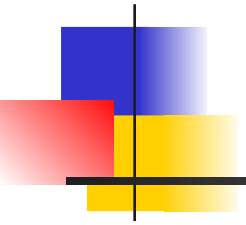


List and ArrayList

James Brucker

Limitations of Arrays

You allocate space for array when you create it:

```
numWords = console.nextInt();

String [ ] words = new String[numWords];
```

What if you don't know the **size of data** in advance?

Example: reading words from a file, but you don't know how many words are in the file?

After you create an array, you cannot change the size.

ArrayList

ArrayList is an alternative:

- an ordered collection of elements
- grows and shrinks as needed!
- can add, delete, replace objects anywhere
- **`ArrayList`** is a class in Java

```
ArrayList food = new ArrayList( );
food.size();  // returns 0.  Its empty
food.add("Apple");
food.add("Banana");
food.size();  // returns 2
System.out.println( food.get(0) ); // Apple
System.out.println( food.get(1) ); // Banana
```

List and ArrayList

List is an interface that defines behavior of all list classes.

- specifies that a list must have: add( ), remove( ), contains( ), indexOf( ), etc.
- you cannot create "List" objects

ArrayList is a class that implements the *List interface*

- ArrayList is a kind of List
- ArrayList provides all the behavior (methods) specified by the List interface

List in Python & Java

Python

```
list = []
list.append("apple")
list.append("banana")
len(list)      # = 2
list[0]        # "apple"
del(list[0])   # delete apple
```

Java

```
ArrayList list = new ArrayList();
list.add("apple");
list.add("banana");
list.size()     // = 2
list.get(0)     // "apple"
list.remove(0) // delete apple
```

Untyped ArrayList holds Objects

- A plain "ArrayList" accepts any kind of Object
- "get" always returns type Object

```
ArrayList list = new ArrayList( );
list.add( "Apple" );  // a string
list.add( LocalDate.now() );  // LocalDate object
list.add( new Double(3.14) ); // another object

// Get something from arraylist
Object obj = list.get(1);

// If you want a String you must use a cast
String fruit = (String) list.get(0);
```

cast

Problem with List

A basic List can contain *any* type of object!

```
ArrayList list = new ArrayList( );
list.add( "apple" ); // String
list.add( new Date() ); // Date object
list.add( new Long(10) ); // Long object
list.add( new Coin(5) );  // Coin

Object obj = list.get(1); // always returns object
```

list.get() requires a *cast*

To get a particular kind of object, we have to ***cast*** the result to the type we want.

This can cause errors.

```
ArrayList list = new ArrayList( );

list.add( "apple" ); // String

list.add( something );

...

String x = (String)list.get(0); // cast to String
```

Using a cast is dangerous

- To get a "String" from list, we *cast* the result to String.
- What if the object is not a String?

```
// Get a string
String fruit = (String) list.get(0);

// If get(1) not a String, an Exception occurs
String fruit2 = (String) list.get(1);

java.lang.ClassCastException: line 5
```

Typed ArrayList

- Arraylist for String only: ArrayList<String>
- <String> is called a **type parameter**.
- Type can be a class or interface, but not primitive

```
ArrayList<String> fruit =
                          new ArrayList<String>( );
list.add( "Apple" );    // a string
list.add( "Orange" );   // string
list.add( new Double(3.14) ); // Error
// Compiler will not allow to add a Double

// No cast!  Result is automatically String
String s = list.get(1);  // No cast!
```

"ArrayList of X"

ArrayList<String> means "ArrayList of Strings"

ArrayList<Person> means "ArrayList of Person"

Common operations

```
ArrayList<String> fruit =
                          new ArrayList<String>( );
fruit.add("Apple");     // add at end of list (0)
fruit.add("Orange");    // add at end of list (1)
fruit.add(1,"Banana"); // insert at index 1

fruit.size();           // 3 things in list
fruit.get(1);           // "Banana" was inserted
fruit.get(2);           // "Orange" was pushed down
fruit.contains("Fig")   // false

fruit.remove("Apple")   // remove first occurrence
fruit.get(0)            // "Banana"
```

Demo

View and inspect an ArrayList using BlueJ.

Notice what happens when number of items in ArrayList increases.

"Code to an Interface"

- Use **List** when declaring variable, parameters, and return types -- not "ArrayList"
- Your code should work with any kind of list.(*)

```
List<String> fruit = new ArrayList<String>( );

public double sum(List<Double> scores)

// Return a list of Students
public List<Student> getEnrollment(String course)
```

(*) Sometimes we need to know if List is *mutable* or *immutable*.

Depend on a Specification

Design principle:

"*Program to an interface, not an implementation*"

- our code should depend on the specification of how a List behaves, not on the way a particular class implements a List.
- GameSolver should depend on the specification for a GuessingGame, not a particular person's implementation of GuessingGame.

Depending on a Specification

```
List<Person> people = ...
```

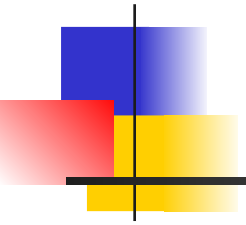
- this ensures that `people` depends only on the *specification* for how List methods work, not on a particular kind of list.

Useful ArrayList<T> Methods

- int size( ) — returns # items in ArrayList
- add(T obj) — add an object to ArrayList (at end)
- add(int k, T obj) — add obj at position k (push others down)
- T get(int index) — get object at given index
- T remove(int index) — delete item from ArrayList & return it
- clear( ) — remove all items from List
- set(int index, T obj) — replace the object at index
- contains(T obj) — "true" if obj is in ArrayList
- ensureCapacity(int size) — make sure ArrayList can hold at least this many elements without resizing

T = the type used to create ArrayList, can be String, Person, Food,...

ensureCapacity( ) improved efficiency when you are adding a *lot* of items to an Arraylist.



Working with ArrayList

Some useful methods

Iterate over all elements

- Print a restaurant menu

```
List<String> menu = Restaurant.getMenu( );
for(int k=0; k < menu.size(); k++) {
    System.out.println( list.get(k) );
}
```

- Print the menu using a for-each loop

```
List<String> menu = Restaurant.getMenu( );
for( String menuItem: menu ) {
    System.out.println( menuItem );
}
```

Copying ArrayList to Array

- Use an ArrayList to save data when you don't know how big the data set is.
- **`list.toArray( array )`** - copy to Array

```
List<String> list = new ArrayList<String>( );

... read some data and add it to list

// create an array large enough to store the data
String [ ] words = new String[ list.size( ) ];
// copy ArrayList to Array
list.toArray( words );
```

Sorting

Sort an ArrayList using the java.util.Collections class

- **Collections.sort(anyList)**
- anyList must contain objects that are *Comparable*
  - String, Double, Long, Int, Date...
  - any class that has a compareTo method

```
List<String> list = Restaurant.getMenu( );

Collections.sort( list ); // sorts the menu
```

Summary

ArrayList is a collection that:

- elements are ordered
- can add, remove, or set elements anywhere
- may contain duplicate values
- size grows/shrinks automatically

ArrayList is not an array.

More Information

- *Big Java*, Chapter 7 *or Core Java*, Volume 1.
- Java Tutorial - has examples
- Java API documentation.