

AP CSA

zhang si 张思

zhangsi@rdfz.cn

ICC 609

ArrayList (2.5-7.5%)

- The ArrayList object has a dynamic size, and the class contains methods for insertion and deletion of elements making reordering and shifting items easier.
- this Chapter focuses on:
 - Introduction to `ArrayList`
 - the `ArrayList` method
 - traversing the `ArrayList`
 - Developing Algorithms Using `ArrayLists`

Preparing for the AP Exam

- When writing solutions to free-response questions that involve the use of an ArrayList, students are often asked to insert or delete elements from the ArrayList.
 - In these cases, adjustments will need to be made to the loop counter to account for skipping an element or attempting to access elements that no longer exist.
- Students may also be asked to traverse multiple data structures simultaneously.

An ArrayList is often called just a list on the CS A exam. In past AP CS A exams, the interface List is often used to declare an ArrayList. Interfaces are no longer on the exam, but if you see List being used, just assume it's an ArrayList.

Intro to ArrayList

- In the last unit, we learned about arrays to hold collections of related data. But arrays have limitations.
 - The size of an array is established at the time of creation and cannot be changed. What if you don't know how big the collection of data will be?
- What if you want to add and remove items from the collection and change the size of the collection while the program is running?
 - For example, if you wanted to represent a shopping list, you might add to the list throughout the week and remove things from the list while you are shopping. You probably would not know how many items will be on the list at the beginning of the week.
- Java has a class called **ArrayList** which is a re-sizable array. An ArrayList has an underlying array that grows or shrinks as needed.
 - ArrayList is mutable, meaning it can change during runtime by adding and removing objects from it.

vocabulary

Arrays are **static** in size, once initialized, their size cannot be changed.

`ArrayLists` are **dynamic** in size, the size of the list can be changed at any time.

Difference between Array and ArrayList

Array

Fixed length
Fundamental Java feature
An Object with no methods
Not as flexible
Can store primitive data

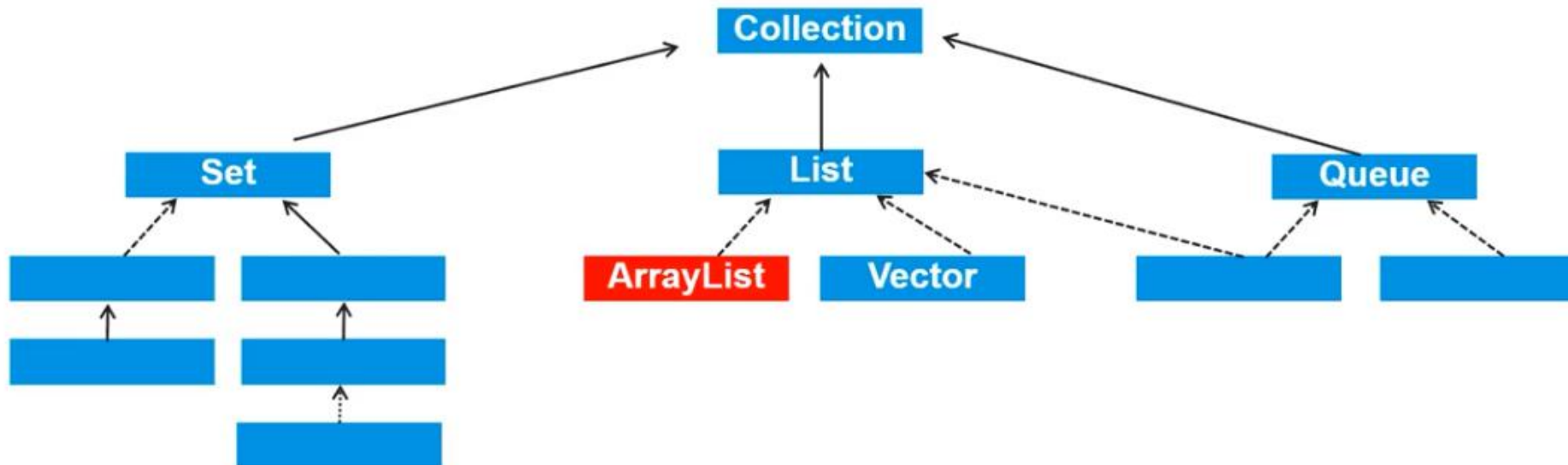
ArrayList

Resizable length
Part of a Framework
A Class with many methods
Is designed to be flexible
Not designed to store primitives
Is slightly slower than Arrays
Can only be used with an import statement

- The ArrayList Class is implemented using Arrays.

Collections Framework

Frameworks are prewritten, high-performing and efficient code that can handle and use objects of groups of data.



The ArrayList Class

- The `ArrayList` class is part of the `java.util` package
- Like an array, it can store a list of values and reference them with an index
- Unlike an array, an `ArrayList` object grows and shrinks as needed
- Items can be inserted or removed with a single method invocation
- It stores references to the `Object` class, which allows it to store any kind of object
- See [DestinysChild.java](#)

Declaring and Creating ArrayLists

- To declare a ArrayList use `ArrayList<Type> name` Change the Type to be whatever type of objects you want to store in the ArrayList, for example `String` as shown in the code below. You don't have to specify the `generic type <Type>`, since it will default to `Object`, but it is good practice to specify it to restrict what to allow in your ArrayList.
- Using a type `ArrayList<Type>` is preferred over just using `ArrayList` because it allows the compiler to find errors that would otherwise be missed until run-time.
- EG:

```
// ArrayList<Type> name = new ArrayList<Type>();
```

```
// An ArrayList of Strings:
```

```
ArrayList<String> shoppingList = new ArrayList<String>();
```

Declaring and Creating ArrayLists

- You can also create ArrayLists of integer values.
- However, you have to use Integer as the type because ArrayLists can only hold objects, not primitive values.
- All primitive types must be wrapped in objects before they are added to an ArrayList.
- For example, int values can be **wrapped** in Integer objects, double values can be wrapped in Double objects.
- You can actually put in any kind of Objects in an ArrayList, even for a class that you wrote in Unit 5 like Student or Bandbooster or Name.

Primitive Values Disguised as `Wrapper` Class Objects

`ArrayList` objects are designed to only store references to objects, not primitive values. A workaround is to use `Wrapper` classes, which store primitive values as objects.

<u>Primitive Data Types</u>	<u>Wrapper Class Data Types</u>
<code>boolean</code>	<code>Boolean</code>
<code>char</code>	<code>Character</code>
<code>double</code>	<code>Double</code>
<code>int</code>	<code>Integer</code>

Note

`ArrayLists` can only hold objects like `String` and the wrapper classes `Integer` and `Double`. They cannot hold primitive types like `int`, `double`, etc.

Examples

```
import java.util.ArrayList;
public class Video7Point1
{
    public static void main(String[] args)
    {
        ArrayList<Integer> a1 = new ArrayList<Integer>();
        ArrayList<String> a2 = new ArrayList<String>(5);
        ArrayList<Student> a3 = new ArrayList<Student>();
    }
}
```

Introduction to ArrayList

- ESSENTIAL KNOWLEDGE
- An ArrayList object is mutable and contains object references.
- The ArrayList constructor ArrayList() constructs an empty list.
- Java allows the generic type ArrayList<E>, where the generic type E specifies the type of the elements.
- When ArrayList<E> is specified, the types of the reference parameters and return type when using the methods are type E.
- ArrayList<E> is preferred over ArrayList because it allows the compiler to find errors that would otherwise be found at run-time.

ArrayList Methods you need to know

- size()
- add
- add
- get
- set
- remove
- int size()
- boolean add(E obj)
- void add(int index, E obj)
- E get(int index)
- E set(int index, E obj)
- E remove(int index)

ArrayList Methods you need to know

- `int size()` – Returns the number of elements in the list
- `boolean add(E obj)` – Appends `obj` to end of list; returns `true`
- `void add(int index, E obj)` – Inserts `obj` at position `index` ($0 \leq \text{index} \leq \text{size}$), moving elements at position `index` and higher to the right (adds 1 to their indices) and adds 1 to size
- `E get(int index)` – Returns the element at position `index` in the list
- `E set(int index, E obj)` – Replaces the element at position `index` with `obj`; returns the element formerly at position `index`

- `E remove(int index)` – Removes element from position `index`, moving elements at position `index + 1` and higher to the left (subtracts 1 from their indices) and subtracts 1 from size; returns the element formerly at position `index`

ArrayList Methods

Size of the ArrayList

`int size()` : Returns the number of elements in the list

Consider the following code:

```
ArrayList<Integer> a1 = new ArrayList<Integer>();
```

The ArrayList a1 has been instantiated with no entries.

```
System.out.println(a1.size());
```



result is : 0

```
ArrayList<Double> a2 = new ArrayList<Double>(15);
```

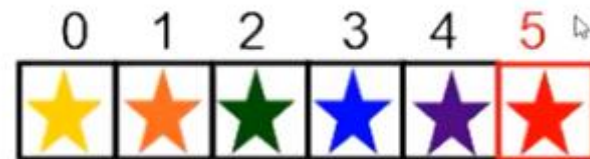

ArrayList Methods

Adding Items to an ArrayList

`boolean add(E obj)` : Appends `obj` to end of list; returns `true`.



`stars.add(★)`



`void add(int index, E obj)` : Inserts `obj` at position `index` ($0 \leq \text{index} \leq \text{size}$), moving elements at position `index` and higher to the right (adds 1 to their indices) and adds 1 to list size.



`stars.add(3, ★)`



Adding Items to an ArrayList

Consider the following code:

```
ArrayList<String> h = new ArrayList<String>(); // line 10
```

```
h.add("Hello"); // line 12
```

```
h.add("Hello"); // line 13
```

```
h.add("HELLO"); // line 14
```

```
h.add("hello"); // line 15
```

```
h.add(1, "Hola");// line 16
```

Hello				
Hello	Hello			
Hello	Hello	HELLO		
Hello	Hello	HELLO	Hello	
Hello	Hola	Hello	HELLO	Hello

**To add objects to an ArrayList, the objects must be of the SAME data type used to instantiate the ArrayList.*

```
h.add(26.2)
```



```
h.add(new String("HeLlO"));
```

```
h.add(false);
```



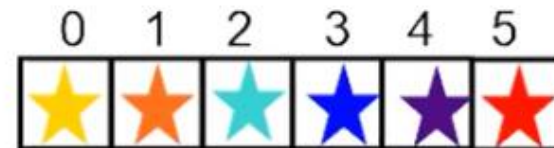
ArrayList Methods

Deleting Items from an ArrayList

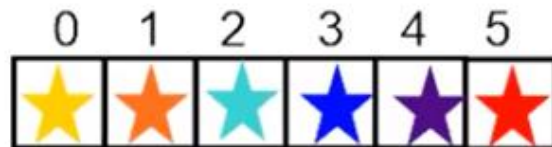
`remove(int index)` : Removes element from position `index`, moves elements at position `index + 1` and higher to the left (subtracts 1 from their indices) and subtracts 1 from size; returns the element formerly at position `index`.



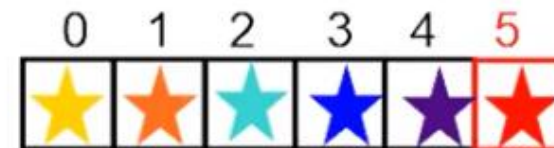
```
Star s1 = stars.remove( 2 );
```



```
System.out.print(s1)
```



```
stars.remove(stars.size() - 1)
```



Deleting Items from an ArrayList

Consider the following code:

0	1	2	3	4
Hello	Hola	Hello	HELLO	Hello

```
h.remove(3); // line 20
```

Hello	Hola	Hello	Hello
-------	------	-------	-------

```
String h1 = h.remove(0); // line 21
```

Hola	Hello	Hello
------	-------	-------


```
System.out.println(h1); // will print out "Hello"
```


ArrayList Methods

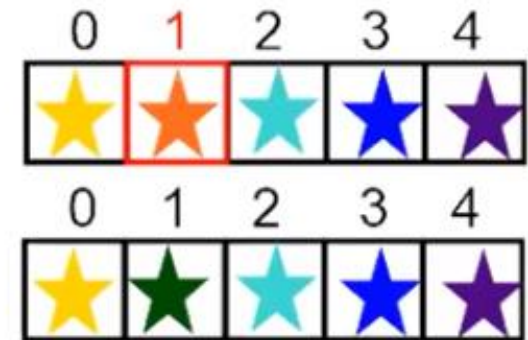
Updating Items in an ArrayList

`E set(int index, E obj)` : Replaces the element at position `index` with `obj`; returns the element formerly at position `index`.



```
Star s1 = stars.set( 1,  );
```

```
System.out.print(s1) 
```



Updating Items in an ArrayList

Consider the following code:

0	1	2
Hola	Hello	Hello

```
h.set(1, "Hola"); // line 30
```

Hola	Hola	Hello
------	------	-------

```
String str = h.set(0, Bonjour); // line 31
```

Bonjour	Hola	Hello
---------	------	-------

```
System.out.println(str); // will print out "Hola"
```

ArrayList Methods

Accessing Items in an ArrayList


`E get(int index)` : Returns the element at position `index` in the list.

`ArrayList<Star> stars` →

0	1	2	3	4
				

`Star s1 = stars.get(1);` `s1` → 

`Star s2 = stars.get(0);` `s2` → 

`Star s2 = stars.get(stars.size()-1);` 

Methods with an `ArrayList` as a Parameter

`ArrayList` is a reference object and, when passed as a parameter in a method, they are passed as references to their addresses, not copies of their values.

```
ArrayList arr = new ArrayList();
```



When a method updates elements of a passed `ArrayList`, the `ArrayList`'s elements are updated automatically.

Methods with an ArrayList as a Parameter

If a method with an `ArrayList` as a parameter only involves accessing elements of the `ArrayList` without assigning or updating the elements from the `ArrayList`, then the `myMethod` method signature below is sufficient.

```
import java.util.ArrayList;
public class Video7P2
{
    public static void main(String[] args)
    {
        ArrayList<Boolean> questions = new ArrayList<Boolean>();
        questions.add(true);
        myMethod(questions);
    }
    public static void myMethod(ArrayList arr)
    {
        if (arr.size() > 0)
        {
            System.out.println(arr.get(0));
        }
    }
}
```

true

If the method involves adding, updating, or storing elements of the passed `ArrayList`, then the `myMethod1` signature below is insufficient and could cause problems.

```
import java.util.ArrayList;
public class Video7P2
{
    public static void main(String[] args)
    {
        ArrayList<Boolean> questions = new ArrayList<Boolean>();
        questions.add(true);
        myMethod1(questions);
        System.out.println(questions.get(0));
    }
    public static void myMethod1(ArrayList arr)
    {
        if (arr.size() > 0)
        {
            arr.set(0, "Hello");
        }
    }
}
```



Hello

Although our main method is passing a **Boolean** `ArrayList`, the compiler warns you that `myMethod1` might not be safe. To prevent these issues, specify the data type of the elements stored in the `ArrayList`.



•VAR-2.D.5

`ArrayList<E>` is preferred over `ArrayList` because it allows the compiler to find errors that would otherwise be found at run-time.

If the method involves adding, updating, or storing elements of the passed `ArrayList`, then the `myMethod1` signature below is insufficient and could cause problems.

```
import java.util.ArrayList;
public class Video7P2
{
    public static void main(String[] args)
    {
        ArrayList<Boolean> questions = new ArrayList<Boolean>();
        questions.add(true);
        myMethod1(questions);
        System.out.println(questions.get(0));
    }
    public static void myMethod1(ArrayList<Boolean> arr)
    {
        if (arr.size() > 0)
        {
            arr.set(0, "Hello");
        }
    }
}
```

When you specify the data type of the elements stored in the `ArrayList`, the compiler is more helpful in describing the errors.

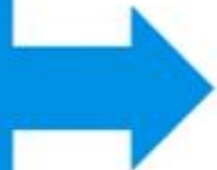
```
Video7P2.java:7: error: incompatible types:
ArrayList<String> cannot be converted to
ArrayList<Boolean>
    myMethod(questions);
    ^
Video7P2.java:12: error: no suitable method found for
add(String)
    arr.add("Hello");
    ^
    method Collection.add(Boolean) is not applicable
        (argument mismatch; String cannot be converted
        to Boolean)
    method List.add(Boolean) is not applicable
        (argument mismatch; String cannot be converted
        to Boolean)
    method AbstractCollection.add(Boolean) is not
    applicable
        (argument mismatch; String cannot be converted
        to Boolean)
    method AbstractList.add(Boolean) is not
    applicable
        (argument mismatch; String cannot be converted
        to Boolean)
    method ArrayList.add(Boolean) is not applicable
        (argument mismatch; String cannot be converted
        to Boolean)
Note: Some messages have been simplified; recompile
with -Xdiags:verbose to get full output
2 errors
```

Returning an ArrayList

In order to return an `ArrayList`, it is preferred that you specify the data type of the elements that the `ArrayList` stores.

```
public static ArrayList<String> methodName()  
{  
    ArrayList<String> arr;  
    arr.add("Hello");  
    return arr;  
}
```

**No longer
problematic**



Hello

ArrayList Methods

- ESSENTIAL KNOWLEDGE

- The ArrayList class is part of the java.util package. An import statement can be used to make this class available for use in the program.
- The following ArrayList methods—including what they do and when they are used—are part of the Java Quick Reference:
 - `int size()`- Returns the number of elements in the list
 - `boolean add(E obj)`-Appends obj to end of list; returns true
 - `void add(int index, E obj)`-Inserts obj at position index ($0 \leq \text{index} \leq \text{size}$), moving elements at position index and higher to the right (adds 1 to their indices) and adds 1 to size
 - `E get(int index)`-Returns the element at position index in the list
 - `E set(int index, E obj)`— Replaces the element at position index with obj;returns the element formerly at position index
 - `E remove(int index)`—Removes element from position index, moving elements at position index + 1 and higher to the left (subtracts 1 from their indices) and subtracts 1 from size; returns the element formerly at position index

ArrayList: other stuff to know


- Enhanced for loop still works (same as array). You cannot add or remove items during enhanced for loop. (It's blocked.)
- When adding or deleting elements and running a loop, be careful not to skip elements or count them twice.

Possible loop Conditions to Traverse an **ArrayList**

	Initialization	Boolean Condition	Update
a)	<code>int i = 0;</code>	<code>i < arr.size();</code>	<code>i++;</code>
b)	<code>int i = 0;</code>	<code>i <= arr.size()-1;</code>	<code>i++;</code>
c)	<code>int i = arr.size()-1;</code>	<code>i >= 0;</code>	<code>i--;</code>
d)	<code>int i = arr.size()-1;</code>	<code>i > -1;</code>	<code>i--;</code>

Traversing: Accessing All the Elements

Suppose we have an **ArrayList** of Strings named **roster**, and we want to know the total number of characters in all of the Strings. In order to do this, we will need to visit **each** entry of the **roster ArrayList**.

```
int sum = 0;
for (int i=0; i <= roster.size()-1; i++)
{
    sum = sum + 
}
System.out.println( sum );
```


Traversing: Accessing All the Elements

Suppose we have an `ArrayList` of Strings named `roster`, and we want to know the total number of characters in all of the Strings. In order to do this, we will need to visit **each** entry of the `roster ArrayList`.

```
int sum = 0;
for (int i=0; i <= roster.size()-1; i++)
{
    sum = sum + roster.get(i).length();
}
System.out.println( sum );
```

Retrieving the entry of
roster at the i^{th} index

Retrieving the number of
characters at the i^{th} index
of **roster**


Traversing: Accessing All the Elements

Suppose we have an `ArrayList` of Strings named `roster`, and we want to know the total number of characters in all of the Strings. In order to do this, we will need to visit **each** entry of the `roster ArrayList`.

```
int sum = 0;
int i = 0;
while ( i < roster.size() )
{
    sum = sum + roster.get(i).length();
    i++;
}
System.out.println( sum );
```

Traversing: Accessing All the Elements

Suppose we have an `ArrayList` of Strings named `roster`, and we want to know the total number of characters in all of the Strings. In order to do this, we will need to visit **each** entry of the `roster ArrayList`.



```
int sum = 0;
for (String name: roster)
{
    
}
System.out.println(sum);
```

Traversing: Accessing All the Elements


Suppose we have an `ArrayList` of Doubles named `grades`, and we want to remove the entries that are lower than 70.0. We will need to visit **each** entry of the `grades ArrayList`.

```
for (int i = 0 ; i < grades.size() ; i++)  
{  
    if (grades.get(i) < 70.0)  
    {  
        grades.remove(i);  
    }  
}
```



```
for (int i = 0; i < grades.size() ; i++)  
{  i=3  
  if (grades.get(i) < 70.0)  
  {  
 i=2      grades.remove(i);  
  }  
}
```

 i=2 **removes 60**



0	100
1	100
2	60
3	50
4	80
5	70





0	100
1	100
2	50
3	80
4	70

Traversing: Accessing All the Elements

Suppose we have an `ArrayList` of Doubles named `grades`, and we want to remove the entries that are lower than 70.0. We will need to visit **each** entry of the `grades ArrayList`.

```
for (int i = grades.size()-1 ; i >= 0 ; i--)  
{  
    if (grades.get(i) < 70.0)  
    {  
        grades.remove(i);  
    }  
}
```



```
for (int i = 0; i < grades.size(); i--)  
{ i=2   
  if (grades.get(i) < 70.0)  
  {  
    grades.remove(i); i=3   
  }  
}
```

0	100
1	100
2	60
3	50
4	80
5	70

i=2 

0	100
1	100
2	60
3	80
4	70

removes 50 removes 60

0	100
1	100
2	80
3	70

Enhanced For Loop for an ArrayList

```
for (DataType item : nameOfArrayList )  
{  
    // statement one;  
    // statement two;  
    // ...  
}
```


Enhanced For Loop Example

Suppose `roster` is an `ArrayList` of `Strings` of names of students and we want to determine the total number of characters needed to write out the roster of student names.

```
int sum = 0;
for (String name : roster )
{
    sum = sum + name.length();
}
```

Print Certain Elements of an ArrayList

Let `grades` be an `ArrayList` of `Doubles`. Use an enhanced `for` loop to print out the grades that are higher than 70.0.

```
for (Double score : grades )  
{  
    if (score > 70.0)  
    {  
        System.out.println(score) ;  
    }  
}
```

```

/** Code written by K. Ayodeji
 * For 7.3 Video 2 to demonstrate ArrayList limitations
 */

import java.util.ArrayList;
public class Video7P3
{
    public static void main(String[] args)
    {
        ArrayList<Double> grades = new ArrayList<Double>();
        grades.add(81.3); grades.add(97.5); grades.add(65.8);
        grades.add(100.0); grades.add(90.0); grades.add(44.3);

        System.out.println("Original grades:");
        for( double score : grades)
        {
            System.out.print(score + " ");
        }

        System.out.println("\nUpdating grades...");
        for (double score : grades)
        {
            if (score < 70)
            {
                score = 70;
            }
            System.out.print(score + " ");
        }

        System.out.println("\nVerification of grades:");
        for( double score : grades)
        {
            System.out.print(score + " ");
        }
    }
}

```

Original grades:

81.3 97.5 65.8 100.0 90.0 44.3

Updating grades...

81.3 97.5 70.0 100.0 90.0 70.0

Verification of grades:

81.3 97.5 65.8 100.0 90.0 44.3

Reminders

1. No index, not set() or get()
2. Enhanced **for** loops make a copy of each entry.

```

/** Code written by K. Ayodeji
 * For 7.3 Video 2 to demonstrate ArrayList limitations
 */
import java.util.ArrayList;
public class Video7P3
{
    public static void main(String[] args)
    {
        ArrayList<Double> grades = new ArrayList<Double>();
        grades.add(81.3); grades.add(97.5); grades.add(65.8);
        grades.add(100.0); grades.add(90.0); grades.add(44.3);

        System.out.println("Original grades:");
        for( double score : grades)
        {
            System.out.print(score + " ");
        }

        System.out.println("\nUpdating grades...");
        for (double score : grades)
        {
            if (score < 70)
            {
                grades.add(130.0);
            }
            System.out.print(score + " ");
        }

        System.out.println("\nVerification of grades:");
        for( double score : grades)
        {
            System.out.print(score + " ");
        }
    }
}

```

Reminder:
Trying to add entries
in an enhanced for
loop will cause an
Exception.

```

Original grades:
81.3 97.5 65.8 100.0 90.0 44.3
Updating grades...
81.3 97.5 65.8 Exception in thread "main"
java.util.ConcurrentModificationException
    at
    java.util.ArrayList$Itr.checkForComodificatio
n(ArrayList.java:909)
    at
    java.util.ArrayList$Itr.next(ArrayList.java:8
59)
    at Video7P3.main(Video7P3.java:17)

```

Warning

During the iterations of an enhanced `for` loop, `ArrayList` elements cannot be modified, removed from, or added to the `ArrayList`.

Common Mistakes

- Forgetting to include `import java.util.ArrayList`
- Declaring and/or instantiating a `ArrayList` with a primitive data type

```
ArrayList<int> myList = new ArrayList<int>();
```

- Forgetting to include the `()` at the end of the `ArrayList` constructor call

```
ArrayList<Integer> myList = new ArrayList<Integer>?
```

- Not specifying the element type that the `ArrayList` references

```
ArrayList myList = new ArrayList();
```


Common Mistakes

- Trying to update **ArrayList** values while using an enhanced **for** loop

```
for (double score : grades)
{
    if (score < 70)
    {
        score = 70;
    }
}
```

```
Original grades:
81.3 97.5 65.8 100.0 90.0 44.3
Updating grades...
81.3 97.5 70.0 100.0 90.0 70.0
Verification of grades:
81.3 97.5 65.8 100.0 90.0 44.3
```

- Changing the size of an **ArrayList** while traversing with an enhanced **for** loop


```
for (double score : grades)
{
    if (score < 70)
    {
        grades.add(130.0);
    }
}
```

```
Original grades:
81.3 97.5 65.8 100.0 90.0 44.3
Updating grades...
81.3 97.5 65.8 Exception in thread "main"
java.util.ConcurrentModificationException
    at
    java.util.ArrayList$Itr.checkForComodification(ArrayList.java:909)
    at java.util.ArrayList$Itr.next(ArrayList.java:859)
    at Video7P3.main(Video7P3.java:17)
```

- Removing an element from an **ArrayList** at the wrong time

```
for (int i = 0; i < grades.size() ; i++)
{
    if (grades.get(i) < 70.0)
    {
        grades.remove(i);
    }
}
```

0	100
1	100
2	60
3	50
4	80
5	70



0	100
1	100
2	50
3	80
4	70

Confusing **Array** With **ArrayList**

```
ArrayList<Character> letters = new ArrayList<Character>(15);  
letters[2] = new Character( 'c' );  
int n = letters.length();  
letters[15] = 'c';
```



Can you find the error(s)?

- `[]` notation—instead use `letters.set(2, ...)` or `letters.add(...)`
- `.length`—instead use `.size()`
- `IndexOutOfBoundsException`—If you want to add another element, use `letters.add(15, 'c');`

Traversing ArrayLists

- ESSENTIAL KNOWLEDGE
- Iteration statements can be used to access all the elements in an ArrayList. This is called traversing the ArrayList.
- Deleting elements during a traversal of an ArrayList requires using special techniques to avoid skipping elements.
- Since the indices for an ArrayList start at 0 and end at the number of elements – 1, accessing an index value outside of this range will result in an `IndexOutOfBoundsException` being thrown.
- Changing the size of an ArrayList while traversing it using an enhanced for loop can result in a `ConcurrentModificationException` being thrown. Therefore, when using an enhanced for loop to traverse an ArrayList, you should not add or remove elements.

ArrayList Efficiency

- The `ArrayList` class is implemented using an array
- The code of the `ArrayList` class automatically expands the array's capacity to accommodate additional elements
- The array is manipulated so that indexes remain continuous as elements are added or removed
- If elements are added to and removed from the end of the list, this processing is fairly efficient
- If elements are inserted and removed from the middle of the list, the elements are constantly being shifted around

Developing Algorithms Using ArrayLists

- ESSENTIAL KNOWLEDGE
- There are standard ArrayList algorithms that utilize traversals to:
 - Insert elements
 - Delete elements
 - Apply the same standard algorithms that are used with 1D arrays
- Some algorithms require multiple String, array, or ArrayList objects to be traversed simultaneously

- Finding the Maximum value in ArrayList<Double>

```
// using an array of doubles
private double findMax(double[ ] values)
{
    double max = values[0];

    for (int index = 1; index < values.length; index++)
    {
        if (values[index] > max)
        {
            max = values[index];
        }
    }
    return max;
}
```

```
// using an ArrayList of Doubles
private double findMax(ArrayList<Double> values)
{
    double max = values.get(0);

    for (int index = 1; index < values.size(); index++)
    {
        if (values.get(index) > max)
        {
            max = values.get(index);
        }
    }
    return max;
}
```

● Finding the Minimum value in ArrayList<Integer>

```
// using an array of ints
private int findMin(int[ ] values)
{
    int min = Integer.MAX_VALUE;

    for (int currentValue : values)
    {
        if (currentValue < min)
        {
            min = currentValue;
        }
    }
    return min;
}
```

```
// using an ArrayList of Doubles
private int findMin(ArrayList<Integer> values)
{
    int min = Integer.MAX_VALUE;

    for (int currentValue : values)
    {
        if (currentValue < min)
        {
            min = currentValue;
        }
    }
    return min;
}
```