, you can use any method declared in the Collection<Integer> interface.
Extending Interfaces

```java
/**
 * A mutable indexed list of E values *
 */
public interface List<E> extends Collection<E>
{
    int size(); // return size of the list
    E get(int index); // return elem at index
    E set(int index, E elem); // change elem at index
    E remove(int index); // remove and return elem at index
    ...
}
```

Implicitly includes all methods in Collection<E>

Abstract Classes

```java
/**
 * Provides default implementations for list methods *
 */
public interface List<E> extends Collection<E>
{
    int size(); // return size of the list
    E get(int index); // return elem at index
    E set(int index, E elem); // change elem at index
    E remove(int index); // remove and return elem at index
    ...
}
```

Indicates that subclasses are responsible for providing the implementation

Only abstract classes can have abstract methods

Defaults in Java 8

```java
/**
 * Provides default implementations for list methods *
 */
public interface List<E> extends Collection<E>
{
    int size(); // return size of the list
    E get(int index); // return elem at index
    E set(int index, E elem); // change elem at index
    E remove(int index); // remove and return elem at index
    ...
}
```

Indicates that the interface is providing a default implementation for this method

Abstract Classes Revisited

```java
public abstract class IntExpression {
    private Integer value = null;
    public int evaluate() {
        if (value == null) value = eval();
        return value.intValue();
    }
    protected abstract int eval();
}
```

Abstract class provides common fields and functionality

Abstract class leaves critical methods abstract for subclasses to implement

Subclasses provide case-dependent implementations

Comparison

<table>
<thead>
<tr>
<th>Normal Classes</th>
<th>Abstract Classes</th>
<th>Interfaces</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>can be used as types and in casts</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>can be newed</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>have constructors</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>can have fields</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>can provide method implementations</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>can have non-public methods</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>can have abstract methods</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>can be inherited multiply</td>
</tr>
</tbody>
</table>