



School of Computing

CS1020 Data Structures and Algorithms I

Lecture Note #6

Vector and ArrayList

Objectives

Using the `Vector` and `ArrayList` classes

References



CS1020 website →
Resources → Lectures

- http://www.comp.nus.edu.sg/~cs1020/2_resources/lectures.html

Outline

1. Vector

1.1 Motivation

1.2 API Documentation

1.3 Example

2. ArrayList

2.1 Introduction

2.2 API Documentation

2.3 Example

Drawback of Arrays

- Array, as discussed in week 2, has a major drawback:
 - Once initialized, the array size is **fixed**
 - Reconstruction is required if the array size changes
 - To overcome such limitation, we can use some classes related to array
- Java has an **Array** class
 - Check API documentation and explore it yourself
- However, we will not be using this **Array** class much; we will be using other classes such as **Vector** and **ArrayList**

Vector and ArrayList

- Both provide re-sizable array, i.e. array that is growable
- Both are implementations of the **List** interface
 - We will cover interface later, under Abstract Data Types (ADTs)
- Differences between **Vector** and **ArrayList** are in **slide 15**

1 Vector class

Class for dynamic-size arrays

Motivation

- Java offers a **Vector** class to provide:
 - Dynamic size
 - expands or shrinks automatically
 - Generic
 - allows any reference data types
 - Useful predefined methods
- Use array if the size is fixed; use **Vector** if the size may change.

API documentation (1/3)

Java™ Platform
Standard Ed. 7

All Classes

Packages

- java.applet
- java.awt
- java.awt.color
- java.awt.datatransfer
- java.awt.dnd
- java.awt.event

ValueMember
ValueMemberHelper
ValueOutputStream
VariableElement
VariableHeightLayoutCache
Vector ←

VerifyError
VersionSpecHelper
VetoableChangeListener
VetoableChangeListenerProxy
VetoableChangeSupport
View
ViewFactory
ViewportLayout

Method Summary

Methods

Modifier and Type	Method and Description
boolean	add (E e) Appends the specified element to the end of this vector.
void	add (int index, E element) Inserts the specified element at the specified position in this vector.
boolean	addAll (Collection <? extends E > c) Appends all of the elements in the specified Collection to the end of this vector.
boolean	addAll (int index, Collection c) Inserts all of the elements in the specified Collection into this vector at the specified position.
void	addElement (E obj) Adds the specified component to the end of this vector.
int	capacity () Returns the current capacity of this vector.
void	clear () Removes all of the elements from this vector.

API documentation (2/3)

PACKAGE

```
import java.util.Vector;
```

SYNTAX

```
//Declaration of a Vector reference  
Vector<E> myVector;
```

```
//Initialize a empty Vector object  
myVector = new Vector<E>();
```

Commonly Used Method Summary

boolean*isEmpty()*

Tests if this vector has no components.

int*size()*

Returns the number of components in this vector.

API documentation (3/3)

Commonly Used Method Summary (continued)

<code>boolean</code>	<code>add(E o)</code> Appends the specified element to the end of this Vector.
<code>void</code>	<code>add(int index, E element)</code> Inserts the specified element at the specified position in this Vector.
<code>E</code>	<code>remove(int index)</code> Removes the element at the specified position in this Vector.
<code>boolean</code>	<code>remove(Object o)</code> Removes the first occurrence of the specified element in this Vector If the Vector does not contain the element, it is unchanged.
<code>E</code>	<code>get(int index)</code> Returns the element at the specified position in this Vector.
<code>int</code>	<code>indexOf(Object elem)</code> Searches for the first occurrence of the given argument, testing for equality using the equals method.
<code>boolean</code>	<code>contains(Object elem)</code> Tests if the specified object is a component in this vector.

Example

TestVector.java

```
import java.util.Vector;
public class TestVector {

    public static void main(String[] args)

        Vector<String> courses;

        courses = new Vector<String>();

        courses.add("CS1020");
        courses.add(0, "CS1010");
        courses.add("CS2010");

        System.out.println(courses);
        System.out.println("At index 0: " + courses.get(0));

        if (courses.contains("CS1020"))
            System.out.println("CS1020 is in courses");

        courses.remove("CS1020");
        for (String c: courses)
            System.out.println(c);
    }
}
```

Output:

```
[CS1010, CS1020, CS2010]
At index 0: CS1010
CS1020 is in courses
CS1010
CS2010
```

Vector class has a nice `toString()` method that prints all elements

The enhanced for-loop is applicable to `Vector` objects too!

2 ArrayList class

Another class for dynamic-size arrays

Introduction (1/2)

- Java offers an `ArrayList` class to provide similar features as `Vector`:
 - Dynamic size
 - expands or shrinks automatically
 - Generic
 - allows any reference data types
 - Useful predefined methods
- Similarities:
 - Both are index-based and use an array internally
 - Both maintain insertion order of element
- So, what are the differences between `Vector` and `ArrayList`?
 - This is one of the most frequently asked questions, and at interviews!

Introduction (2/2)

- Differences between [Vector](#) and [ArrayList](#)

Vector	ArrayList
Since JDK 1.0	Since JDK 1.2
Synchronised * (thread-safe)	Not synchronised
Slower (price of synchronisation)	Faster ($\approx 20 - 30\%$)
Expansion: default to double the size of its array (can be set)	Expansion: increases its size by $\approx 50\%$

- [ArrayList](#) is preferred if you do not need synchronisation
 - Java supports multiple threads, and these threads may read from/write to the same variables, objects and resources. Synchronisation is a mechanism to ensure that Java thread can execute an object's synchronised methods one at a time.
- When using [Vector](#) /[ArrayList](#), always try to initialise to the largest capacity that your program will need, since expanding the array is costly.
 - Array expansion: allocate a larger array and copy contents of old array to the new one

API documentation (1/3)

Java™ Platform
Standard Ed. 7

All Classes

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event

ArrayIndexOutOfBoundsException
ArrayList ←
Arrays
ArrayStoreException
ArrayType
ArrayType
AssertionError
AsyncBoxView
AsyncHandler
AsynchronousByteChannel
AsynchronousChannel
AsynchronousChannelGroup
AsynchronousChannelProvider
AsynchronousCloseException
AsynchronousFileChannel

Method Summary

Methods

Modifier and Type	Method and Description
boolean	add (E e) Appends the specified element to the e
void	add (int index, E element) Inserts the specified element at the spe
boolean	addAll (Collection <? ext Appends all of the elements in the spec iterator.
boolean	addAll (int index, Colle Inserts all of the elements in the specifi
void	clear () Removes all of the elements from this
Object	clone () Returns a shallow copy of this Array
boolean	contains (Object o) Returns true if this list contains the s
void	ensureCapacity (int minC

API documentation (2/3)

PACKAGE	<pre>import java.util.ArrayList;</pre>
SYNTAX	<pre>//Declaration of a ArrayList reference ArrayList<E> myArrayList; //Initialize a empty ArrayList object myArrayList = new ArrayList<E>();</pre>

Commonly Used Method Summary

boolean	<i>isEmpty()</i> Returns true if this list contains no element.
int	<i>size()</i> Returns the number of elements in this list.

API documentation (3/3)

Commonly Used Method Summary (continued)	
<code>boolean</code>	<p><i>add</i>(E e)</p> <p>Appends the specified element to the end of this list.</p>
<code>void</code>	<p><i>add</i>(<code>int</code> index, E element)</p> <p>Inserts the specified element at the specified position in this list.</p>
E	<p><i>remove</i>(<code>int</code> index)</p> <p>Removes the element at the specified position in this list.</p>
<code>boolean</code>	<p><i>remove</i>(Object o)</p> <p>Removes the first occurrence of the specified element from this list, if it is present.</p>
E	<p><i>get</i>(<code>int</code> index)</p> <p>Returns the element at the specified position in this list.</p>
<code>int</code>	<p><i>indexOf</i>(Object o)</p> <p>Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.</p>
<code>boolean</code>	<p><i>contains</i>(Object elem)</p> <p>Returns <code>true</code> if this list contains the specified element.</p>

Example

TestArrayList.java

```
import java.util.ArrayList;
import java.util.Scanner;

public class TestArrayList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<Integer> list = new ArrayList<Integer>();

        System.out.println("Enter a list of integers, press ctrl-d to end.");
        while (sc.hasNext()) {
            list.add(sc.nextInt());
        }

        System.out.println(list); // us

        // Move first value to last
        list.add(list.remove(0));

        System.out.println(list);
    }
}
```

Output:

Enter a list ... to end.

31

17

-5

26

50

(user pressed ctrl-d here)

[31, 17, -5, 26, 50]

[17, -5, 26, 50, 31]

Practice Exercises

- A bumper crop of practice exercises (exercises 15 – 21) are mounted on CodeCrunch this week
- The files are also available on the CS1020 website:
http://www.comp.nus.edu.sg/~cs1020/4_misc/practice.html
- You are urged to work on these exercise as they are important for you to cement your basic understanding of the topics that are covered so far (OOP and arrays)

Practice Exercises

Vector and ArrayList

- #15: Missing Digits version 2
 - Using [Vector](#)
- #16: Set Containment
 - Using [ArrayList](#) and writing your own class
- #17: Nearest Points
 - Using [ArrayList](#) and [Point](#)

OOP

- #18: Overlapping Rectangles Version 2
- #19: Overlapping Rectangles Version 3
- #20: Redeem Coupon

OOP and ArrayList

- #21: Turning Knobs

Detecting Duplicates (1/4)

- Using `ArrayList` class and random number generation.
 - You may use the `Math.random()` method or the `Random` class
- Write a program `DetectDuplicates.java` to read the following values:
 - The number of unique random integers to generate; and
 - Limit of the values: each random number generated should be in the range from 0 (inclusive) to limit (exclusive), or $[0, \text{limit} - 1]$.
 - (Certainly, the second input value must not be smaller than the first)
- Each time a random integer is generated, you must check if it is a duplicate of an earlier generated value. If it is, it must be discarded. The program goes on to generate the required number of unique random integers.
- You are to count how many duplicates were detected.

Detecting Duplicates (2/4)

- Sample run
 - (In testing your code, each time a random number is generated, you may want to print it to check that the computation is correct)

```
Enter number of unique integers to generate: 10
Enter limit: 20
List: [16, 3, 15, 17, 2, 10, 18, 5, 12, 14]
Duplicates detected: 8
```

Detecting Duplicates (3/4)

DetectDuplicates.java

```
import java.util.*;

public class DetectDuplicates {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        ArrayList<Integer> list = new ArrayList<Integer>();

        System.out.print("Enter number of unique ...: ");
        int numUnique = sc.nextInt();

        System.out.print("Enter limit: ");
        int limit = sc.nextInt();

        Random rnd = new Random();
        int countUnique = 0;
        int countDuplicates = 0;
        int num; // the random number
```


Detecting Duplicates (4/4)

DetectDuplicates.java

```
System.out.println("List: " + list);  
System.out.println("Duplicates detected: "  
                    + countDuplicates);  
}  
}
```

End of file
