Arrays Continued

Chapter 12, page 441 - 467
Chapter 14, page 535 - 546

Review of arrays

- A data structure to store a collection of common and related data in one contiguous chunk of memory
- It's size cannot be changed
- It can store primitive data type or object references
- Subscripts start at 0
- Length is stored in a constant, `.length`

Objectives

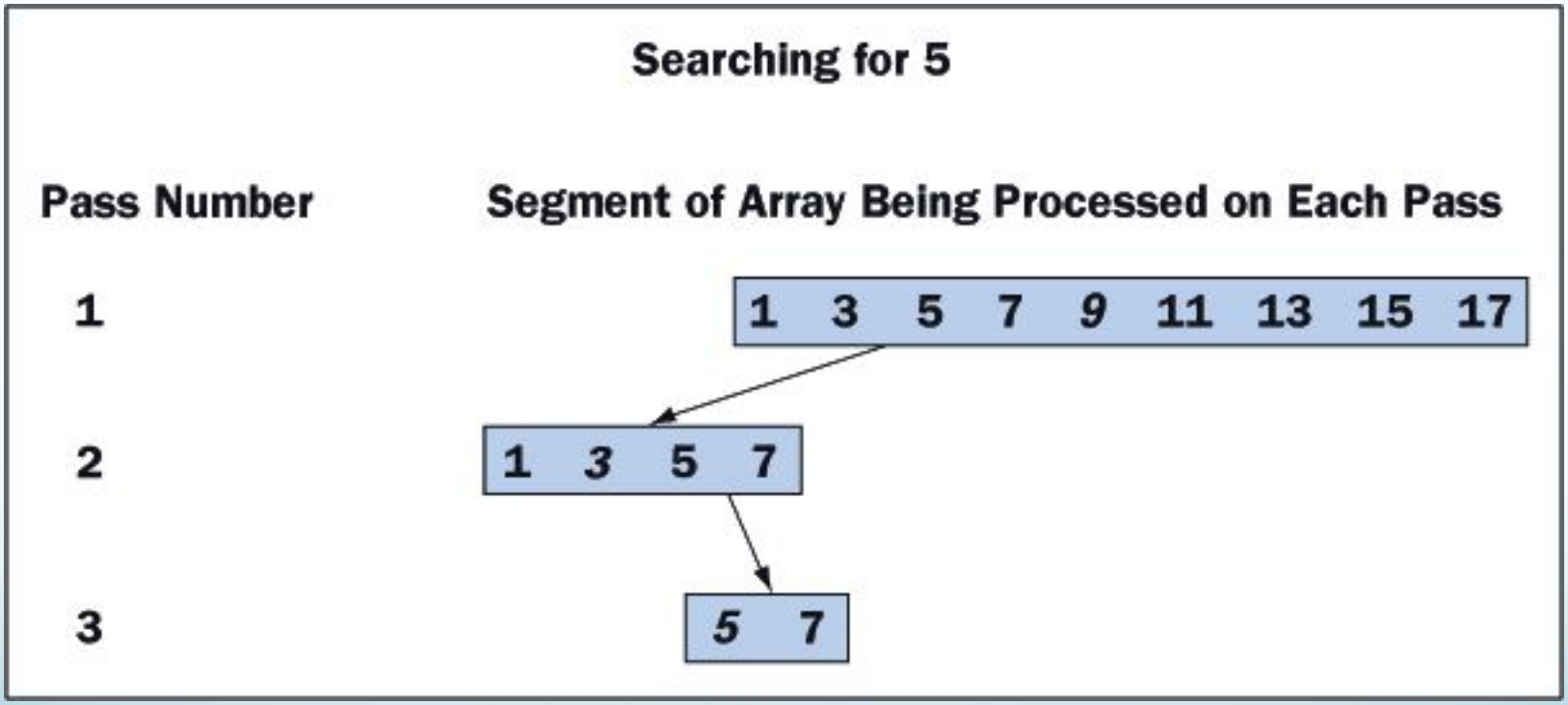
- Chapter 12
  - Searching and Sorting Arrays
- Chapter 14
  - Collections
  - ArrayList

Searching Algorithms

- Linear Search
  - Each element is examined in sequence until the target value is found.
  - When searching through an array of objects, use `.equals` or `compareTo`
- Binary Search
  - Array must be sorted.
  - Look at middle element and eliminate upper half or lower half

Binary Search

- Works only with sorted data



Section 12.2

Sorting

- Selection Sort
  - In ascending order, find the smallest element.
  - Swap it with element in first position
  - Repeat
- Bubble Sort
  - Compare pairs of elements, swap if necessary
  - At end of array, the largest element is in the correct position
  - Repeat

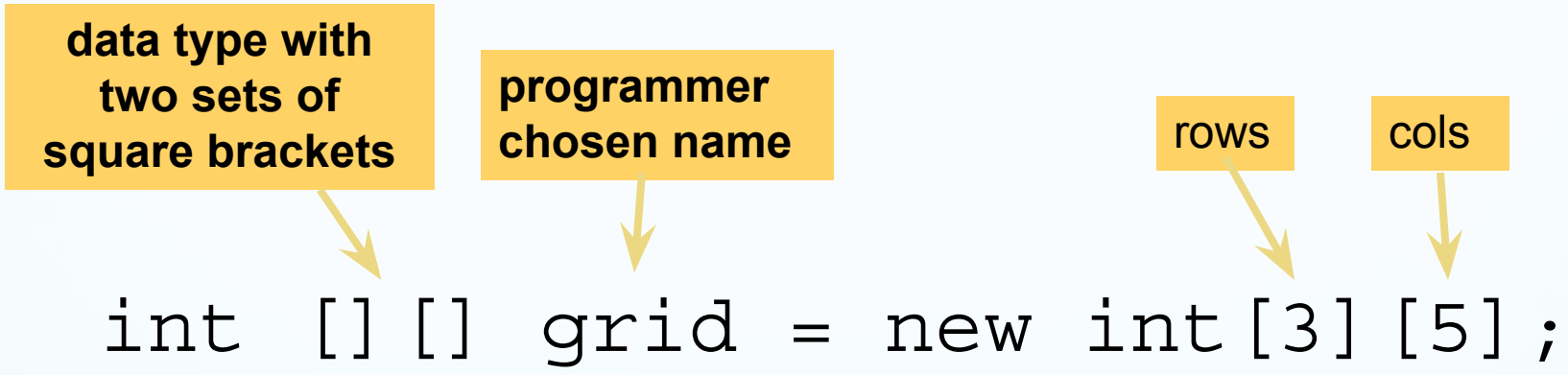
Sorting

- Insertion Sort (ordering a hand of cards)
  - Compare first two elements, swap if necessary. This becomes the sorted part of the array.
  - Get the next element from the unsorted portion of the array and “insert” it into the sorted part of the array
  - Repeat

Insertions and Removals

- Insertion
  - Determine if array is large enough to insert
  - Find location of where element should be inserted
  - Shift all elements right
  - Move the new element in
- Removal
  - Locate the element to remove
  - Shift elements left

Two-Dimensional Arrays



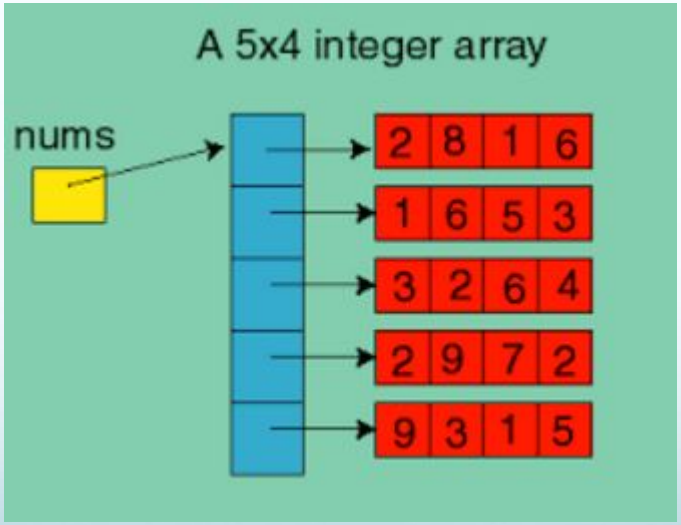
- Row subscripts 0 – 2
- Column subscripts 0-4

An array of arrays

```
int[][] nums = new int [5][4];
   for (int (r = 0; r < nums.length; r++)
      for (int c = 0; c < nums[0].length; c++)
```

Number of rows

Number of columns



An array of arrays

- What does the following loop do?

```
int[][] table = new int[4][5];
//some code to populate table
int[] mystery = new int[4];
int m = 0;
for (int[] oneRow : table){
    for (int element : oneRow)
        mystery[m]+= element;
    m++
}
```

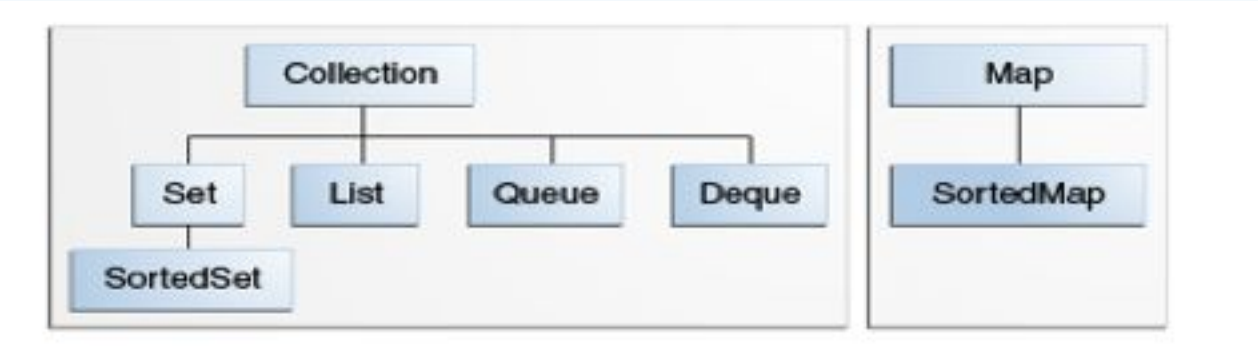
Algorithms for 2-dim arrays

- Finding totals by rows
  - Nested loop
  - The outside loop control variable is the row
  - The inside loop control variable is the column
- Finding totals by columns
  - Nested loop
  - The outside loop control variable is the column
  - The inside loop control variable is the row

Introduction to Collections

- The Java Collections Framework
- Includes
  - Interfaces
  - Data structures
  - Methods

The core collection interface



The ArrayList Class

- It can only store objects, no primitive data
- It is resizable. Elements can be added and removed (dynamic)
- Elements can be accessed by their integer index (position in the list)
- It can be homogeneous (all items are the same type) or heterogeneous (items can be of different type)
- It **`extends AbstractList`** and **`implements List`**

Methods of List<E>

- `boolean add(E obj)`
  - adds to end of the list
  - always returns true
  - if element is not of type `E`, throws a `ClassCastException`

- `void add(int index, E ele)`
  - all elements at `index` and higher are shifted right by having indices incremented by 1
  - `ele` is added at index
  - if `index` is out of range, throws an `IndexOutOfBoundsException`

Methods of List<E>

- `E set(int index, E element)`
  - if element is not of type `E`, throws a `ClassCastException`
  - returns the original element at `index`, before `set` is called
  - if `index` is out of range, throws an `IndexOutOfBoundsException`

- `E remove(int index)`
  - element at `index` is returned
  - all elements at indices greater than `index` are shifted left by having their indices decremented by 1
  - if `index` is out of range, throws an `IndexOutOfBoundsException`

Methods of List<E>

- `E get(int index)`
  - returns element at `index`
  - if `index` is out of range, throws an `IndexOutOfBoundsException`
- `int size()`

ArrayList declaration

- Two ways to declare an ArrayList. "E" represents the type of object for each element
- Method 1

```
ArrayList<E> myArrayList = new ArrayList<E>();
```

  - This method creates an ArrayList object with an ArrayList reference.

- Method 2

```
List<E> myArrayList = new ArrayList<E>();
```

  - This method offers a little more flexibility. It creates an ArrayList object with a List reference.

Example 1

```
List<String> club = new ArrayList<String>();
club.add("Sheldon");
club.add("Leonard");
club.add("Raj");
club.add("Howard");
System.out.print(club);
```

```
[Sheldon, Leonard, Raj, Howard]
```

Example 1 (cont)

```
club.set(1, "Mickey");

System.out.print(club);
```

```
[Sheldon, Mickey, Raj, Howard]
```

```
club.add(0, club.remove(club.size()-1));

System.out.print(club);
```

```
[Howard, Sheldon, Mickey, Raj]
```

Example 2

```
List <Integer> mylist = new ArrayList<Integer>();
for (int j = 0; j < 10; j++) {
        int r = (int)(Math.random()*101);
        if ((int)(Math.random()*2) == 1)
           mylist.add(r);
        else
           mylist.add(-1* r);
}
for (Integer num:mylist)
        System.out.print(num + "\t");
System.out.println();
```

-65 -23 -14 15 -20 31 40 79 -97 -91

Example 3

```
List<String> students = new ArrayList<String>();
students.add("Alex");
students.add("Bob");
students.add("Carl");
for (int k=0; k<students.size(); k++) {
  System.out.print(students.set(k,"Alex") + " "
 );
}
System.out.println();
for (String str : students) {
      System.out.print(str + " " );
}
```

Alex Bob Carl
Alex Alex Alex

Example 4

```
//pre: li contains only String objects
//post: all Strings of length=len removed
public static void removeAllLength
                              (List li, int len)
{
//  !!wrong way!!
   String temp;
   for(int i = 0; i < li.size(); i++)
   {  temp = (String)li.get(i);
      if( temp.length() == len )
      li.remove(i);
   }
}
```

Example 4

```
//pre: li contains only String objects
//post: all Strings of length=len removed
public static void removeAllLength
                              (List li, int len)
{   right way  :)
//right way  :)
   String temp;
   for(int i = 0; i < li.size(); i++)
   {  temp = (String)li.get(i);
   if( temp.length() == len )
   {  li.remove(i);
      i--;
   }
   }
}
```

Wrapper Classes

- Primitive data types cannot be used in ArrayList
- Java provides a mechanism to get around this by using a wrapper class.
- For each primitive data type, there is a corresponding wrapper class
  - `int → Integer`
  - `double→ Double`
- Auto-boxing is the automatic wrapping of a primitive data type into its corresponding wrapper class
- Auto-unboxing is the automatic conversion back to a primitive data type.

A useful Integer Class method

- **`Integer.parseInt()`**
  - converts a **`String`** into an integer.
  - It is a **`static`** method (class). Call it through a class **`Integer`**, not through an **`Integer`** object

```
int x = Integer.parseInt("1234");
```

A useful Double Class method

`Double.parseDouble("1.234")`

- Converts the `String "1.234"` to a `double`
- Also a static method. Call it through the class name

instanceof

- You can mix object types in an ArrayList.

```
List<Object> stuff = new ArrayList<Object>();
stuff.add(new Student ("Jay"));
stuff.add(new Car ("Honda"));
stuff.add(new Candy ("Twix"));

if (stuff.get(0) instanceof Student)
   Student s = (Student) (stuff.get(0));
```

Arrays or ArrayLists

- ArrayList elements can easily be added to and removed from the list
- Uses methods
- One dimension only
- Arithmetic operations involve the overhead of “boxing” and “unboxing”

- Arrays are immutable
- Uses subscripts
- The physical size is not always the same as the logical size. You have to instantiate an array greater than what you really need.
- Can be multi-dimensional
- Arithmetic operations are easier