Java Collections
(data structures)
Java Collections
(data structures)

- Collection:
  - an object that stores data; a.k.a. "data structure"
  - the objects stored are called elements

- Some types of collections are “ordered” (not sorted)
  - ordered: the concepts of “Nth”, “last” and “next” make sense in these
  - some collections maintain an ordering; some allow duplicates
  - typical operations: add, remove, clear, contains (find), size
Java Collections

- Examples of collections that can be found in the Java class libraries:
  - List - ArrayList (ordered)
  - Map - HashMap (NOT ordered)
  - Tree - TreeSet (ordered)

- Collections are in the java.util package

import java.util.*;
Lists
(Java ArrayList)

- A list is a collection storing an ordered sequence of elements
  - each element is accessible by a 0-based index
  - elements can be added to the front, back, or elsewhere
Java ArrayList and “Generics”

Java Form:
ArrayList<Type> name = new ArrayList<Type>();

Example:
ArrayList<Person> people = new ArrayList<Person>();

When constructing an ArrayList, you must specify the type of elements it will contain between < >.
Java ArrayList and “Generics”

Java Form:

ArrayList<Type> name = new ArrayList<Type>();
Java ArrayList and “Generics”

When constructing an ArrayList, you must specify the type of elements it will contain between < >.

- The ArrayList class can be instantiated to contain a list of any type object.
- This is called a type parameter or a generic class.
- These types MUST be objects (vs. primitive types)
  - <int> is a primitive. Can NOT be used with ArrayList
  - use the wrapper class <Integer> with ArrayList
Boxed Primitive Types

You can not use primitives with ArrayList<int> (or other collections) but Java provides “boxed primitives”:

<table>
<thead>
<tr>
<th>Primitive Type</th>
<th>Wrapper Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
Some ArrayList Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(value)</td>
<td>appends value at end of list</td>
</tr>
<tr>
<td>add(index, value)</td>
<td>inserts given value just before the given index, shifting subsequent values to the right</td>
</tr>
<tr>
<td>clear()</td>
<td>removes all elements of the list</td>
</tr>
<tr>
<td>indexOf(value)</td>
<td>returns first index where given value is found in list (-1 if not found)</td>
</tr>
<tr>
<td>get(index)</td>
<td>returns the value at given index</td>
</tr>
<tr>
<td>remove(index)</td>
<td>removes/returns value at given index, shifting subsequent values to the left</td>
</tr>
<tr>
<td>set(index, value)</td>
<td>replaces value at given index with given value</td>
</tr>
<tr>
<td>size()</td>
<td>returns the number of elements in list</td>
</tr>
<tr>
<td>toString()</td>
<td>returns a string representation of the list such as &quot;[3, 42, -7, 15]&quot;</td>
</tr>
</tbody>
</table>
Map

Generically known as:
- dictionary
- associative array
- associative container

Maps use:
  Key, Value pairs to associate data

That is:
“keys” are used to access “values”
Map Interface in Java

Map<String, Integer> hm
    = new HashMap<String, Integer>();

hm.put("a", new Integer(100));
hm.put("b", new Integer(200));
hm.put("c", new Integer(300));
hm.put("d", new Integer(400));
Map Interface Methods

- **put**(k, v)  
  Associate v with k

- **get**(k)  
  The value associated with k

- **size()**  
  The number of pairs

- **isEmpty()**  
  Whether it is empty

- **remove**(k)  
  Remove the mapping for k

- **clear()**  
  Remove all mappings

- **containsKey**(k)  
  Whether contains a mapping for k

- **containsValue**(v)  
  Whether contains a mapping to v