Java sorting routines, comparator, comparable

Dr Tom Ridge

2018 Q1

Java sorting routines

- Java has a method `Collections.sort`

Sorting and the comparator interface

Look at the following:

```
import java.util.*;

tmpl = cons(1,cons(3,cons(4,cons(2,nil()))));

tmph = copy(tmpl);

// this sorts in place! tmph is changed!!!
Collections.sort(tmph);

tmph;
```

- Running this code gives you a sorted list.
- So Java “knows” how to sort lists of integers.
- But what if you want to sort things that Java doesn’t know about?

A simple class

```
class Person {
  public String first=""; public String last="";
  Person(String f, String l) {first=f; last=l; }
  public String toString() { return first+" "+last; }
}

t = new Person("Tom","Ridge");
u = new Person("Alf","Bloggs");
Collections.sort(cons(t,cons(u,nil())));
```

What happens when you call the sort method? Something goes wrong (exception)

Interfaces

- Recall Java interfaces

The comparator interface, comparable

- Look at the Java API, `Comparator` and `Comparable`

Comparator: `int compare(T o1, To2)`
Comparable: `int compareTo(T o)`

- The int that is returned means:

  (Comparator) Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

  (Comparable) Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Difference between comparator and comparable

- What is the difference between Comparator and Comparable?
- If `c` is a comparator, you compare two objects by typing `c.compare(o1,o2)`. If a class (such as `String`) supports the `compareTo` method, you can type e.g. `s1.compareTo(s2)` to compare the strings `s1` and `s2`.
- With `c.compare(o1,o2)` there are 3 objects: c,o1,o2
- With `s1.compareTo(s2)` there are 2 objects: s1 and s2
- In the String class, `compareTo` compares strings in dictionary order. YOU CAN NOT CHANGE THIS! With comparators, the comparator c can be whatever you want: you can compare strings in dictionary order, or reverse dictionary order, or whatever. Comparators are much more flexible!

Warm up to comparators - an int comparator

Note anonymous class

```
Comparator c = new Comparator() {
    public int compare(Object o1, Object o2) {
      boolean b1 = o1.getClass().equals(Integer.class);
      boolean b2 = o2.getClass().equals(Integer.class);
      if(b1 && b2) {
        return ((Integer)o1) - ((Integer)o2);
      }
      return 0; // should really throw an exception?
    }
  };

c.compare(new Integer(2),new Integer(3));
c.compare(new Integer(5),new Integer(1));
```

Experimenting with comparators

```
tmpl = cons(1,cons(3,cons(4,cons(2,nil()))));
tmph = copy(tmpl);

Collections.sort(tmph,c);

Comparator d = new Comparator() {
    public int compare(Object o1, Object o2) {
      return (-1) * c.compare(o1,o2);
    }
  };

tmph = copy(tmpl);

Collections.sort(tmph,d);
```

More experiments with comparators

```
tmpk = cons("bert",cons("charlie",cons("alf",nil())));

tmph = copy(tmpk);

// sorts strings in dictionary order
Collections.sort(tmph);
```

More experiments with comparators (sorting strings in dictionary order)

```
Comparator e = new Comparator() {
    public int compare(Object o1, Object o2) {
      boolean b1 = o1.getClass().equals(String.class);
      boolean b2 = o2.getClass().equals(String.class);
      if(b1 && b2)
        return ((String)o1).compareTo((String)o2);
      // should really throw an exception?
      return 0; } };

e.compare("bert","alf");

tmph = copy(tmpk);

Collections.sort(tmph,e);
```

More experiments with comparators (sorting strings in reverse order)

```
Comparator f = new Comparator() {
    public int compare(Object o1, Object o2) {
      return (-1) * e.compare(o1,o2);
    }
  };

tmph = copy(tmpk);

Collections.sort(tmph,f);
```

Comparators on our own class

```
class Person {
  public String first = "";
  public String last = "";
  public Person(String f, String l) { first=f; last=l; }
  public String toString() { return first+" "+last; }
}

alf = new Person("alf","michaels");
bert = new Person("bert", "koala");
charlie = new Person("charlie", "pickens");

tmpj = cons(bert,cons(charlie,cons(alf,nil())));

tmph = copy(tmpj);

Collections.sort(tmph); // this can't work! (why not?)
```

Comparators on our own class

```
// sort on first name
Comparator g = new Comparator() {
    public int compare(Object o1, Object o2) {
      String a = ((Person)o1).first;
      String b = ((Person)o2).first;
      return a.compareTo(b);
    }
  };

Collections.sort(tmph,g);
```

Comparators for our own class

```
// sort on last name
Comparator h = new Comparator() {
    public int compare(Object o1, Object o2) {
      String a = ((Person)o1).last;
      String b = ((Person)o2).last;
      return a.compareTo(b);
    }
  };

Collections.sort(tmph,h);
```

Comparators further reading

http://docs.oracle.com/javase/tutorial/collections/interfaces/order.html - ordering tutorial

Some exercises on comparators

- Task 1: Write a comparator to compare strings ignoring their first character (you can use `lexcompare(s1,s2)`, which returns true if s1 is less than s2 in dictionary order).
- Task 2: Write a comparator to compare names ("first last" eg "Tom Ridge") based on the dictionary order of the last name (you need to discard the first word and any spaces following the first word);

```
Comparator c = new Comparator() {
    public int compare(Object o1, Object o2) {
      String s1 = ((String)o1);
      String s2 = ((String)o2);
      ...;
    }
  };
```

Solutions, task 1, lexcompare

```
// assume the strings are not equal
boolean lexcompare(String s1, String s2) {
  while(true) {
    if(s1.equals("")) return true;
    if(s2.equals("")) return false;
    if(s1.charAt(0) < s2.charAt(0)) return true;
    if(s1.charAt(0) > s2.charAt(0)) return false;
    s1=s1.substring(1);
    s2=s2.substring(1); } }

{ lexcompare("dog","dogmatic");
  lexcompare("cannot","cant");
  lexcompare("dog","cat"); }
```

Solutions, task 1

```
/* Task 1: Comparator to compare strings ignoring
their first character (you can use
lexcompare(s1,s2), which returns true if s1 is less
than s2 in dictionary order) */

Comparator c = new Comparator() {
    public int compare(Object o1, Object o2) {
      String s1 = ((String)o1);
      String s2 = ((String)o2);
      // FIXME are s1 and s2 non-empty?
      s1=s1.substring(1);
      s2=s2.substring(1);
      if(lexcompare(s1,s2)) return -1;
      else if(s1.equals(s2)) return 0;
      else return 1;
    } };
```

Solutions, task 2

```
/* Task 2: Comparator to compare names (first last)
based on the dictionary order of the last name (you
need to discard the first word and any spaces
following the first word); */

String discard_spaces(String s) {
  while(true) {
    if(s.equals("")) return s;
    if(!s.charAt(0)==' ') return s;
    s=s.substring(1); } }

String discard(String s) {
  while(true) {
    if(s.equals("")) return ""; // FIXME what to do here?
    if(s.charAt(0) == ' ') return discard_spaces(s);
    s=s.substring(1);  } }
```

Solutions, task 2

```
Comparator c = new Comparator() {
    public int compare(Object o1, Object o2) {
      String s1 = ((String)o1);
      String s2 = ((String)o2);
      s1=discard(s1);
      s2=discard(s2);
      if(lexcompare(s1,s2)) return -1;
      else if(s1.equals(s2)) return 0;
      else return 1;
    }
  };
```