Collections

The Plan
- Why use collections?
- What collections are available?
- How are the collections different?
- Examples
- Practice

Why use collections?
Consider the code below. What if you wanted 1000 grades? Why is this code not designed well?

```java
int grade0, grade1, grade2, grade3, ..., grade100;
grade0=Integer.parseInt(field0.getText());
grade1=Integer.parseInt(field1.getText());
...
grade100=Integer.parseInt(field100.getText());
int sum=grade0+grade1+grade2+...+grade100;
average.setText("average is "+sum/100.0);
```

Collections & Loops
Recall:

- Loops
  - group repeatedly executed code for uniformity
  - make the number of repetitions easily changeable
  - can be combined with selection to make more complex algorithms
Collections Motivation

Collections help enable

- declaring multiple variables
- naming multiple variables
- grouping similar variables under one name
- grouping similar code that acts on the variables
- changing the number of variables easily
- implementing more complex algorithms

Why use collections?

The code below uses an array to average the 1000 grades. What change would make it do 10 grades?

```java
int[] grade=new int[1000];
int sum=0;
for(int i=0; i<grade.length; i++)
{
    grade[i]=Integer.parseInt(field[i].getText());
sum+=grade[i];
}
average.setText("average is "+sum/grade.length);
```

What an array looks like

<table>
<thead>
<tr>
<th>grade</th>
<th>grade[0]</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>grade[1]</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>grade[2]</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>grade[3]</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>grade[5]</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>grade[6]</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>grade[n-2]</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>grade[n-1]</td>
<td>92</td>
</tr>
</tbody>
</table>

What collections are available?

- Arrays
- java.util.Collection
  - HashSet
  - LinkedList
  - LinkedHashSet
  - ArrayList
- java.util.Map
  - HashMap
  - TreeMap
Arrays

- Store primitives or particular Objects
- Size is immutable
- Contain length field
- Is an Object
- Indexed 0 to length-1
- Can generate ArrayIndexOutOfBoundsException

java.util.Collection

- Store Objects directly
- Size is typically dynamic
- Has a size() method
- Is an Object
- Indexing varies
- Has toArray(Object[]) method for converting to an array.

java.util.Map

- Store Object-to-Object mapping
- Size is typically dynamic
- Has a size() method
- Is an Object
- Indexing by keys
- Use keySet().toArray(Object[]) method for converting keys to an array.
- Use valueSet().toArray(Object[]) method for converting values to an array.

HashSet

- Fast for
  - insertion
  - removal
  - checking membership
- Uses 1-way function (like used with passwords)
- Not great for
  - ordering
  - iterating over members
  - memory use
### LinkedList
- **Fast for**
  - insertion at ends
  - removal at ends
- **Slow for**
  - insertion in middle
  - removal from middle
  - checking membership
- **Minimal memory waste**
- **Used for stacks and queues**

### LinkedHashSet
- Combines good elements of LinkedList and HashSet
- **Fast**
  - insertion
  - removal
  - check for membership
- **Keeps insertion and removal order**
- **Fast for iteration**
- **Memory still underutilized**

### ArrayList
- **Fast**
  - insertion
  - removal
  - random access
- **Slow**
  - check for membership
- **Great for**
  - iterating over members
  - memory use
  - ordering

### HashMap
- Like the HashSet except contains key->value mapping.
**TreeMap**

- Relatively fast
  - insertion
  - removal
  - checking membership
- Best for maintaining order
  - always in sorted order by keys
  - uses the `compareTo` method

**Arrays**

- Fastest
  - Access
- Slower
  - insertion
  - removal
- Very general purpose
- Immutable size can be a problem
- When full, most memory efficient
- Best for holding primitives
- Best when used in conjunction with Collections and Maps

**Examples**

- Sprites in AnimationCanvas
  - LinkedHashSet
  - array
- Alarms in FrameAdvancer
  - TreeMap
- Shapes in BlurSprite
  - LinkedList

**Sprites in AnimationCanvas**

```java
public class AnimationCanvas extends JPanel {
    private LinkedHashSet sprites;
    private Color background=Color.WHITE;
    private Sprite[] array;
    private boolean dirty;
}
```
public void addSprite(Sprite sprite) {
    sprites.add(sprite);
    dirty=true;
}

public boolean containsSprite(Sprite sprite) {
    return sprites.contains(sprite);
}

public void updateSprites() {
    if (dirty) {
        array=(Sprite[])sprites.toArray(new Sprite[0]);
        dirty=false;
    }
    for (int i=0; i<array.length; i++) {
        if (array[i].isDestroyed()) {
            removeSprite(array[i]);
        }
        array[i].updateInternal();
    }
}

private void paintSprites(Graphics2D brush) {
    if (dirty) {
        array = (Sprite[]) sprites.toArray(new Sprite[0]);
        dirty = false;
    }
    for (int i = 0; i < array.length; i++) {
        array[i].paintInternal(brush);
    }
}
public class FrameAdvancer implements Runnable {
    protected static AnimationCanvas canvas = new AnimationCanvas();
    ... 
    private static TreeMap alarms = new TreeMap(); 
    private static Thread worker;
}

private static void soundAlarms() {
    if (alarms.isEmpty())
        return;
        double timeToAlarm = ((Double)alarms.firstKey()).doubleValue();
    while (timeToAlarm < totalTime) {
        Alarm alarm = (Alarm) alarms.remove(alarms.firstKey());
        alarm.alarm();
        if (alarms.isEmpty()) {
            break;
        }
        timeToAlarm = ((Double)alarms.firstKey()).doubleValue();
    }
}

public class BlurSprite extends Sprite {
    private LinkedList previous = new LinkedList();
    ... 
    public void update() {
        super.update();
        drawNumber++;
        int frequency = Math.max(1, 
        FrameAdvancer.getUpdatesPerFrame() / numPerFrame);
        if (drawNumber % frequency != 0) {
            return;
        }
        boolean bounding = useBoundingBox();
        setUseBoundingBox(false);
        previous.addLast(getShape());
        setUseBoundingBox(bounding);
        if (previous.size() > numFrames * numPerFrame) {
            previous.removeFirst();
        }
    }
}
Practice

Write code on paper to
- Declare an array of integers
- Initialize the array to be able to hold 10 integers
- Set the values in the array to be the first ten squares (i.e. 1, 4, 9, 16, 25 ...)
- Sum the values
- Output the average
- Alter your code to do the first 100 integers instead

Practice

Write code on paper to
- Declare an array of JButtons
- Initialize the array to be able to hold 12 buttons
- Make 12 buttons and put them into the array. The text on the button should be the integers 0-9 and '.', '-'.
- Write a method to put the buttons into a 4x3 grid on a panel.

Practice

Write code on paper to
- Declare a HashSet variable called bullets
- Initialize the HashSet variable
- Put 10 Sprites of your choice into the bullets hash set
- Write a loop to go through the bullets hash set and print the location of all Sprites.
- Modify the loop above to print only those Sprites which are enabled.

Practice

Complete the following method
```java
//return all Sprites intersecting with target Sprite
public Sprite[] getIntersecting(Sprite target, Sprite[] all) {
    LinkedList list = new LinkedList();
    for(int i = 0; i < all.length; i++) {
        if(all[i] == target) {
            list.addLast(all[i]);
        }
    }
    return (Sprite[]) list.toArray(new Sprite[0]);
}
```