STUDENT LESSON

ArrayList

**INTRODUCTION:** It is very common for a program to manipulate data that is kept in a list. Lists are a fundamental feature of Java and most programming languages. Because lists are so useful, the Java Development Kit includes the ArrayList class. The ArrayList class provides the classic operations for a list.

The key topics for this lesson are:

A. ArrayList Implementation of a List

B. The ArrayList Class

C. Object Casts

D. The Wrapper Classes

E. Iterator

**VOCABULARY:** ABSTRACT DATA TYPE ArrayList

CAST for each LOOP  
LISTWRAPPER

**DISCUSSION:** A. ArrayList Implementation of a List

1. A data structure combines data organization with methods of accessing and manipulating the data. For example, an ArrayList is a data structure for storing a list of elements and provides methods to find, insert, and remove an element. At a very abstract level, we can describe a general “list” object. A list contains a number of elements arranged in sequence. We can find a target value in a list, add elements to the list, remove elements from the list and process each element of the list.

2. An abstract description of a data structure, with the emphasis on its properties, functionality, and use, rather than on a particular implementation, is referred to as an *Abstract Data Type* (ADT). An ADT defines methods for handling an abstract data organization without the details of implementation.

3. A “list” ADT, for example, may be described as follows:

Data organization:

– Contains a number of data elements arranged in a linear sequence

Methods:

– Create an empty List

– Append an element to List

– Remove the i-th element from List

– Obtain the value of the i-th element

– Traverse List (process or print out all elements in sequence, visiting each element once)

4. An ArrayList object contains an array of object references plus many methods for managing that array. The most convenient feature of an ArrayList is that you can keep adding elements to it no matter what size it was originally. The size of the ArrayList will automatically increase and no information will be lost.

B. The ArrayList Class

1. a. To declare a reference variable for an ArrayList, do this:

// myArrayList is a reference to

// a future ArrayList object

ArrayList <ClassName> myArrayList;

An ArrayList is an array of references to objects of type ClassName, where ClassName can be any class defined by you or Java.   
  
b. To declare a variable and to construct an ArrayList with an unspecified initial capacity, do this:

// myArrayList is a reference to an ArrayList

// object. The Java system picks the initial

// capacity.

ArrayList <ClassName> myArrayList =

**new** ArrayList <ClassName> ();

This may not be very efficient. If you have an idea of what size ArrayList you need, start your ArrayList with that capacity.   
  
c. To declare a variable and to construct an ArrayList with an initial capacity of 15, do this:

// myArrayList is a reference to an ArrayList

// object with an initial capacity of 15 elements.

ArrayList <ClassName> myArrayList =

**new** ArrayList <ClassName> (15);

1. One way of accessing the elements of an ArrayList is by using an integer index. The index is an integer value that starts at 0 and goes to size()-1.

To access the object at a particular index, use:

// Returns the value of the element at index

Object get(**int** index);

System.out.println(myArrayList.get(i));

3. To add an element to the end of an ArrayList, use:

// add a reference to an Object to the end of the

// ArrayList, increasing its size by one

**boolean** add(Object obj);

Program Example 15 - 1:

**import** java.util.ArrayList;

ArrayList <String> names = **new** ArrayList <String> (10);

names.add("Cary");

names.add("Chris");

names.add("Sandy");

names.add("Elaine");

// remove the last element from the list

String lastOne = names.remove(names.size()-1);

System.out.println("removed: " + lastOne);

names.add(2, "Alyce"); // add a name at index 2

**for** (**int** j = 0; j < names.size(); j++)

System.out.println( j + ": " + names.get(j));

*Run Output:*

removed: Elaine

0: Cary

1: Chris

2: Alyce

3: Sandy

1. A shorthand way to iterate through the collection is provided by a “for each” loop. A for each loop starts you at the beginning of the collection and proceeds through all of the elements. It will not allow you to skip elements, add elements or remove elements.

An example of using it on the collection created in the previous section is

for(String n : names){

System.out.println(n);

}

5. The add() method adds to the end of an ArrayList. To set the data at a particular index, use:

// replaces the element at index with

// objectReference

Object set(**int** index, Object obj)

The index should be within 0 to size-1. The data previously at index is replaced with obj. The element previously at the specified position is returned.

6. Removing an element from a list: The ArrayList class has a method that will remove an element from the list without leaving a hole in place of the deleted element:

// Removes the element at index from the list and

// returns its old value; decrements the indices of // the subsequent elements by 1

Object remove(**int** index);

The element at location index will be eliminated. Elements at locations index+1, index+2, …, size()-1 will each be moved down one to fill in the gap.

7. Inserting an element into an ArrayList at a particular position: When an element is inserted at index, the element previously at index is moved up to index+1, and so on until the element previously at size()-1 is moved up to size(). The size of the ArrayList has now increased by one, and the capacity can be increased again if necessary.

// Inserts obj before the i-th element; increments

// the indices of the subsequent elements by 1

**void** add(**int** index, Object obj);

Inserting is different from setting an element. When set(index, obj) is used, the object reference previously at index is replaced by the new obj. No other elements are affected, and the size does not change.

8. Whether you are adding at the beginning, middle or end, remember that you are adding an object and must instantiate that object somewhere. Strings hide this fact.

names.add(2, "Alyce");

This statement actually creates a String object with the value Alyce.

If we are using any other object, we must instantiate the object. If we have an ArrayList drawList of DrawingTools, we could add a DrawingTool in the following way.

drawList.add(new DrawingTool());

C. Object Casts

1. Java compilers before J2SE 1.5 (codename: Tiger) did not support the typing of an ArrayList as shown above. This new feature is called *generics*. It is a safe way to deal with ArrayLists. You declare what kind of objects you are going to put in the ArrayList. Java will only allow you to put that type of object in, and it knows the type of object coming out. In previous versions of Java, you had to tell the compiler what kind of object you were putting in and taking out.

2. For example, consider the following:

ArrayList aList = **new** ArrayList();

aList.add("Chris");

String nameString = aList.get(0); // SYNTAX ERROR!

System.out.println("Name is " + nameString);

This code creates an ArrayList called aList and adds the single String object "Chris" to the list. The intent of the third instruction is to assign the item "Chris" to nameString. The state of program execution following the add is that aList stores the single item, "Chris". Unfortunately, this code will never execute, because of a syntax error with the statement:

String nameString = aList.get(0); // SYNTAX ERROR!

The problem is a type conformance issue. The get method returns an Object, and an Object does not conform to a String (even though this particular item happens to be a String).

3. The erroneous instruction can be modified to work as expected by incorporating the (String) cast shown below.

String nameString = (String)aList.get(0);

D. The Wrapper Classes

1. Because numbers are not objects in Java, you cannot insert them directly into pre 1.5 ArrayLists. To store sequences of integers, floating-point numbers, or boolean values in a pre 1.5 ArrayList, you must use *wrapper classes*.

2. The classes Integer, Double, and Boolean wrap primitive values inside objects. These wrapper objects can be stored in ArrayLists.

3. The Double class is a typical number wrapper. There is a constructor that makes a Double object out of a double value:

Double r = **new** Double(8.2057);

Conversely, the doubleValue method retrieves the double value that is stored inside the Double object:

double d = r.doubleValue();

4. To add a primitive data type to a pre 1.5 ArrayList, you must first construct a wrapper object and then add the object. For example, the following code adds a floating-point number to an ArrayList:

ArrayList grades = **new** ArrayList();

**double** testScore = 93.45;

Double wrapper = **new** Double(testScore);

grades.add(wrapper);

grades.add(new Double(93.45));

To retrieve the number, you need to cast the return value of the get method to Double, and then call the doubleValue method:

wrapper = (Double)grades.get(0);

testScore = wrapper.doubleValue();

With Java 1.5, declare your ArrayList to only hold Doubles. With a new feature called *auto-boxing* in Java 1.5, when you define an ArrayList to contain a particular wrapper class, you can put the corresponding primitive value directly into the ArrayList without having to wrap it. You can also pull the primitive directly out.

ArrayList grades2 <Double> = new ArrayList <Double>();

grades2.add(93.45);

System.out.println(“Value is “ + grades2.get(0));

1. Iterator

An Iterator is an object that is attached to the ArrayList and allows you to traverse the array from first to last element. Many data structures implement an Iterator. The Iterator keeps track of where it is in the list even if we are adding and removing from it. We will cover the topic of Iterators much more thoroughly in the AB level curriculum.

ArrayList <String> names = new ArrayList <String>();

names.add(“George”);

names.add(“Nancy”);

names.add(“John”);

Iterator iter = names.iterator();

While(iter.hasNext()){

System.out.println(iter.next());

}

|  |  |
| --- | --- |
| **SUMMARY/ REVIEW:** | An ArrayList contains elements that are accessed using an integer index. The package java.util also includes a few other classes for working with objects. |

**ASSIGNMENT:** Lab Assignment A15.1, *IrregularPolygon*

Lab Assignment A15.2, *Permutations*

Lab Assignment A15.3, *Statistics*

Worksheet A15.1, *ArrayList*

Handout A15.1*, ArrayList Methods*