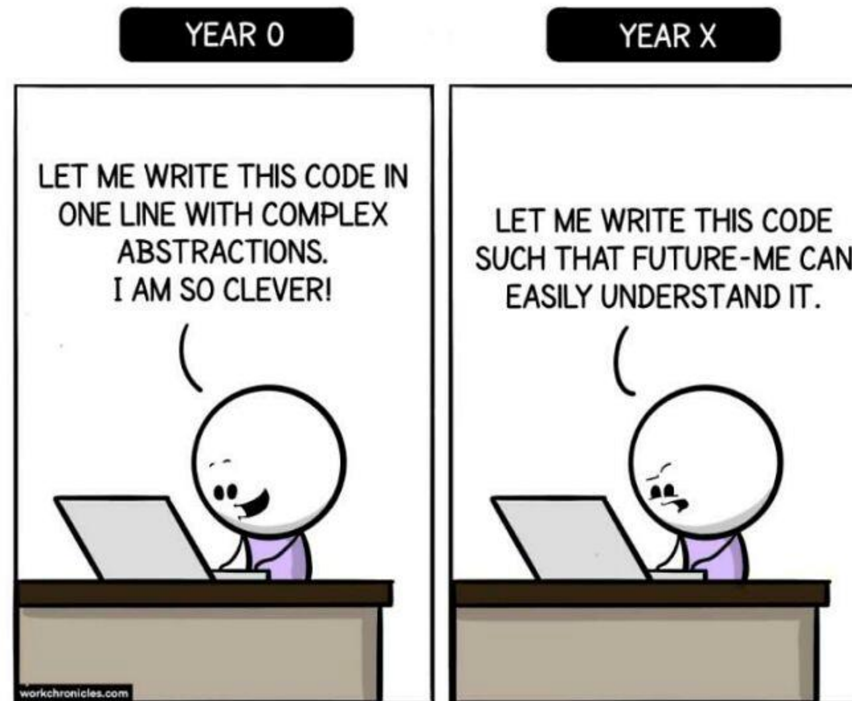
CS 142

Lecture 24: interfaces; `Comparable`



Thanks to Marty Stepp and Stuart Reges for parts of these slides

Recall: Searching

- What do we need to know about a type in order to search a list or array that contains instances of that type?

index	0	1	2	3	4	5	6	7	8
value	42	-3	17	9	12	22	6	4	72

`Collections` class

(all `List`s are a type of `Collection`)

Method name	Description
`binarySearch(`**list**`, `**value**`)`	returns the index of the given value in a sorted list (< 0 if not found)
`copy(`**listTo**`, `**listFrom**`)`	copies **listFrom**'s elements to **listTo**
`emptyList(), emptyMap(), emptySet()`	returns a read-only collection of the given type that has no elements
`fill(`**list**`, `**value**`)`	sets every element in the list to have the given value
`max(`**collection**`), min(`**collection**`)`	returns largest/smallest element
`replaceAll(`**list**`, `**old**`, `**new**`)`	replaces an element value with another
`reverse(`**list**`)`	reverses the order of a list's elements
`shuffle(`**list**`)`	arranges elements into a random order
`sort(`**list**`)`	arranges elements into ascending order

Ordering and objects

- Can we `sort` an array of Strings?
  - Operators like < and > do not work with `String` objects.
  - But we do think of strings as having an alphabetical ordering.

- **natural ordering**: Rules governing the relative placement of all values of a given type.

- **comparison function**: Code that, when given two values *A* and *B* of a given type, decides their relative ordering:
  - A < B, A == B, A > B

The `compareTo` method

- The standard way for a Java class to define a comparison function for its objects is to define a `compareTo` method.
  - Example: in the `String` class, there is a method:

```
public int compareTo(String other)
```

- A call of **A**`.compareTo(`**B**`)` will return:

a value <	0	if **A** comes "before" **B** in the ordering,
a value >	0	if **A** comes "after" **B** in the ordering,
or	0	if **A** and **B** are considered "equal" in the ordering.

Using `compareTo`

- `compareTo` can be used as a test in an `if` statement.

```
String a = "alice";
String b = "bob";
if (a.compareTo(b) < 0) {   // true
    ...
}
```

Primitives	Objects
`if (a < b) { ...`	`if (a.compareTo(b) < 0) { ...`
`if (a <= b) { ...`	`if (a.compareTo(b) <= 0) { ...`
`if (a == b) { ...`	`if (a.compareTo(b) == 0) { ...`
`if (a != b) { ...`	`if (a.compareTo(b) != 0) { ...`
`if (a >= b) { ...`	`if (a.compareTo(b) >= 0) { ...`
`if (a > b) { ...`	`if (a.compareTo(b) > 0) { ...`

compareTo and collections

- You can use an array or list of strings with Java's included binarySearch method because it calls compareTo internally.

```
String[] a = {"al", "bob", "cari", "dan", "mike"};
int index = Arrays.binarySearch(a, "dan");  // 3
```

- Java's TreeMap uses compareTo internally for ordering.

- A call to your compareTo method should return:

a value <	0	if this object is "before" the other object,
a value >	0	if this object is "after" the other object,
or	0	if this object is "equal" to the other.

Comparable

```
public interface Comparable<E> {
    public int compareTo(E other);
}
```

- A class can implement the `Comparable` interface to define a natural ordering function for its objects.

- A call to your `compareTo` method should return:

  a value < 0 if `this` object is "before" the `other` object,

  a value > 0 if `this` object is "after" the `other` object,

  or 0 if `this` object is "equal" to the `other`.

- If you want multiple orderings, use a `Comparator` instead

Comparable template

```
public class name implements Comparable<name> {

    ...

    public int compareTo(name other) {
        ...
    }
}
```

`compareTo` tricks

- *delegation trick* - If your object's fields are comparable (such as strings), use their `compareTo` results to help you:

```
// sort by employee name, e.g. "Jim" < "Susan"
public int compareTo(Employee other) {
    return name.compareTo(other.getName());
}
```

- *`toString` trick* - If your object's `toString` representation is related to the ordering, use that to help you:

```
// sort by date, e.g. "09/19" > "04/01"
public int compareTo(Date other) {
    return toString().compareTo(other.toString());
}
```

`compareTo` tricks

- *subtraction trick* - Subtracting related values produces the right result for what you want `compareTo` to return:

```
// sort by x and break ties by y
public int compareTo(Point other) {
    if (x != other.x) {
        return x - other.x;    // different x
    } else {
        return y - other.y;    // same x; compare y
    }
}
```

  - The idea:
    - if `x > other.x`, then `x - other.x > 0`
    - if `x < other.x`, then `x - other.x < 0`
    - if `x == other.x`, then `x - other.x == 0`
      - NOTE: This trick doesn't work for `doubles` (but see `Math.signum`)

Graphical User Interfaces

Why GUIs

GUI: Graphical User Interface

- write programs where the user doesn't have to interact with the console
- more practice using built in objects
- Try event-driven programming in Java

History of GUIs in Java

- **Abstract Windowing Toolkit** (**AWT**): *(JDK 1.0 - 1.1)*
  - Maps general Java code to each operating system's real GUI system.
  - *Problems:* Limited to lowest common denominator; clunky to use.
- **Swing**: *(JDK 1.2+)*
  - Paints GUI controls itself pixel-by-pixel rather than handing off to OS.
  - *Benefits:* Features; compatibility; OO design.
- **Java FX**: *(JDK 7.6+)*
  - Adds many features for more sophisticated interfaces including CSS.

GUI Parts

- **window**: A first-class citizen of the graphical desktop.
  - Also called a *top-level container*.
  - examples: frame, dialog box, applet

- **component**: A GUI widget that resides in a window.
  - Also called *controls* in many other languages.
  - examples: button, text box, label

- **container**: A logical grouping for storing components.
  - examples: panel, box