Computational Expression

ArrayList

Iterators

Janyl Jumadinova

1-3 April, 2019
Collections

- **Collection**: an object that stores data; a.k.a. “data structure”
- The objects stored are called **elements**
- Some collections maintain an ordering; some allow duplicates
- Typical operations: add, remove, clear, contains (search), size

Examples found in the Java class libraries: ArrayList, LinkedList, HashMap, TreeSet, PriorityQueue. All collections are in the `java.util` package.
Collections

- **Collection**: an object that stores data; a.k.a. “data structure”
- The objects stored are called **elements**
- Some collections maintain an ordering; some allow duplicates
- Typical operations: add, remove, clear, contains (search), size
- Examples found in the Java class libraries: **ArrayList, LinkedList, HashMap, TreeSet, PriorityQueue**
- all collections are in the **java.util** package
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>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>
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.

```
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
Iterators

Several classes in Java API define iterators

- **Scanner**:
  - `hasNext()`: returns true if there is another input token to process
The type you specify when creating an ArrayList must be an object type; it cannot be a primitive type.
The type you specify when creating an ArrayList must be an object type; it cannot be a primitive type.

// illegal -- int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();
The type you specify when creating an ArrayList must be an object type; it cannot be a primitive type.

// illegal -- int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();

But we can still use ArrayList with primitive types by using special classes called wrapper classes in their place.

// creates a list of ints
ArrayList<Integer> list = new ArrayList<Integer>();
Wrapper Classes

- A wrapper is an object whose sole purpose is to hold a primitive value.
- Once you construct the list, use it with primitives as normal:

  ```java
  ArrayList<Double> grades = new ArrayList<Double>();
  grades.add(3.2);
  grades.add(2.7);
  ...
  double myGrade = grades.get(0);
  ```

### Table: Wrapper Types for Primitive Types

<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>