

Nested Classes

(or "How to impress during your next job interview...")

Computer Science and Engineering • College of Engineering • The Ohio State University

Lecture 24

Introduction

Computer Science and Engineering • The Ohio State University

- So far, all our class declarations have been outermost in a .java file
 - Inside a package, which can be inside another package, etc
 - Called *top-level classes*
- Java also permits class declarations to appear within *smaller* scopes
- Recall?
 - The members of a class include: fields, methods, and *other classes*

Nested Classes

Computer Science and Engineering • The Ohio State University

- A class declared within something else (ie not at package level) is called a *nested class*
- 4 kinds of nested classes
 1. Static nested classes
 - Static members of an enclosing class
 2. Inner classes
 - Nonstatic members of an enclosing class
 3. Local classes (or local inner classes)
 - Declared inside a method, like a local variable
 4. Anonymous classes (or anonymous inner classes)
 - Declared/used at same time, nameless

Role: Helper Classes

Computer Science and Engineering • The Ohio State University

- Sometimes a class, H, is needed by *exactly one* other class, C
 - H bundles state into 1 object for C to use
 - H implements an interface that C needs to instantiate
- Example:

```
class SlowSetOfChar extends
    AbstractSet<Character> {
    private . . . //fields representing set
    public Iterator<Character> iterator () {
        //problem: can not instantiate interface
        return new Iterator<Character>();
        //ok: class that implements Iterator<Charac>
        return new MySlowIteratorOfChar();
    }
}
```
- Key point: clients of SlowSetOfChar do not need to know about MySlowIteratorOfChar class!

Example: Transcript

Computer Science and Engineering • The Ohio State University

```
/**
 * @mathmodel t : sequence of <<Q,C,W,G>>
 * @convention (exists k : dateList.length = k,
 *   courseList.length = k,
 *   creditList = k,
 *   gradeList.length = k)
 */
public class Transcript {
  private ArrayList<Quarter> dateList;
  private ArrayList<CourseNumber> courseList;
  private ArrayList<Integer> creditList;
  private ArrayList<Grade> gradeList;
  . . .
  public addEntry(Course c, Offering t, Grade g) {
    //extend all 4 lists by extracting info from c/t/g
    . . .
  }
}
```

Solution 1: Transcript

Computer Science and Engineering • The Ohio State University

```
/**
 * @mathmodel t : sequence of <<Q,C,W,G>>
 */
public class Transcript {
  private ArrayList<TranscriptLine> transcriptList;
  . . .
  public addEntry(Course c, Offering t, Grade g) {
    //extend list by extracting info from c/t/g
    TranscriptLine entry = new TranscriptLine();
    . . .
  }
}

class TranscriptLine { //one more top-level class
  Quarter Q;
  Course C;
  int W;
  Grade G;
}
```

Solution 2: Transcript

Computer Science and Engineering • The Ohio State University

```
/**
 * @mathmodel t : sequence of <<Q,C,W,G>>
 */
public class Transcript {
    class TranscriptLine { //inner class
        Quarter Q;
        Course C;
        int W;
        Grade G;
    }

    private ArrayList<TranscriptLine> transcriptList;
    . . .
    public addEntry(Course c, Offering t, Grade g) {
        //extend list by extracting info from c/t/g
        TranscriptLine entry = new TranscriptLine();
        . . .
    }
}
```

Visibility

Computer Science and Engineering • The Ohio State University

- Two choices for top level classes:
 - Public, or package-private (ie default)
- Inner classes are like any other member:
 - Public, package-private, protected, or private
- Regardless of inner class's visibility:
 - Inner class can access outer's *private* members!
 - Outer class can access inner's *private* members!
- Can be static
 - Makes it a *static nested class*

Solution 3: Transcript

Computer Science and Engineering • The Ohio State University

```
public class Transcript {
    private class TranscriptLine { //private inner class
        private Quarter Q; //same visibility as public
        private Course C;
        private int W;
        private Grade G;
    }

    private ArrayList<TranscriptLine> transcriptList;
    . . .
    public addEntry(Course c, Offering t, Grade g) {
        //extend list by extracting info from c/t/g
        TranscriptLine entry = new TranscriptLine();
        entry.G = new Grade(g);
        . . .
    }
}
```

Instantiation and Access

Computer Science and Engineering • The Ohio State University

- Typically, an inner class is private
 - Instantiate in outer class with new()
`Inner innerObject = new Inner();`
 - Outer's access of Inner: use reference
`innerObject.innerMethod();`
 - Inner's access of Outer: use (qualified) this
`g(); //Inner's g if it exists, else Outer's`
`this.g(); //same as above`
`Outer.this.g(); //Outer's g`
- Inner classes can also be public
 - Can be instantiated/used outside of Outer
`Outer outerObject = new Outer();`
`Outer.Inner innerObject = outerObject.new`
`Inner();`
`innerObject.innerMethod();`

Inner Class vs Static Nested Class

Computer Science and Engineering • The Ohio State University

- Instances of an *inner class* are always associated with a (*one!*) instance of their outer class
 - Called “enclosing instance”
 - Thus, instance of outer class must be created first
- Instances of *static nested classes* are not associated with any instances of their outer class
 - Thus, can only access static members of outer class

Good Practice: Use Static Nested

Computer Science and Engineering • The Ohio State University

- Prefer static nested classes over inner classes
- Bad rule: considering when static nested *must* be used
 - If nested class will itself have static members
 - If nested class must be accessed from outer's static methods
- Better rule: Use inner classes only if
 - Nested class needs access to *instance members* of outer class
- Otherwise, use static nested classes
 - Degenerate case: Nested class has no methods
 - Common case: Nested class methods use only arguments and nested class's fields
 - Note: There are *instances* of a static nested class!
- Clients of outer access static nested through class name

```
public class Animal {
    public static class Migration { . . . }
}
Animal.Migration x = new Animal.Migration();
```

Solution 4: Transcript

Computer Science and Engineering • The Ohio State University

```
public class Transcript {
    private static class TranscriptLine { //static nested
        private Quarter Q;
        private Course C;
        private int W;
        private Grade G;
    }

    private ArrayList<TranscriptLine> transcriptList;
    . . .
    public addEntry(Course c, Offering t, Grade g) {
        //extend list by extracting info from c/t/g
        TranscriptLine entry = new TranscriptLine();
        . . .
    }
}
```

Role: Event Handlers

Computer Science and Engineering • The Ohio State University

- Recall roles for H and C
 - H bundles state into 1 object for C to use
 - H implements an interface that C needs to instantiate
- Common example of #2: Event handlers
 - More general description: “call-backs”
- Recall Swing components and listeners
 - Event handlers implement an interface

```
interface ActionListener {
    void actionPerformed (ActionEvent e);
}
```
 - Component has a method for registering a listener

```
public abstract class AbstractButton {
    void addActionListener (ActionListener l)
}
```

Example: ActionListener

Computer Science and Engineering • The Ohio State University

```
public class SimpleWindow extends JFrame {
    public SimpleWindow() {
        . . .
        Button test = new Button();
        BHandler handler = new BHandler();
        test.addActionListener(handler);
        setVisible(true);
    }

    private static class BHandler implements
    ActionListener {
        public void actionPerformed(ActionEvent event) {
            JOptionPane.showMessageDialog(null,
                "You pressed: " + event.getActionCommand());
        }
    }
}
```

Example: ActionListener

Computer Science and Engineering • The Ohio State University

```
public class SimpleWindow extends JFrame {
    public SimpleWindow() {
        . . .
        Button test = new Button();
        //common idiom: anonymous object
        test.addActionListener(new BHandler());
        setVisible(true);
    }

    private static class BHandler implements
    ActionListener {
        public void actionPerformed(ActionEvent event) {
            JOptionPane.showMessageDialog(null,
                "You pressed: " + event.getActionCommand());
        }
    }
}
```

Anonymous Classes

Computer Science and Engineering • The Ohio State University

- Simultaneous declaration and use
 - Occur within an *expression*
 - Usually an argument in a method call
`test.addActionListener(/*here*/);`
- Anonymous class has no class name
 - Can not use as declared type
`AnonClass anObject = new AnonClass();`
 - Instead, use some other (named) type, and have anonymous class subtype it
`SomeInterface anObject = new AnonClass();`
 - Replace constructor name with declaration
`SomeInterface anObject = new SomeInterface() {
 public void methodName() { . . . }
};`
- Result is either
 - Compact clean code, or
 - Dense impenetrable code

Anonymous ActionListener

Computer Science and Engineering • The Ohio State University

```
public class SimpleWindow extends JFrame {
    public SimpleWindow() {
        . . .
        Button test = new Button();
        //anonymous class
        test.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                JOptionPane.showMessageDialog(null,
                    "You pressed: " + event.getActionCommand());
            }
        });
        setVisible(true);
    }

    //no need for an inner class!
}
```

Example of Anonymous Class

Computer Science and Engineering • The Ohio State University

- In java.util:

```
public class Arrays {
    public static <T> void sort (T[] a, Comparator<T> c)
    {
        . . .
    }
}
interface Comparator<T> {
    int compare (T o1, T o2);
}
```

- In client code somewhere:

```
Arrays.<String>sort (args, new Comparator<String>() {
    public int compare (String s1, String s2) {
        return s1.length() - s2.length();
    }
});
```

Compilation

Computer Science and Engineering • The Ohio State University

- Source (.java) --> byte code (.class)

- Example:

```
$ javac Classname.java
```

- Produces:

```
Classname.class
```

- If class Outer contains a nested class, Nested, *two* class files are produced

- Example:

```
$ javac Outer.java
```

- Produces:

```
Outer.class    Outer$Inner.class
```

Good Practice: Use Sparingly

Computer Science and Engineering The Ohio State University

- ❑ Proper use makes code smaller and cleaner
- ❑ Improper use makes code hard to understand
- ❑ Stick with basic patterns:
 - Bundling state (static nested)
 - Adaptors (inner)
 - Event handlers (inner or anonymous)
 - ❑ ie Call-backs (inner or anonymous)
 - ❑ ie Single-method interface implementations (inner or anonymous)
 - Avoid local classes altogether (very rare)

Summary

Computer Science and Engineering The Ohio State University

- ❑ Four kinds of nested classes
 - Static nested, inner, local, anonymous
- ❑ Mutual access of private members
- ❑ Static vs inner:
 - Inner have enclosing instance
- ❑ Anonymous classes declared & used at same time
- ❑ Use: helper class used by 1 other class
 - Bundle state
 - Instantiate interface
- ❑ Commonly encountered "interface instantiation"
 - Event handlers (Swing)
 - Thread creation
 - Iteration