
Arrays and ArrayLists

CSE 413, Autumn 2002
Programming Languages

<http://www.cs.washington.edu/education/courses/413/02au/>

Readings and References

- Reading
 - » Chapter 3, Section Arrays, *Core Java Volume 1*
 - » Chapter 5, Section Object, Subsection Array Lists, *Core Java Volume 1*
- Other References
 - » "Arrays", Java tutorial
 - » <http://java.sun.com/docs/books/tutorial/java/data/arrays.html>

Arrays

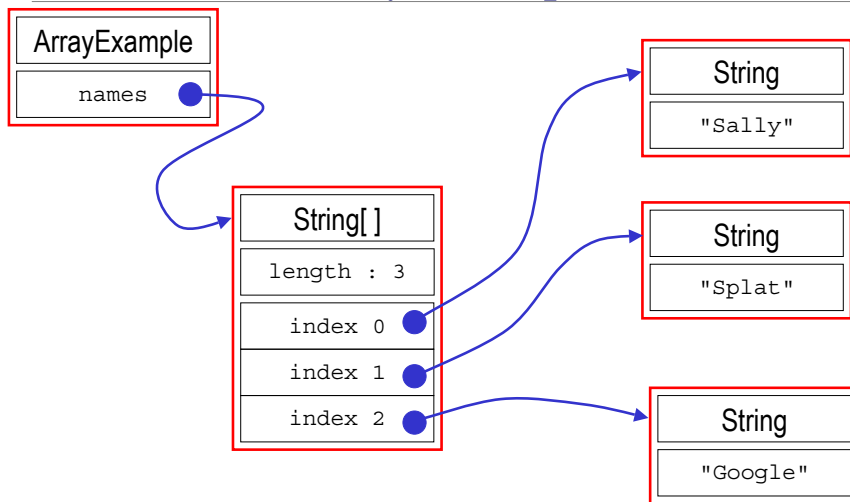
- Java (and many other languages) include *arrays* as the most basic kind of collection.
 - » Simple, ordered collections
 - » Special syntax for declaring values of array type
 - » Special syntax for accessing elements by position
- Unlike ArrayLists:
 - » The size is fixed when the array is created
 - » Can specify the type of the elements of arrays

Array Example

```
public class ArraySample {
    public ArraySample() {
        names = new String[3];
        names[0] = "Sally";
        names[1] = "Splat";
        names[2] = "Google";
        for (int i=0; i<names.length; i++) {
            System.out.println("Name "+i+" is "+names[i]);
        }
    }

    String[] names;
}
```

Array Example



Java Array Object

- Arrays are objects! They...
 - » Must be instantiated with **new** unless immediately initialized
 - » Can contain **Object** references or primitive types
 - » Have class members (`length`, `clone()`,...)
 - » Have zero-based indexes
 - » Throw an exception if bounds are exceeded

Array Declaration and Creation

- Array have special type and syntax:
`<element type>[] <array name> = new <element type> [<length>];`
- Arrays can only hold elements of the specified type.
 - » Unlike `ArrayList`, element type can be `int`, `double`, etc.
 - » type can be `Object`, in which case very similar to `ArrayList`
- `<length>` is any positive integer expression
- Elements of newly created arrays are initialized
 - » but generally you should provide explicit initialization
- Arrays have an instance variable that stores the length
`<array name>.length`

Declaring and Allocating Arrays

- Declare an Array of ten **String** references
`String[] myArray = new String[10];`
- Declare an array and initialize elements
 - » the compiler counts the number of elements in this case
`String[] myArray = { "Java", "is", "cool" };`
- Declare, initialize, and use an array
 - » this is an "anonymous" array
`boolean okay = doLimitCheck(x, new int[] {1,100});`

Array Element Access

- Access an array element using the array name and position: `<array name> [<position>]`
- Details:
 - » `<position>` is an integer expression.
 - » Positions count from zero
 - » Type of result is the element type of the array
- Can update an array element by assigning to it:
`<array name> [<position>] = <new element value> ;`

Looping Over Array Contents

- The length attribute makes looping over Array objects easy:

```
for (index=0; index<myArray.length; index++) {  
    System.out.println(myArray[index]);  
}
```

- The length attribute is a read-only value
 - » You can't change the size of the array after it has been created

Passing Array Objects to Methods

- You must declare that a method parameter is an Array:
`public static void main(String[] args)`
- Arrays are objects and so you are passing a reference when you call a method with an array
 - » This means array contents can be changed by methods
 - » This may be what you want, but if not, you need to make sure that other methods only get a copy of your array and the elements in it

Array Summary

- Arrays are the fundamental low-level collection type built in to the Java language.
 - » Also found in essentially all programming languages
- Size fixed when created
- Indexed access to elements
- Used to implement higher-level, richer container types
 - » ArrayList for example
 - » More convenient, less error-prone for users

The Arrays Class

- There is also a class called `java.util.Arrays`
 - » Note the capital A, this is a class name
 - » part of package `java.util`
 - » utility functions for using arrays
 - search
 - sort
 - initialize
 - » These are **static** methods so they exist and can be used without creating an object first

An Ordered Collection: ArrayList

- `ArrayList` is a Java class that specializes in representing an ordered collection of things
- The `ArrayList` class is defined in the Java libraries
 - » part of the `java.util` package
- We can store *any* kind of object in an `ArrayList`
 - » `myList.add(theDog);`
- We can retrieve an object from the `ArrayList` by specifying its index number
 - » `myList.get(0)`

ArrayList

- **`ArrayList()`**
 - » This constructor builds an empty list with an initial capacity of 10
- **`int size()`**
 - » This method returns the number of elements in this list
- **`boolean add(Object o)`**
 - » This method appends the specified element to the end of this list and increases the size of the array if needed
- **`Object get(int index)`**
 - » This method returns the element at the specified position

Using ArrayLists

- `ArrayList` is part of the `java.util` package
 - » `import java.util.*;` to use `ArrayList`
- Creating a list
 - `ArrayList names = new ArrayList ();`
- Getting the size
 - `int numberOfNames = names.size();`
- Adding things
 - `names.add("Billy");`
 - `names.add("Susan");`
 - `names.add("Frodo");`

Using ArrayLists : import

- ArrayList is part of the java.util package
 - » `import java.util.ArrayList;` to use ArrayList
- The import statement tells the Java compiler where to look when it can't find a class definition in the local directory
 - » We tell the compiler to look in package java.util for the definition of ArrayList by putting an import statement at the top of the source code file
 - » Java always looks in package java.lang on its own

Using ArrayLists : constructor

- Creating a new ArrayList object

```
ArrayList names = new ArrayList ( );
```

- There are several constructors available

- » `ArrayList ()`

Construct an empty list with an initial capacity of 10

- » `ArrayList(int initialCapacity)`

Construct an empty list with the specified initial capacity

- » `ArrayList(Collection c)`

Construct a list containing elements from another collection

Using ArrayLists : size

- Getting the size

```
int numberOfNames = names.size( );
```

- `size ()` method returns integer value that caller can use to control looping, check for limits, etc
 - » Design pattern: The object keeps track of relevant information, and can tell the caller when there is a need to know

Using ArrayLists : add

- Adding things

```
names.add("Billy");
```

- `add(Object o)` method adds an object to the list at the end of the list
- The object can be of any class type
 - » String, File, InputStream, ...
 - » can't add "primitive" types like int or double directly
 - Can use the wrapper classes like Integer to store primitives

Using ArrayLists: get

- ArrayLists provide *indexed* access
 - » We can ask for the i^{th} item of the list, where the first item is at index 0, the second at index 1, and the last item is at index $n-1$ (where n is the size of the collection).

```
ArrayList names = new ArrayList ( );
names.add("Billy");
names.add("Susan");
Object x = names.get(0);
Object y = names.get(1);
```

A Problem

- We want to get things out of an ArrayList
- We might write the following:

```
public void printFirstNameString(ArrayList names) {
    String name = names.get(0);
    System.out.println("The first name is " + name);
}
```
- But the compiler complains at the green line:
 - » incompatible types:
 - » found : java.lang.Object
 - » required: java.lang.String

Recall: Casting

- The pattern is
 - » (<class-name>)<expression>
- For example

```
String name = (String)names.get(0);
```
- Casting an object does *not* change the type of the object
- A cast is a promise by the programmer that the object can be used to represent something of the stated type and nothing will go wrong

Miscasting

- We can lie about casting, but it will be caught at runtime

```
public void printFileList() {
    for (int i=0; i<names.size(); i++) {
        File f = (File)names.get(i);
        System.out.println(f);
    }
}
```

this will fail when you run the program

Reference vs. Primitive Types

- A few Java types are *primitive*:
 - int, double, boolean, and a few other numeric types we haven't seen
 - » Are atomic chunks with no parts (no instance variables)
 - » Exist without having to be allocated with new
 - » Cannot be message receivers, but can be arguments of messages and unary and binary operators
- All others are *reference types*:
 - Rectangle, BankAccount, Color, String, etc.
 - » Instances of the class are created using “new”
 - » Can have instance variables and methods
 - » All are special cases of the generic type “Object”

The Collections Class

- There is a class called java.util.Collections
 - » utility functions for using classes that implement the Collection interface
 - » This class consists exclusively of static methods that operate on or return collections. It contains polymorphic algorithms that operate on collections, "wrappers", which return a new collection backed by a specified collection, and a few other odds and ends.
 - » These are **static** methods so they exist and can be used without creating an object first

Useful methods in Collections class

- static void sort(List list)
 - » Sorts the specified list into ascending order, according to the natural ordering of its elements.
 - » "natural order" is defined when you implement the interface Comparable
- static void sort(List list, Comparator c)
 - » Sorts the specified list according to the order induced by the specified comparator
 - » Comparator lets you define several different orders