Computational Expression

ArrayList
Iterators

Janyl Jumadinova

11-13 November, 2019
Lists

- **List**: a collection storing an ordered sequence of elements
- Each element is accessible by a 0-based index
- A list has a size (number of elements that have been added)
- Elements can be added to the front, back, or elsewhere
Lists

- **List**: a collection storing an ordered sequence of elements
- Each element is accessible by a 0-based index
- A list has a size (number of elements that have been added)
- Elements can be added to the front, back, or elsewhere
- In Java, a list can be represented as an **ArrayList** object
Lists

- **List**: a collection storing an ordered sequence of elements
- Each element is accessible by a 0-based index
- A list has a size (number of elements that have been added)
- Elements can be added to the front, back, or elsewhere
- In Java, a list can be represented as an **ArrayList** object
## A Few ArrayList Methods

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

- `ArrayList<Type> name = new ArrayList<Type>();`
- When constructing an `ArrayList`, you must specify the type of elements it will contain between `<` and `>`. This is called a **type parameter** or a **generic class**.
- Allows the same `ArrayList` class to store lists of different types.
Type Parameters (Generics)

- `ArrayList<Type> name = new ArrayList<Type>();`
- When constructing an `ArrayList`, you must specify the type of elements it will contain between `<` and `>`. This is called a **type parameter** or a **generic class**.
- Allows the same `ArrayList` class to store lists of different types.
  ```java
  ArrayList<String> names = new ArrayList<String>();
  names.add("Marty Stepp");
  names.add("Stuart Reges");
  ```
Iterators

- One of the most useful operations for any collection is the ability to run through each of the elements in a loop.
- This process is called iteration.
Iterator object

Methods:

- `hasNext()`: returns a boolean value
  - true if there is at least one more item to process
- `next()`: retrieves the next item in the collection to process
Example:
...
ArrayList<String> words = ...; // declared and initialized
Iterator<String> wordsIterator = words.iterator();
    while(wordsIterator.hasNext()) {
        String word = wordsIterator.next();
        ...
    }