Searching and Sorting in Java

Chris La Pilla
CompSci 6
Summer 2007

Sorting

• Consider the problem of sorting items in an ArrayList in ascending order
• Ascending order means that each element is less than or equal to all subsequent elements
• Example:
  – Input array list: [29, 32, 4, 10, 2, 45, 9, 10, 4]
  – Sorted list: [2, 4, 4, 9, 10, 10, 29, 32, 45]

Sorting in Java

• In Java API, there is a utility class Collections which contains static methods for operating on lists, sets, and maps
• The Collections.sort method takes a List as a parameter and sorts it
• Note that this method modifies the given list
• Example (suppose list is a variable of type ArrayList<Integer>):
  Collections.sort(list); //list is now sorted

Comparing objects

• Sorting in ascending order means each element is less than or equal to the subsequent elements
• “Less than”, “equal to”, “greater than” make sense for integers, floating-point types, and characters
• But what if we want to sort arbitrary objects? (e.g. Circles, StudentRecords, etc.)
• We need to define how objects of this type are ordered

Java API: Comparable Interface

• The Java API contains an interface Comparable<T>, which contains a single method
  int compareTo(T obj)
• compareTo returns
  – zero if the given object is equal to this object
  – any negative integer if this object is less than the given object
  – any positive integer if this object is greater than the given object
• The Collections.sort method will work properly with any class that correctly implements the Comparable interface

Example: String class

• The String class implements the Comparable interface
• Strings are compared lexicographically (like alphabetical order)
• Digits come first, then uppercase characters, then lowercase characters
• See Chapter 6.2.4 for details
**Example**

```java
String cat = "cat";
int c = cat.compareTo("dog"); //negative
       c = cat.compareTo("cat"); //zero
       c = cat.compareTo("CAT"); //positive

    • In last example, "cat" comes after "CAT" because uppercase letters come first
```

**Implementing Comparable**

- You can use the Comparable interface with your own classes
- Must implement the compareTo method so that it conforms to the specification (negative, zero, or positive for less than, equal to, or greater than)
- Once you've done this, you can use the Collections.sort method with your own classes
- Example: sorting Circle objects by size

**Comparators**

- What happens if we want to be able to sort our objects more than one way?
  - e.g. in iTunes, can sort music by song title, album title, artist, rating, etc.
- Comparable interface only allows us to implement a single sorting criterion
- There is a second version of the Collections.sort method that takes in a list and a comparator, which is an actor object that defines the sorting criterion
- We can use multiple comparators to define multiple ways of sorting the same type of object

**Comparator**

- See the Comparator<T> interface in the Java API
- It contains a method
  ```java
compare(T obj1, T obj2)
  ```
  which behaves the same as the compareTo method of Comparable:
  - return negative int if obj1 less than obj2
  - return zero if obj1 equals obj2
  - return positive int if obj1 greater than obj2
- Example: comparators for Circle

**Searching**

- We've already seen the contains method of the List interface
- It goes through each element in the list from start to finish until it finds the given element (this is called a linear search)
- In the worst case, how many elements must we look at?
- Can we do better if we know that the list is sorted? How?

**Searching**

- In the Collections class, there is a method called binarySearch
- Does the same thing as contains, but does it more efficiently
- However, the list must be sorted (precondition) or else the method won't work
- We'll discuss algorithmic details tomorrow
- In today's classwork, we'll compare running times of contains and binarySearch