Outline

- Array Lists
- Enhanced for Loop
- ArrayList and LinkedList
- Collection Interface
- Sets and Maps
  - Section 6.8, 15.1, 15.2, 15.4 and 15.5
- slides are available at [www.dcs.bbk.ac.uk/~roman/sp1](http://www.dcs.bbk.ac.uk/~roman/sp1)
Arrays

- an array collects a sequence of values of the same type

```java
// empty array of 5 Students (objects)
Student[] students = new Student[5];
// list of initial values
String[] data = { "I", "V", "X", "L" };
```
The Length of Arrays is Fixed

come up with a guess on the maximum number of elements and keep a companion variable for the current size

```java
final int LENGTH = 100; // max number of elements
double[] data = new double[LENGTH];
int currentSize = 0; // companion variable
Scanner in = new Scanner(System.in);
while (in.hasNextDouble()) // read all doubles
    if (currentSize < data.length) {
        data[currentSize] = in.nextDouble();
        currentSize++;
    }
for (int i = 0; i < currentSize; i++) // start at 0!
    System.out.println(data[i]); // print them
```
Removing an Element

removing the element at the position pos:

```java
for (int i = pos; i < currentSize - 1; i++)
    data[i] = data[i+1]; // move elements up

currentSize--; // update currentSize
```

SP1 2015-10
Inserting an Element

inserting a newElement at the position pos:

```java
if (currentState < data.length) {
    for (int i = currentState; i > pos; i--) // reverse
        data[i] = data[i-1]; // move elements down
    data[pos] = newElement; // store the new element
    currentState++; // update currentState
}
```
// String is the type of elements
ArrayList<String> friends = new ArrayList<String>();
...
// append an element to the array, increasing its size
friends.add("Allan");
// accessing elements: the index i must be
// >= 0 and < friends.size()
String name = friends.get(i);
friends.set(i, "Harry");
// print out the contents
System.out.println(friends);

**NB:** do not forget = new ArrayList<...>() (otherwise, it is null)
for (String name: friends) {
    System.out.println(name);
}

// is equivalent to
for (int i = 0; i < friends.size(); i++) {
    String name = friends.get(i);
    System.out.println(name);
}

**NB:**
- array: a.length
- String: a.length()
- array list: a.size()

arrays are part of language
strings and array lists are objects
so, it is a method call
Inserting and Removing Elements

```java
ArrayList<String> friends = new ArrayList<String>();
friends.add("Tom"); // adds an element to the end
friends.add("Laura"); // now ["Tom", "Laura"]
// inserts an element at index 1
friends.add(1, "Giles");
// now ["Tom", "Giles", "Laura"]
// removes the element at index 0
friends.remove(0); // now ["Giles", "Laura"]
```

**NB:** inserting and removing elements is slow:
on average, takes $n/2$ steps for an array with $n$ elements
LinkedList: Inserting Elements

LinkedList<String>

head

Node
value = "Tom"
prev = null
next

Node
value = "Laura"
prev
next = null

Node
value = "Giles"
prev
next
public void insertAfter(Node current, String s) {
    Node newNode = new Node(s);
    newNode.next = current.next;
    newNode.prev = current;
    current.next.prev = newNode;
    current.next = newNode;
}

**NB:** effective insertion and deletions indexing, however, is slow:
finding the $n$th element takes $n$ steps
// a list of Strings
LinkedList<String> list = new LinkedList<String>();

// adding elements
list.addLast("Tom"); // to the end
list.addFirst("Laura"); // to the beginning
list.addFirst("Giles"); // to the beginning

// accessing elements
String fname = list.getFirst(); // the first
String lname = list.getLast(); // the last

// removing
String removed = list.removeFirst(); // "Giles"
 LinkedList: Iterators and For Loop

An iterator is used to access elements inside a linked list.

```java
Iterator<String> iterator = list.iterator();
while (iterator.hasNext()) {
    String name = iterator.next();
    // do something with name
}
// is equivalent to
for (String name: list) {
    // do something with name
}
```
LinkedList: Iterators and Removing

```java
Iterator<String> iterator = list.iterator();
while (iterator.hasNext()) {
    String name = iterator.next();
    if (name.equals("Tom")) {
        // removes the last traversed element
        iterator.remove();
        // can be called only once after next
    }
}
```
List Iterator

A list iterator is obtained by using `list.listIterator()`.

```java
public interface ListIterator<E> extends Iterator<E> {
    boolean hasPrevious();
    E previous();
    void add(E e);
    void set(E e);

    int nextIndex();
    int previousIndex();
}
```
both ArrayList and LinkedList implement the Collection interface

public interface Collection<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);
    boolean remove(Object element);
    Iterator<E> iterator();
    ...
}

this is an example of a generic interface: E is any type name
More Collections: Sets

- sets cannot contain duplicate elements (lists can)
- three class implementations
  - HashSet: stores the elements in a hash table
    best-performing implementation, no order
  - TreeSet: stores its elements in a tree
    and orders its elements based on their values
  - LinkedHashSet: a hash table with a linked list running through it, orders its elements based on the order in which they were inserted into the set

more:

docs.oracle.com/javase/tutorial/collections/TOC.html
More Collections: Maps

A map is an object that maps keys to values.

It models the mathematical function abstraction.

Q: Is array a map?

A map cannot contain duplicate keys:
   each key can map to at most one value.

```
public interface Map<K,V> {
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    ...
}
```
import java.util.*;

class Freq {
    public static void main(String[] args) {
        Map<String, Integer> m = new HashMap<String, Integer>();
        for (String a : args) {
            Integer freq = m.get(a);
            m.put(a, (freq == null) ? 1 : freq + 1);
        }
        System.out.println(m.size() + " distinct words:
                         " + m);
    }
}
Take Home Messages

- a collection groups together elements and allows them to be retrieved later
- a list is a collection that remembers the order of its elements
- a set is an unordered collection of unique elements
- a map keeps associations between key and value objects