CSE 142

Arrays & Implementing List Collections

Outline for Today

- Quick Review – collection classes: ArrayList
- Arrays – a low-level collection-like data structure
- Using arrays to implement higher-level collection classes

What is an ArrayList?

- ArrayList objects are fairly sophisticated
  - Contain 0 or more objects
  - Can add new objects to the collection – makes room if needed
  - Can delete objects from the collection
  - Can find objects in the collection
  - Can iterate through the collection
- How is this implemented?
  - We've already gotten some idea from drawing the pictures…

Arrays

- Java (and many other languages) include ***arrays*** as the most basic kind of collection
  - Simple, ordered collections, similar to ArrayLists
  - Special syntax for declaring values of array type
  - Special syntax for accessing elements by position
- Unlike ArrayLists:
  - The size is fixed when the array is created
  - No iterator; must use explicit indexes and for/while loops
  - Can specify the type of the elements of arrays

Array Example

```
String[ ] pets = new String[3];

pets[0] = “Sally”;
pets[1] = “Puff”;
pets[2] = “Spot”;

pets[1] = "Rex";

String allMyPets = "";
for (int i = 0; i < pets.length; i++) {
      allMyPets = allMyPets + " " + pets[i];
}
```


Array Declaration and Creation

- **Array have special type and new expression syntax:**
  <element type>[ ] *<array name>* = **new** *<element type>* [*<length>*];
- **With an array, we can precisely specify the element type**
  - **Can be any class or interface type, not just Object**
  - **Can also be a primitive type like int, double, boolean, etc.**
- ***<length>* is any integer expression**
  - **Doesn’t need to be a constant**
  - **Value should be greater than 0 (can be 0, but then you get an empty array)**
- **Elements of newly created arrays initialized to null (zero, false)**
- **Arrays have an instance variable, *length*, that stores their length**
  ***<array name>* . length**

Array Element Access

- **Access an array element using the array name and desired position**
  <array name> [*<position>*]
- **Details:**
  - ***<position>* is an integer expression**
  - **Positions count from 0, as with ArrayLists**
  - **Type of result is the element type of the array (not necessarily Object)**
- **Can update an array element by assigning to it**
  <array name> [*<position>*] = *<new element value>* ;
  - **Like ArrayList's set method**

Implementing Containers

- **Example: Implement a simple version of ArrayList to illustrate what’s going on underneath**
- **Specification:**

```
class SimpleList {                          // a list of objects
    SimpleList(int capacity);               // create new SimpleList with given capacity
    int size( );                            // return # of Objects in this SimpleList
    boolean add(Object obj);                // add obj to this SimpleList, result true if success
    boolean contains(Object obj);           // return whether this SimpleList contains obj
    void clear( );                          // remove all objects from this SimpleList
    Object get(int pos);                    // return Object at given position
    Object set(int pos, Object newObj);     // update Object at given position, and
                                            // return previous Object at that position
}
```

  - **For simplicity, we’ll use a fixed maximum capacity.**

SimpleList Representation

- **Underlying representation is an array of objects**
- **Need a separate variable to keep track of how many objects have actually been added to the collection so far (Why?)**

```
/** A collection of Objects */
public class SimpleList {
    // instance variables:
    private Object[ ] objects;          // Objects in this SimpleList are stored in
    private int       numObjects;       //    objects[0] through objects[numObjects-1]
    ...
}
```

  - **Array length gives the *capacity* of the SimpleList container; numObjects is the container's current *size***
  - **Class invariant relates instance variables – essential comment**

SimpleList Constructor

- **Need to allocate the actual array and initialize the SimpleList to "empty"**

```
public class SimpleList {                   // a list of objects
    private Object[ ] objects;              // Objects in this SimpleList are stored in
    private int       numObjects;           //    objects[0] through objects[numObjects-1]
    /** Construct new empty SimpleList with given maximum capacity */
    public SimpleList(int capacity) {




    }
}
```


add

```
/**  Add obj to this SimpleList if there is room.  Return true if added successfully,
 *   otherwise return false. */
public boolean add(Object obj) {




}
```


size

```
/** Return number of elements currently in this SimpleList */
public int size( ) {




}
```

contains

```
/** Return whether this SimpleList contains obj (testing using equals) */
public boolean contains(Object obj) {



}
```


get

```
/** Return the object at position pos in this SimpleList, or null if pos is out of bounds */
public Object get(int pos) {



}
```


set

```
/** update Object at given position to be newStr, and return previous Object at that
* position, or return null if pos is out of bounds */
public Object set(int pos, Object newStr) {



}
```


clear

```
/** Remove all objects from this SimpleList */
public boolean clear( ) {



}
```

Challenges

- **Remove and insert operations**

```
// Remove and return object at given position (shifting all later objects up)
Object remove(int pos) { ... }
// Insert the given object at the given position (shifting all later objects down)
void add(int pos, Object obj) { ... }
```

  - **Requires shifting elements after pos up or down one position**

- **Remove the maximum capacity limitation**
  - **When out of space, allocate a bigger array, copy the current elements over, then replace the old array with the bigger array**

Array Summary

- **Arrays are the fundamental low-level collection type built in to the Java language**
  - **Also found in essentially all interesting programming languages**
- **Size fixed when created**
- **Indexed access to elements**
- **Often used to implement higher-level, richer container types**
  - **More convenient, less error-prone, closer to what users normally want**