



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 for variable size data

- ArrayList is an **ordered collection** of elements
- ArrayList **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 a basic data type (not a class).

- in Java, *List* is an interface, not a class
- you cannot create "List" objects

ArrayList is a class that behaves like a *List*

- you can ignore "List" for now

Untyped ArrayList is a zoo

- A plain "ArrayList" accepts any kind of Object
- When you "get" an element, it 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);  
String fruit = (String) list.get(0);
```

Using a *cast* is dangerous

- To get a "String" from ArrayList, we must cast the result to String.
- What if the element is not a String?

```
// Get Strings. Must cast the result
```

```
String fruit = (String) list.get(0);
```

```
// If result is not a String, an Exception occurs
```

```
String fruit2 = (String) list.get(1);
```

```
java.lang.ClassCastException: line xx
```

Typed ArrayList

- ArrayList for String (only): `ArrayList<String>`
- `<String>` is called a **type parameter**.
- **Type** can be any class name, but not primitive

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

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

"ArrayList of String"

ArrayList<String> means "ArrayList of Strings"

ArrayList<Food> means "ArrayList of Food"

Common operations

```
ArrayList<String> fruit =  
    new ArrayList<String>( );  
list.add( "Apple" );    // add at end of list (0)  
list.add( "Orange" );  // add at end of list (1)  
list.add( 1, "Banana" ); // add at index 1  
  
list.size( );          // 3 things in list  
list.get(1);           // "Banana" was inserted  
list.get(2);           // "Orange" was pushed down  
list.contains("Fig")   // false  
  
list.remove("Apple")   // remove first occurrence  
list.get(0)            // "Banana"
```

Demo

View and inspect an ArrayList using BlueJ.

Notice what happens when number of items in ArrayList increases.

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 the elements

- Print everything in the restaurant menu

```
ArrayList<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

```
ArrayList<String> list = 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

```
ArrayList<String> list = new ArrayList<String>( );  
  
... read all the data and save in 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

```
ArrayList<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 at any pos'n
- ◆ duplicate values are allowed
- ◆ 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](#).