Chapter 12

Dynamic Data Structures and Generics

- 12.1 Array-Based Data Structures
- 12.2 The Java COLLECTIONS Framework
- 12.3 Linked Data Structures
- 12.4 GENERICS

Objectives

- Define and use an instance of `ArrayList`
- Describe general idea of linked list data structures and implementation
- Manipulate linked lists
- Use inner classes in defining linked data structures
- Describe, create, use iterators
- Define, us 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 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

Class `ArrayList`

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

Class `ArrayList`

- Class ArrayList
  - » 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

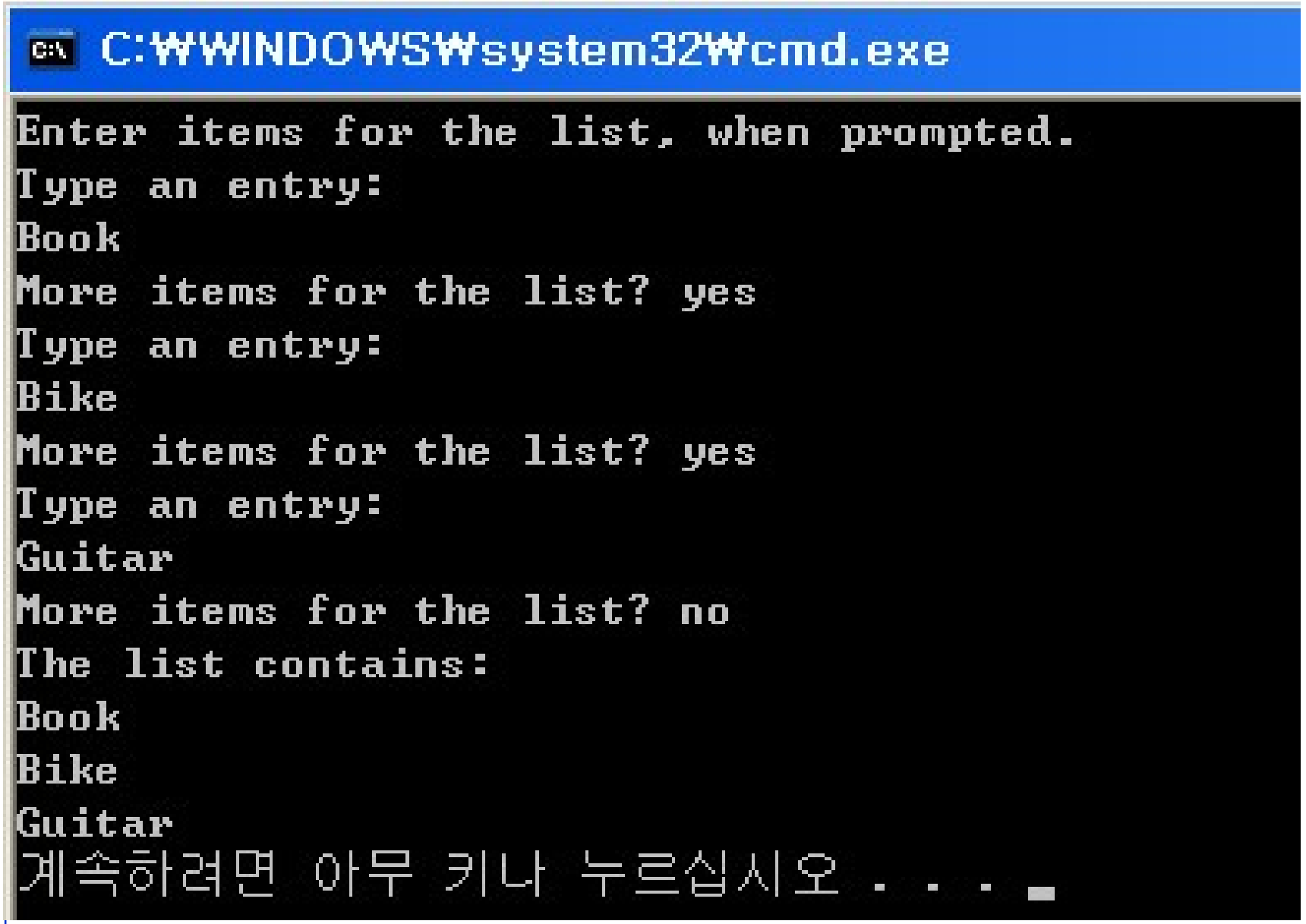
- A To-Do List
  - » Maintains a list of everyday tasks
  - » User enters as many as desired
  - » Program displays the list
- View source code, listing 12.1

```
class ArrayListDemo
```

```
// Listing 12.1

import java.util.ArrayList;
import java.util.Scanner;
public class ArrayListDemo
{
   public static void main (String [] args)
   {
      ArrayList < String > toDoList = new ArrayList < String > ();
      System.out.println ("Enter items for the list, when prompted.");
      boolean done = false;
      Scanner keyboard = new Scanner (System.in);
      while (!done)
      {
         System.out.println ("Type an entry:");
         String entry = keyboard.nextLine ();
         toDoList.add (entry);
         System.out.print ("More items for the list? ");
         String ans = keyboard.nextLine ();
         if (!ans.equalsIgnoreCase ("yes"))
            done = true;
      }
```

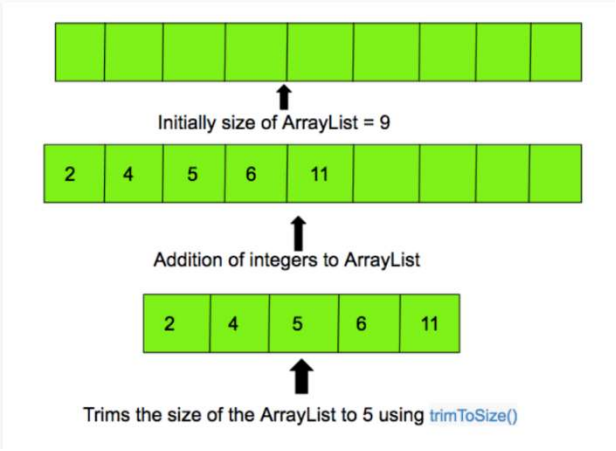
```
        System.out.println ("The list contains:");
        int listSize = toDoList.size ();
        for (int position = 0 ; position < listSize ; position++)
            System.out.println (toDoList.get (position));
    }
}
```


C:₩WINDOWS₩system32₩cmd.exe
Enter items for the list, when prompted.
Type an entry:
Book
More items for the list? yes
Type an entry:
Bike
More items for the list? yes
Type an entry:
Guitar
More items for the list? no
The list contains:
Book
Bike
Guitar
계속하려면 아무 키나 누르십시오 . . .

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



```
import java.util.ArrayList;
import java.util.Scanner;
public class ArrayListDemo
{
   public static void main (String [] args)
   {
      ArrayList < String > toDoList = new ArrayList < String > ();
      System.out.println ("Enter items for the list, when prompted.");
      boolean done = false;
      Scanner keyboard = new Scanner (System.in);
      while (!done)
      {
         System.out.println ("Type an entry:");
         String entry = keyboard.nextLine ();
         toDoList.add (entry);
         System.out.print ("More items for the list? ");
         String ans = keyboard.nextLine ();
         if (!ans.equalsIgnoreCase ("yes"))
            done = true;
      }
```

```
    System.out.println("The list contains:");
    for (String element : toDoList)
        System.out.println(element);    }
}
```

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