

## Class 23: Anonymous Inner Classes

**Held:** Wednesday, March 14, 2007

**Summary:** Today we consider Java's anonymous inner classes, how we create them, and why we'd want to create them.

### Related Pages:

- EBoard.

### Notes:

- Enjoy today's class.
- I should be available via email for questions on the exam.

### Overview:

- Anonymous values.
- Encapsulation and inner classes.
- Anonymous inner classes.
- Applications.

## Anonymity

- An important issue in the design of languages is that you relieve the programmer from too many bookkeeping tasks.
- One such task is the need to name intermediate or temporary values.
- For example, we'd rather write

```
z = (-b + Math.sqrt(b*b - 4*a*c))/2a;
```

than

```
double tmp1 = -b;  
double tmp2 = b*b;  
double tmp3 = a*c;  
double tmp4 = 4*tmp3;  
double tmp5 = tmp2-tmp4;  
...
```

- You've seen in Scheme that it's useful to have *anonymous functions*, functions without a name.
  - If you only use the function once, why bother naming it?
- In object-oriented languages, it is useful to have *anonymous classes*, classes without a name.

## Encapsulation and Inner Classes

- As we've said in the past, *encapsulation* is one of the key object-oriented design principles.
- It's easy to encapsulate data (fields) and capabilities (methods).
- However, when we're implementing one class with another class (e.g., lists with nodes), we may want to restrict access to the whole class to the using class.
- Inner classes provide one mechanism for doing so.

## Java's Anonymous Inner Classes

- Create an object in a new class with

```
new NameOfSuperClass() {  
    code-for-overridden methods;  
}
```

- Why? Your reasons:
  - Callbacks
  - Event handling
  - Co-routining
- Why? My reasons
  - Functional reasons, e.g., Comparators for sorting routines.
  - Alternate mechanism for conditional access to fields.
  - ...