CSC207.01 2014S, Class 22: Detour: Anonymous Inner Classes

Overview

- Preliminaries.
  - Upcoming work.
  - Admin.
  - Questions.
- Anonymous inner classes - the concept.
- Anonymous inner classes - the implementation.
- Some subtleties.
- Lab.

Preliminaries

Upcoming Work.

- Homework 5 is due *next* Wednesday (March 5), but you may need two weeks to get it done.
- The exam makeup is due Sunday at 10:30 p.m. There will be no extensions.
- No reading for tomorrow.
- Lab writeup (whoops, coming soon)

Admin

- MathLAN may or may not be acting up again. Yesterday morning was backups left over from the weekend. I'm not sure what last night was, or what today will bring.
- I took off 1/4 point for incorrect tarballs. A few of you talked to me about it and I accept that for the first exam, a bit of carelessness or confusion is possible. I will restore those points if you show me that I took them off.
- Review session tonight at 7pm
- Extra credit:
  - Town hall meeting Today at noon or 7:30pm.
  - Wellness fair, around the JRC, 5:30-8:00 pm
  - CS Extras, Thursday at 4:30 p.m.: The new CS Curriculum.
  - More?

Questions

Anonymous inner classes - the concept

Sometimes you want functions and it's not worth your time/effort to name them.

```
(set! grades (map (l-s + .25) grades))
```

vs.

```
(define fixgrade
  (lambda (x) (+ .25 x)))
(set! grades (map fixgrade grades))
```

First is shorter

Giving something a name takes mental energy and pollutes the namespace.

In Java, anonymous things are also useful

- When sorting, we need a way to compare values
- When searching, we need a way to determine if we've found the desired value
  - And we may need to build values on the fly.

In Java, anonynous is particularly useful because we have to write a lot when we declare a new class.

Java has anonymous classes

We use them to provide the functions that we need above.

Anonymous inner classes - the implementation

new Interface() { Method implementations }

E.g.,

public interface Checker { public boolean okay(Object o); } // interface Checker

public class Sam { public static Object search(Object[] values, Checker check) { for (v : values) if (check.okay(v)) return v; } // search(Object[], Checker) } // class Sam

Sam.search(students, new Checker() { public boolean okay(Object o) { return o.toString.contains("k"); } // okay } // Checker);

public class SamR { public static Object searchByString(Object[] values, final String str) { Sam.search(students, new Checker() { public boolean okay(Object o) { return o.toString.contains(str); } // okay } // Checker); } } // class SamR

Some subtleties

- How do you deal with fields?
- How do you deal with fields of the enclosing object?
- How do you deal with parameters of the enclosing method?
- Can we do this with subclassing, too?