Arrays

ACS-1904

LECTURE 1

Arrays

Java, most programming languages in fact, provide tools for dealing with collections of data;

- Arrays
- ArrayLists
- Sets
- …

Arrays

Array

- once allocated with a certain length, the number of elements cannot change

  ```
  int[] x = new int[10];
  ```

  - Now the array x has 10 elements, and it will have 10 elements until the program ends!
- Once declared the array has a *length* field. This field is final (remember what that means?)
- This is different from the ArrayList *size()* method.
  - There's lots more to talk about here but for our purposes it's enough that we know about the length field.

Arrays

- elements of an array can be of any type (but just one type), including primitive types
  - Arrays are homogeneous data structures
  - All elements are the same primitive data type or the same type of object.

Accessing individual elements

```
x[5] = 2042;
```

For now we consider arrays of one dimension → 1D arrays

… only one subscript used. E.g. `x[5]`

Example

Consider the months of the year and the number of days per month.

- We can represent this information using a 1D array:

```
int[] daysInMonth=
    {31,28,31,30,31,30,31,31,30,31,30,31};
```

- This declaration says the variable `daysInMonth` is an array of 12 `int` values that are initialized t…
  `31,28,31,30,31,30,31,31,30,31,30,31`

Example

- Individual elements of an array are referenced using an ***index***
  - Also called *subscript* or *position*
- For instance the first element can be accessed using

  `daysInMonth[0]`

  And the last element, element 12, can be accessed using

  `daysInMonth[11]`
- Indexing starts at 0 and ends at n -1:
  - `daysInMonth[0], … daysInMonth[11]`

Example

- Each element of an array is a variable in its own right and can be used just like a variable
- So, instead of the previous initialization, we could have used a declaration and subsequent assignment statements:

```
int[] daysInMonth = new int[12];
daysInMonth[0] = 31;
daysInMonth[1] = 28;
daysInMonth[2] = 31;
…etc…
daysInMonth[11] = 31;
```

Declaring daysInMonth to be an array of ints

Length of the array

Assignment statements

Invalid index

- If an index is invalid; for example:

```
daysInMonth[-4]

daysInMonth[12]
```

  - a runtime error called an *ArrayIndexOutOfBoundsException:* occurs → program is halted.

Array length

- Each array has its own length field which specifies the length of the array (the number of elements in the array).
  - For example

```
daysInMonth.length
```

- A very common usage of this field is as part of a for statement. For example

```
for (int i = 0; i<daysInMonth.length; i++)
    System.out.print(daysInMonth[i]+" ");
```

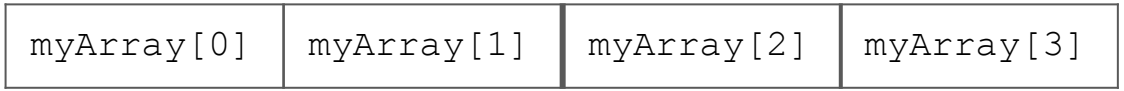
Example(MonthLengths.java)

```
/**
 * Display number of days in each month
 */
public class MonthLengths{
    public static void main(String[] args){
        int[] daysInMonth =
            {31,28,31,30,31,30,31,31,30,31,30,31};
        System.out.println("Days for each of "
            +daysInMonth.length+" months ");
        for (int i = 0; i< daysInMonth.length; i++)
            System.out.print(daysInMonth[i]+"   ");
    }
}
```

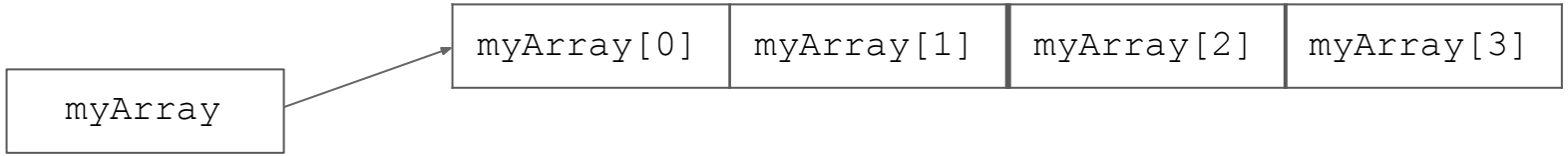
Storage of arrays in main memory

An array is allocated contiguous space – enough memory to hold its length field, its values and no more

- So an array `myArray` of 4 elements would have space reserved for it as

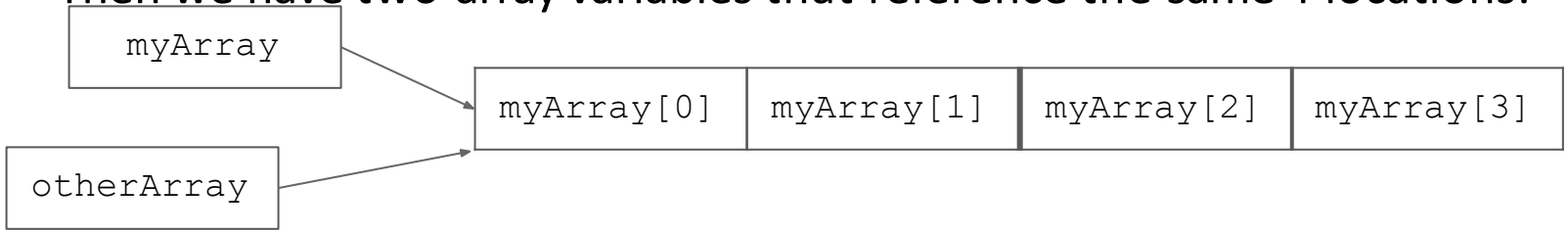


- And the variable `myArray` holds the address of those locations:



Storage of arrays in main memory

- If we have two arrays declared as
  - `int[] myArray = {5, 7, 1, 9};`
  - `int[] otherArray = new int[4];`
  - And we execute
  - `otherArray = myArray;`
  - Then we have two array variables that reference the same 4 locations:



Storage of arrays in main memory

- To create a distinct copy of an array of primitive types we must copy element-by-element…consider:

```
int[] myArray = {5, 7, 1, 9};
int[] otherArray = new int[4];
```

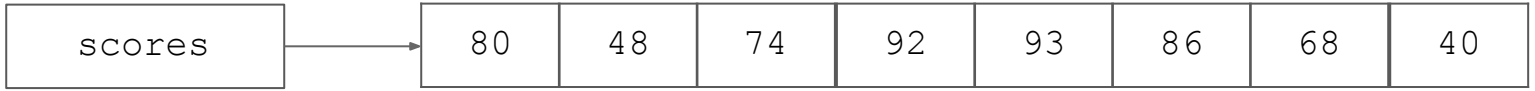
and

```
for (int i=0; i<otherArray.length; i++)
    otherArray[i]=myArray[i];
```

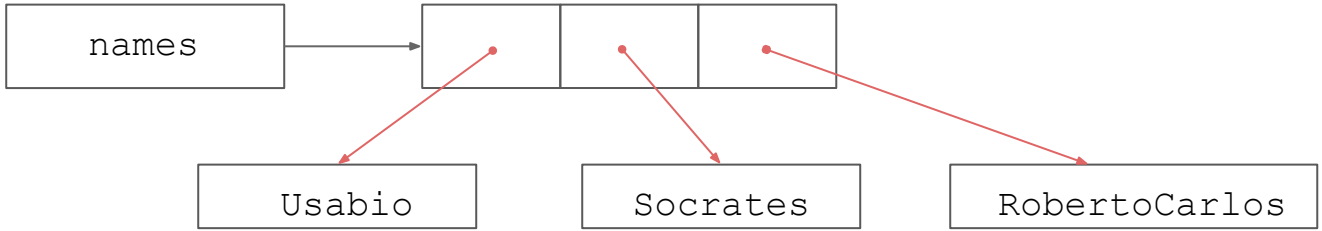
Then we end up with:

Storage of arrays: primitive vs object

An array of primitive types e.g. ints



An array of objects - only the references are in contiguous locations



Enhanced for

- Can be used with arrays
  - e.g.

```
for (int days : daysInMonth )
    System.out.print( days +" ");
```

  - The *enhanced for* is sometimes called the *for-each* statement

Enhanced for

- expresses a programming idiom succinctly as no loop counter is required.
  - cannot be used :
    - to iterate backwards
    - to access elements of more than one array
    - on partially filled arrays
    - to assign new values

Passing arguments to main(Args.java)

By convention the main method receives an array of strings.

A comma separated list of Strings

Example: list the arguments passed in to main

```
public class Args{
    public static void main(String[] args){
        System.out.println("The elements of args:");
        for (String s: args)
            System.out.print(" "+s);
    }
}
```

Arrays and static methods

- Some static methods review
  - Values are passed into methods using the argument-parameter interface
  - Java is a “pass by value” language
    - a copy of the value of the argument is passed into the method
  - Changes made to the parameters in the method body are not reflected back to the caller.

Arrays and static methods

- An array variable is a reference variable
- So, what's passed into the method
  - A copy of the reference
  - Not a copy of all of the values in the array
- Changes made to the values of the array ARE reflected back to the caller
- Note: a change made to the reference variable won't change it in main

Arrays and static methods

```
35          // print the list
36          System.out.println(list);
37          for(int i = 0; i < count; i++){
38              System.out.println(list[i]);
39          }//end for
40
41          // change an element using a static meth
42          for(int i = 0; i < count; i++){
43              list[i] = list[i] + 100;
44          }//end for
45
46          // print the list witht he change
47          System.out.println(list);
48          for(int i = 0; i < count; i++){
49              System.out.println(list[i]);
50          }//end for
51
52          System.out.println("end of program");
53      }// end main
54
```

Some typical main method code

Look at these two routines

We can do better!

Arrays and static methods

```
35        // print the list
36        System.out.println(list);
37        printList(list, count);
38
39        // change an element using a static method
40        changeTheList(list, count);
41        System.out.println(list);
42
43        // print the list witht he change
44        printList(list, count);
45
```

Some typical main method code

Ahhh! that’s a relief. So much tidier, So much neater.

Arrays and static methods

```java
35      // print the list
36      System.out.println(list);
37      printList(list, count);
38
39      // change an element using a st
40      changeTheList(list, count);
41      System.out.println(list);
42
43      // print the list witht he change
44      printList(list, count);
45
```

```java
51
52      public static void printList(int[] l, int c){
53          for(int i = 0; i < c; i++){
54              System.out.println(l[i]);
55          }//end for
56      }// end print list
57
```

Arrays and static methods

```java
35        // print the list
36        System.out.println(list);
37        printList(list, count);
38
39        // change an element using a static method
40        changeTheList(list, count);
41        System.out.println(list);
42
43        // print the list witht he change
44        printList(list, count);
45
```

```java
59
60    public static void changeTheList(int[] l, int c){
61        System.out.println(l);
62
63        for(int i = 0; i < c; i++){
64            l[i] = l[i] + 100;
65        }//end for
66    }// end change the list
67
```

Parallel arrays

Can be useful to use two or more arrays where elements at the same index have a relationship

Example. Consider two parallel arrays (a table) where for each student we have a number and a name

number	name
123	Joe
222	Linda
345	Mary
567	Peter
890	Lee

Use an array for each column

Parallel arrays

Given a number it's easy for us to look up the name

The look-up can be programmed in multiple ways

- Like using an if-else or switch structure
- For example, the table can be represented with two arrays:

```
String[] name = {"Joe","Linda","Mary","Peter","Lee"};
int[] number = {123, 222, 345, 567, 890};
```

Example(StudentInfo.java)

```
import java.util.Scanner;
public class StudentInfo{
    public static void main(String[] args){
        String[] name = {"Joe","Linda","Mary","Peter","Lee"};
        int[] number = {123, 222, 345, 567, 890};
        Scanner kb = new Scanner(System.in);
        System.out.println("Enter student number: ");
        int toFind = kb.nextInt();
        for (int i=0; i<number.length; i++)
            if (toFind==number[i])
                System.out.println(name[i]);
    }
}
```

Partially filled arrays

- Partially filled arrays are useful when we define our arrays to handle a maximum size, but at runtime we use only part of the array(s)
- E.g. Consider previous example, but now the arrays are larger and only a portion of the arrays are used:

number	name
123	Joe
222	Linda
345	Mary
567	Peter
890	Lee
?	?
?	?

used portion

unused portion

Partially filled arrays(StudnetInfo1.java) *not in package

```
import java.util.Scanner;
public class StudentInfo{
   public static void main(String[] args){
      // the last two entries of each array are unused
      // ... they could have any value
      String[] name = {"Joe","Linda","Mary","Peter","Lee", null, null};
      int[] number = {123, 222, 345, 567, 890, 0, 0};
      // size of used portion
      int lastElement = 5;
      Scanner kb = new Scanner(System.in);
      System.out.println("Enter student number: ");
      int toFind = kb.nextInt();
      for (int i=0; i<lastElement; i++)
         if (toFind==number[i])
            System.out.println(name[i]);
   }
}
```

Arrays class: sorting and searching

The `Arrays` class

Two of its methods are

- `sort` – a static method that can be used to sort the elements of an array into ascending order
- `binarySearch` – a static method that can be used to locate an element of a sorted array.

To use these a program must import `java.util.Arrays`

Arrays class: sorting

`sort`

- an overloaded method that accepts
  - an array of primitive types
    - `int`, `long`, `short`, `char`, `byte`, `float`, `double`
  - or any array of objects that have a `compareTo` method
    - E.g. `String` class

The array's elements are rearranged in ascending order.

Arrays class: sorting

E.g. 1: array of ints

```
int[] myArray = {8,5,4,3,3,1};
Arrays.sort(myArray);
for (int elt : myArray) System.out.print(elt+" ");
```

produces the output: `1 3 3 4 5 8`

E.g. 2: array of Strings

```
String[] names ={"John", "Jim", "Tom", "Sue"};
Arrays.sort(names);
for (String n : names) System.out.print(n+" ");
```

produces the output: `Jim John Sue Tom`

Arrays class: searching

`binarySearch`

- an overloaded method that accepts two arguments: an array and a search element.
- The array's elements must be in ascending order
- The searches the array and returns the index of the search element or a negative integer (i.e. <0) if it cannot be found

The algorithm used is the binary search technique

Arrays class: searching(SortAndSearch.java)

E.g. 1:

```
String[] names ={"John", "Jim", "Tom", "Sue"};
Arrays.sort(names);
System.out.print(n+ "Tom found at"
    +Arrays.binarySearch(names,"Tom"));
```

produces the output: `Tom found at 3`

E.g. 2:

```
String[] names ={"John", "Jim", "Tom", "Sue"};
Arrays.sort(names);
System.out.print(n+ "Searching for Bill results in"
    +Arrays.binarySearch(names,"Bill"));
```

produces the output: `Searching for Bill results in -1`

Traversing arrays

- Maybe the most common/fundamental array operation is array traversal:
- Many (maybe even most) array computations can be implemented by looping over each element in the array and performing some operation on each element in turn.
- This is called traversing the array.

Traversing arrays

```
// now travers the array, only the used portion
for(int i = 0; i < count; i++){
    // do some computation
    // could be adding, counting, squaring,....

    //how can we get the numbering to start at 1?
    System.out.println(i + ". " + list[i]);
}// end for
```

As an exercise can you change this code so that it traverses the array from index n-1 to index 0?

Some common array algorithms

Given an array named X

- What is the sum of its elements?
- What is the average of its elements?
- What is the largest value of its elements?
- What is the smallest value of its elements?

Some common array algorithms

Given an array named x where x is an array of primitive numeric values

e.g. `int x[] = …`

What is the sum of its elements?

```
int sum = 0;
for (int i = 0; i < x.length; i++){
    sum += x[i];
}
```

What statement must we include to compute the average?

Some common array algorithms

Given an array named x where x is an array of primitive numeric values
e.g. `int x[] = …`

What is the largest value of its elements?

```
int largest= x[0];
for (int i = 1; i < x.length; i++){
    if (x[i] > largest)
        largest= x[i];
}
```

What change is required to find the index of the largest value?
What change is required to find the smallest value?
What change is required to find the index of the smallest value?

array v. array list

ArrayLists	Arrays
Can have elements of any type (recommendation: have just one type)	Has elements of only one type
A collection of objects	A collection of primitive types or objects
Resizable	Of fixed length
Can easily grow in size	How do you make an array larger?
Can easily shrink in size	Create a new, bigger array.
`size()` method	`length` field
Aside: The backing storage structure for an ArrayList is actually an array.	

Converting between an array and an array list

Sometime you may need to convert from one data structure to another structure:

Perhaps because you :

- Must use a pre-defined method of some class that uses just one of these structures.
- You are coding a method that must return an array, but your data is only available to you as an array list.
- You want to add an unknown number of elements to an array.

Converting from an arraylist to an array

Sometime you may need to **convert an array list to an array**:

- `java.util.ArrayList` contains a method `toArray` that can be used for this purpose. Consider the code:

```
// the array list
ArrayList<String> al = new ArrayList();
al.add("John"); al.add("Jim"); al.add("Tom"); al.add("Sue");

// an empty array: names
String[] names= new String[al.size()];

// insert values into the array
al.toArray(names);
for (String n : names) System.out.print(n+" ");
```

The output produced is: `John Jim Tom Sue`

Converting from an array and an array list

Sometime you may need to **convert an array to an array list**:

- `java.util.Collections` contains a method `addAll` that can be used for this purpose. Consider the code:

```
// the array
String[] names ={"John", "Jim", "Tom", "Sue"};
Arrays.sort(names);

// an empty array list: al
ArrayList<String> al = new ArrayList();

// insert values into the array
Collections.addAll(al, names);
System.out.println(al);
```

The output produced is: `[Jim, John, Sue, Tom ]`