Lecture 4: Arrays

Bohyung Han
CSE, POSTECH
bhhan@postech.ac.kr

Array

• A data structure holding a group of variables under a single identifiers

```java
byte[] anArrayOfBytes;
t[] anArrayOfInts;
long[] anArrayOfLongs;
float[] anArrayOfFloats;
double[] anArrayOfDoubles;
boolean[] anArrayOfBooleans;
char[] anArrayOfChars;
String[] anArrayOfStrings;
```

```java
int[] a;
// array declaration without assignment
da = new int[2];
// specify the length of the array
int[] b = {1, 2, 3, 4};
// array declaration with assignment
int bLength = b.length;
// get the length of an array
int c = b[2];
// access an element in array
```

• Two-dimensional arrays
  ▪ Array of arrays
    ```java
    int[][] Y = new int[8][10];
    ```
  ▪ Size of two-dimensional array
    • Number of rows: Y.length
    • Number of elements in each row: Y[i].length
    • Note: Number of elements in each row may be different.

• Java: Array class
  ▪ [http://docs.oracle.com/javase/7/docs/api/java/util.Arrays.html](http://docs.oracle.com/javase/7/docs/api/java/util.Arrays.html)

Abstract Data Type (ADT)

• Definition
  ▪ A mathematical model of the data objects that make up a data type as well as the functions that operate on these objects
  ▪ Defined indirectly, by the operations that may be performed on it and by mathematical constraints on the effects of those operations

• ADT is composed of
  ▪ A data structure
  ▪ A set of operations (called the methods or operations)
  ▪ A precise description of the types of the methods (called a signature)
  ▪ A precise set of rules about how it behaves (called the abstract specification or the axiomatic description)
  ▪ An implementation hidden from the programmer
Array List ADT

• Array list is not array, but an extension of array.
  ▪ An array list stores a sequence of arbitrary objects.
  ▪ An element can be accessed, inserted or removed by specifying its index (number of elements preceding it).
  ▪ An exception is thrown if an incorrect index is given, e.g., negative one.
  ▪ Java: [URL](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html)

Java: Exception

• An event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
• The Java platform provides numerous exception classes.
  ▪ All the classes are descendants of the Throwable class.
  ▪ All allow programs to differentiate among the various types of exceptions.
• Example: if (size == 0) { throw new EmptyStackException(); }

Array-Based Implementation

• Array A of size n

- A
- Size of array list
  ▪ Number of elements in the array
  ▪ Stored in a variable

Operations

• get(i):
  ▪ Returns A[i]
  ▪ Runs in O(1) time
• set(i,o):
  ▪ Performs t = A[i] and A[i] = o, and returns t
  ▪ Runs in O(1) time

Element Insertion

• Operation
  ▪ add(o)
  ▪ Adds an object at the end of the array list, A[n]
  ▪ Increases the size of the array by 1

• Time complexity
  ▪ O(1)
### Element Insertion

- **Operation**
  - `add(i,o)`
  - Makes a room for the new element by shifting forward the `n-1` elements `A[i]`, ..., `A[n-1]`
  - Adds an object at `A[i]`

- **Worst case time complexity**
  - `O(n)` when `i = 0`

![Element Insertion Diagram]

### Element Removal

- **Operation**
  - `remove(i)`
  - Removes an element, `A[i]`
  - Fills the hole left by the removed element by shifting backward the `n-i-1` elements `A[i+1]`, ..., `A[n-1]`

- **Worst case time complexity**
  - `O(n)` when `i = 0`

![Element Removal Diagram]

### Performance

- **Space complexity**
  - The space used by the data structure: `O(n)`

- **Time complexity of methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>size()</code></td>
<td><code>O(1)</code></td>
</tr>
<tr>
<td><code>isEmpty()</code></td>
<td><code>O(1)</code></td>
</tr>
<tr>
<td><code>get(i)</code></td>
<td><code>O(1)</code></td>
</tr>
<tr>
<td><code>set(i,o)</code></td>
<td><code>O(1)</code></td>
</tr>
<tr>
<td><code>add(o)</code></td>
<td><code>O(1)</code></td>
</tr>
<tr>
<td><code>add(i,o)</code></td>
<td><code>O(n)</code> time in worst case</td>
</tr>
<tr>
<td><code>remove(i)</code></td>
<td><code>O(n)</code> time in worst case</td>
</tr>
</tbody>
</table>

### Growable Array-based Array List

- **When the array is full in an add operation**
  - Throwing an exception
  - Replacing the array with a larger one

- **How large should the new array be?**
  - Incremental strategy: increase the size by a constant
  - Doubling strategy: double the size
Growable Array-based Array List

**Algorithm add(o)**

**Input** array A of n integers  
**Output** array A of n+1 integers

if \( t = A.\text{length} \) then

Create a new array S (larger than A)

Copy the elements in A to S

Replace A with S

Increase array size by 1 and add o as the last element

Incremental Strategy

- **How many times is array replaced?**  
  \( k = n/c \) times (c: increment constant)

- The total time \( T(n) \) of a series of \( n \) add operations is proportional to
  \[
  n + \left( c + 2c + 3c + 4c + \cdots + kc \right) = n + c \frac{k(k + 1)}{2}
  \]

- **Time complexity**
  \[
  T(n) = n + c \frac{k(k + 1)}{2} = n + \frac{n(n+1)}{2c} = O(n^2)
  \]
  The amortized time of an add operation is \( O(n) \).

Doubling Strategy

- **How many times is array replaced?**  
  \( k = \log_2 n \) times

- The total time \( T(n) \) of a series of \( n \) add operations is proportional to
  \[
  n \cdot \left( 1 + 2 + 4 + 8 + \cdots + 2^k \right) = 3n - 1
  \]

- **Time complexity**
  \[
  T(n) = O(n)
  \]
  The amortized time of an add operation is \( O(1) \).
Example with Array List

```java
public class ArrayListTest {
    public static void main(String args[]) {
        ArrayList<Integer> IntegerList = new ArrayList<Integer>();
        System.out.println("Empty array? " + IntegerList.isEmpty());
        for (int i=0; i<20; i++)
            IntegerList.add(i);
        System.out.println("Num. of elements: " + IntegerList.size());
        for (int i=1; i<IntegerList.size(); i*=2)
            IntegerList.remove(i);
        System.out.println("Num. of elements: " + IntegerList.size());
        System.out.println("Elements in array");
        for (int i=0; i<IntegerList.size(); i++)
            System.out.println("t" + i + ":" + IntegerList.get(i));
    }
}
```

Simple Implementation of ArrayList

```java
import java.io.*;

public class SimpleArrayList<T> {
    protected int count; // number of elements in ArrayList
    protected T[] contents; // array of stored values
    protected static int DEFAULT_CAPACITY = 10;

    // constructors
    public SimpleArrayList() {
        contents = (T[]) new Object[DEFAULT_CAPACITY];
    }
    public SimpleArrayList(int initCapacity) {
        contents = (T[]) new Object[initCapacity];
    }

    public void add(T newElem) {
        if (count == contents.length) {
            T[] temp[] = contents;
            contents = (T[]) new Object[2*count];
            for (int i=0; i<count; i++)
                contents[i] = temp[i];
            count++;
        }
        contents[count++] = newElem ;
    }

    public T get(int index) {
        if (index < 0 || index >= count)
            throw new IndexOutOfBoundsException("ArrayList:" + index);
        T temp = contents[index];
        for (int j = index; j < count-1; j++)
            contents[j] = contents[j+1];
        count--;
        return temp;
    }

    public int size() {
        return count;
    }

    public T remove (int index) {
        if (index < 0 || index >= count)
            throw new IndexOutOfBoundsException("ArrayList:" + index);
        T temp = contents[index];
        for (int j = index; j < count-1; j++)
            contents[j] = contents[j+1];
        count--;
        return temp;
    }
}
```
Summary

• An ADT consists of
  ▪ A data structure:
    • A collection of data, and
    • its organization, structure, properties;
  ▪ The operations that can be performed:
    • What can be done to change, manipulate, or look at the data

• Information hiding
  ▪ The “user” of an ADT must know enough to:
    • Create one of them, and
    • Perform the operations, Some members, constructors and methods of ArrayList should be known.
  ▪ but not
    • What fundamental construct is used to store the data, and
    • How the operations are done. Detailed implementation of ArrayList class does not need to be known.