ArrayList and Generics

Chapter 12

JAVA: An Introduction to Problem Solving & Programming, 8th Ed. By Walter Savitch
ISBN 0134462033

Objectives

- Define and use an instance of **`ArrayList`**
- Define and use classes with generic types

Array-Based Data Structures: Outline

- The Class **`ArrayList`**
- Creating an Instance of **`ArrayList`**
- Using Methods of **`ArrayList`**
- Programming Example: A To-Do List
- Parameterized Classes and Generic Data Types

Class `ArrayList`

- Consider limitations of Java arrays
  - Array length is not dynamically changeable
  - Possible to create a new, larger array and copy elements – but this is awkward, contrived
- More elegant solution is use instance of `ArrayList`
  - Length is changeable at run time

Class `ArrayList`

- Drawbacks of using `ArrayList`
  - Less efficient than using an array
  - Can only store objects
  - Cannot store primitive types
- Implementation
  - Actually does use arrays
  - Expands capacity in manner previously suggested

Class `ArrayList`

- Class ArrayList is an implementation of an **A**bstract **D**ata **T**ype (ADT) called a *list*
- Elements can be added
  - At end
  - At beginning
  - In between items
- Possible to edit, delete, access, and count entries in the list

Class `ArrayList`

- Figure 12.1a Methods of class `ArrayList`

`public ArrayList<`*Base_Type*`>(int initialCapacity)` Creates an empty list with the specified *Base_Type* and initial capacity. The *Base_Type* must be a class type; it cannot be a primitive type such as `int` or `double`. When the list needs to increase its capacity, the capacity doubles.
`public ArrayList<`*Base_Type*`>()` Behaves like the previous constructor, but the initial capacity is ten.
`public boolean add(`*Base_Type* `newElement)` Adds the specified element to the end of this list and increases the list's size by 1. The capacity of the list is increased if that is required. Returns true if the addition is successful.
`public void add(int index,` *Base_Type* `newElement)` Inserts the specified element at the specified index position of this list. Shifts elements at subsequent positions to make room for the new entry by increasing their indices by 1. Increases the list's size by 1. The capacity of the list is increased if that is required. Throws `IndexOutOfBoundsException` if `index` < 0 or `index` > `size()`.

Class `ArrayList`

- Figure 12.1b Methods of class `ArrayList`

`public` *Base_Type* `get(int index)` Returns the element at the position specified by `index`. Throws `IndexOutOfBounds-Exception` if `index` < 0 or `index` ≥ `size()`.
`public` *Base_Type* `set(int index,` *Base_Type* `element)` Replaces the element at the position specified by `index` with the given element. Returns the element that was replaced. Throws `IndexOutOfBoundsException` if `index` < 0 or `index` ≥ `size()`.
`public` *Base_Type* `remove(int index)` Removes and returns the element at the specified index. Shifts elements at subsequent positions toward position `index` by decreasing their indices by 1. Decreases the list's size by 1. Throws `IndexOutOfBoundsException` if `index` < 0 or `index` ≥ `size()`.
`public boolean remove(Object element)` Removes the first occurrence of `element` in this list, and shifts elements at subsequent positions toward the removed element by decreasing their indices by 1. Decreases the list's size by 1. Returns true if `element` was removed; otherwise returns false and does not alter the list.

Creating Instance of `ArrayList`

- Necessary to
  `import java.util.ArrayList;`
- Create and name instance
```
ArrayList<String> list =
  new ArrayList<String>(20);
```
- This list will
  - Hold `String` objects
  - Initially hold up to 20 elements

Using Methods of `ArrayList`

- Object of an ArrayList used like an array
  - But methods must be used
  - Not square bracket notation
- Given

```
ArrayList<String> aList =
    new ArrayList<String>(20);
```

  - Assign a value with

```
aList.add(index, "Hi Mom");
aList.set(index, "Yo Dad");
```

Programming Example

- A To-Do List
  - Maintains a list of everyday tasks
  - User enters as many as desired
  - Program displays the list
- View source code, listing 12.1

`class ArrayListDemo`

Programming Example

```
Enter items for the list, when prompted.
Type an entry:
Buy milk
More items for the list? yes
Type an entry:
Wash car
More items for the list? yes
Type an entry:
Do assignment
More items for the list? no
The list contains:
Buy milk
Wash car
Do assignment
```

Sample screen output

Programming Example

- When accessing all elements of an **`ArrayList`** object
  - Use a For-Each loop
- Use the **`trimToSize`** method to save memory
- To copy an **`ArrayList`**
  - Do not use just an assignment statement
  - Use the **`clone`** method

Parameterized Classes, Generic Data Types

- Class **`ArrayList`** is a *parameterized class*
  - It has a parameter which is a type
- Possible to declare our own classes which use types as parameters
- Note earlier versions of Java had a type of **`ArrayList`** that was not parameterized

Generics: Outline

- The Basics
- Programming Example: A Generic Linked List

Basics of Generics

- Beginning with Java 5.0, class definitions may include parameters for types
  - Called *generics*
- Programmer now can specify any class type for the type parameter
- View class definition, listing 12.2
  `class Sample<T>`
- Note use of `<T>` for the type parameter

Basics of Generics

- Legal to use parameter T almost anywhere you can use class type
  - Cannot use type parameter when allocating memory such as `anArray = new T[20];`
- Example declaration

```
Sample <String> sample1 =
    new Sample<String>();
```

  - Cannot specify a primitive type for the type parameter

Summary

- Java Class Library includes **`ArrayList`**
  - Like an array that can grow in length
  - Includes methods to manipulate the list
- Class can be declared with type parameter
- Object of a parameterized class replaces type parameter with an actual class type