**LIST**

Collection of elements, each with a position or index

Iterators facilitate sequential access to lists

List interface 🡪 part of its API java.util

Classes that implement the List interface provide functionality of an indexed data structure and offer many more operations.

A sample of the operations of List interface:

----> get()

Text

Description automatically generated

----> iterator()

-add/remove()

----> remove()

----> add()

----> indexof()

----> set()

All classes introduced in this chapter support these operations, but they do not support them with the same degree of efficiency.

Diagram

Description automatically generated

List Interface and ArrayList Class

An array is an indexed structure

In an indexed structure,

* elements may be accessed in any order using subscript values
* elements can be accessed in sequence using a loop that increments the subscript

With the Java Array object, you cannot

* increase or decrease its length (length is fixed)
* add an element at a specified position without shifting elements to make room
* remove an element at a specified position and keep the elements contiguous without shifting elements to fill in the gap

Unlike the Array data structure, classes that implement the List interface cannot store primitive types.

Classes must store values as objects.

This requires you to wrap primitive types, such an int and double object wrappers, in these cases, Integer and Double.

**ARRAYLIST CLASS**

The simplest class that implements the List interface

An improvement over an array object, there is no limitation in capacity, it is increased by new elements

Use when:

* you will be adding new elements to the end of a list
* you need to access elements quickly in any order

To declare a List “object” whose elements will reference String objects:

List<String> myList = new ArrayList<String>();

The initial List is empty and has a default initial capacity of 10 elements.

To add strings to the list,

myList.add(“Bashful”);

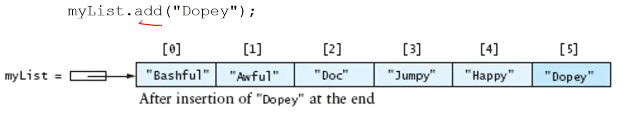
myList.add(“Awful”);

myList.add(“Jumpy”);

myList.add(“Happy”);

Graphical user interface, application, table, Excel

Description automatically generated



Graphical user interface, application, table, Excel

Description automatically generated

Table, timeline, Excel

Description automatically generated

You cannot access an element using a bracket index as you can with arrays (array[1])

Instead, you must use the get() method:

String dwarf = myList.get(2); 🡪 gets the reference of “Sneezy” and makes dwarf to point to same

The value of dwarf becomes “Sneezy”

You can also search an ArrayList:

myList.indexOf(“Sneezy”);

This returns 2 while

myList.indexOf(“Jumpy”);

returns -1 which indicates an unsuccessful search

Generic Collections

The statement

List<String> myList = new ArrayList<String>();

uses a language feature called generic collections or generics.

The statement creates a List of String; only references of type String can be stored in the list.

String in this statement is called a type parameter.

The type parameter sets the data type of all objects stored in a collection.

The general declaration for generic collection is

CollectionClassName<E> variable = new CollectionClassName<E>();

The <E> indicates a type parameter.

Adding a noncompatible type to generic collection will generate an error during compile time.

However, primitive types will be autoboxed:

* ArrayList<Integer> myList = new ArrayList<Integer>();
* myList.add(new Integer(3));
* my.List.add(3); 🡪 also ok! 3 is automatically wrapped in an Integer object
* myList.add(new String(“Hello”)); 🡪 generates a type incompatibility error

Why Use Generic Collections?

Better type-checking: catch more errors, catch them earlier.

Avoids the need to downcast from Object.

Table

Description automatically generated



Marked ones return elements.

If you don’t keep size variable somewhere, it keeps linear time (θ(n))

**Applications of ArrayList**

Graphical user interface, text, application, email

Description automatically generated

You can create your ArrayList with “List<Integer> sth = …” but then you cannot call ArrayList special methods but method calls will be overridden ones because they are virtual by default in JAVA.

🡪 getters, setters happen in constant time

Text, letter

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Method indexOf searches theDirectory by applying the equals method for class DirectoryEntry. Assume DirectoryEntry’s equals method compares name fields.**

**Implementing an ArrayList Class**

KWArrayList 🡪 simple implementation of ArrayList

* Physical size of array indicated by data field *capacity*
* Number of data items indicated by the data field *size*

KW: writers of the book

Chart, diagram

Description automatically generated with medium confidence

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Implementing ArrayList.add(E)

* We will implement 2 add methods
  + One will append at the end of the list
  + The other will insert an item at a specified position

If size < capacity, then to append a new item:

1. insert the new item at the position indicated by the value of size
2. increment value of size
3. return true to indicate successful insertion

Diagram

Description automatically generated with medium confidence

Implementing ArrayList.add(int index, E anEntry)

To insert into the middle of the array, the values at the insertion point are shifted over to make room, beginning at the end of the array and proceeding in the indicating order.

Diagram

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

remove Method

When an item is removed, the items that follow it must be moved forward to close the gap

Begin with the item closest to the removed element and proceed in the indicated order

Diagram

Description automatically generated

Text

Description automatically generated

reallocate Method

Create a new array that is twice the size of the current array and then copy the contents of the new array.

A picture containing text

Description automatically generated

Old array is collected by the garbage collector.

This is amortized analysis. We make the capacity twice as big to not to deal with it everytime new element occurs.

KWArrayList as a Collection of Objects

Earlier versions of Java did not support generics; all collections contained only Object elements

To implement KWArrayList this way,

* remove the parameter type <E> from the class heading
* replace each reference to data type E by Object
* the underlying data array becomes,
  + private Object[] theData;

**VECTOR CLASS**

The Java API java.util contains 2 very similar classes, Vector and ArrayList

New applications normally use ArrayList rather than Vector as ArrayList is generally more efficient

Vector class is synchronized, which means that multiple threads can access a Vector object without conflict