

CS 535 Object-Oriented Programming & Design
Fall Semester, 2000
Doc 16 Nested Classes & Exception Lectures Examples
Contents

Nested Classes & Exception Lecture Examples.....	2
Scope 2 Example	3
Scope 2a Example	7
Scope 3 Example	9
Scope 4 Example	10
Scope 5 Example	11
Scope 6 Example	12
Scope 7 Example	13
Exception Example.....	15
Exception 2 Example.....	16
Exception 3 Example.....	17

Nested Classes & Exception Lecture Examples

These are the examples that Dr. Eckberg used when he lectured in CS 535 while I was out of town. For more examples using Nested and Inner classes see:

<http://www.eli.sdsu.edu/courses/fall98/cs596/notes/nested/nested.html>

Scope 2 Example

This example explores both static and non-static inner classes. Like other class components, a static inner class is the property of the class and not objects of that class. If the phrase 'class var.' is correct for a static data member, then a static inner class must be referred to as a 'class class' (hehehe). A non-static inner class object can only be created thru an instance of the containing class--it belongs to objects of the containing class. This applies at run time to creation attempts from an outer class method. It does not apply to creation of data members of the outer class at 'definition time'.

```
public class scope2 {  
    static int j = 4, k = 9;  
    int p = 14;
```

```
    //creates references  
    staticInner sinn3, sinn4;  
    inner motel = new inner();
```

```
    // creates reference only, presumably  
    inner motel2;
```

```
    static inner motel3 = new inner(); // illegal,
```

```
    // Next two lines show
```

```
    // that a static instance of a member instance can be created.
```

```
    static scope2 sc1 = new scope2(); // legal  
    static inner motel6 = sc1.new inner();
```

Scope 2 Example - Continued

```
public static void main( String [] aa) {  
    int i = 7;  
    int k = 5;  
    scope2 sc2 = new scope2();  
    staticInner sinn = new staticInner(); // static inner class instance  
  
    sinn.foop();  
  
    // an outer class instance is needed to  
    // invoke the inner class's 'new' method and constructor  
    // An inner class cannot have a static member or method  
    inner inn = sc2.new inner();  
    inn.foo();  
    sc2.print();  
    motel6.foo();  
    System.out.println( "inner class k is " + inn.k + " block k is " + k);  
} //main  
  
public void print() {  
    motel.foo();  
}
```

Scope 2 Example - Continued

```
class inner {  
    // outer class static var can be redefined  
    int n = 10; int k = 8;  
    public void foo() {  
        // can access the outer static k with a classname modifier  
        System.out.println("outer k is " + scope2.k);  
        System.out.println("outer static j is " + j + "outer p is " + p);  
  
        // the outer class can access an inner class method, and an inner  
        // class can access an outer class "class variable", which is within  
        // scope, with out a classname modifier; ditto for an outer class  
        // instance variable! thus at 'run time' an 'enclosing' outer class  
        // object must be available; This ability of an inner class to refer to  
        // outer instance vars. forces each inner class object to be  
        // assoc. with an outer class object, one way or another  
    }  
} //inner
```

Scope 2 Example - Continued

```
static class staticInner{
    int q = 15;
    int u = 22;
    void foop() {
        System.out.println(p); //this is illegal; cannot refer to instance
                               //variable within a static class
        System.out.println(" inner static foop prints outer static j " + j);
        // static var.
    } //foop
} //staticInner
}
```

A static inner class, if thought of as a 'record', makes sense in that one can have multiple occurrences of that record as a 'part' of the static components of the containing class, but getting definitional 'copies' of the inner static class. The merit of being able to get dynamic 'copies' is very unclear, and inconsistent with the usual concept of 'static'. Does one get multiple copies????

Scope 2a Example

This example explores static inner classes. Like other class components, a static inner class is the property of the class and not objects of that class. If the phrase 'class var.' is correct for a static data member, then a static inner class must be referred to as a 'class class' (hehehe). Of course there really is a Class class. :(

```
public class scope2a {
    static int j = 4,k = 9;
    int p = 14;
    staticInner sinn3,sinn4; //creates references
    static staticInner sinn5 = new staticInner();

    public static void main( String [] aa) {
        int i = 7;
        int k = 5;
        scope2a sc2 = new scope2a();
        staticInner sinn = new staticInner(); // static inner class instance
            // What does that mean????
        sinn.foop();
        // sinn3 is not static, but main() is, so we need an instance of scope2a
        sc2.sinn3 = new staticInner();
        sc2.sinn3.foop();
        sinn5.foop(); //sinn5 is static data member, i.e. a 'class' member

        //sc2.print();
    }
}
```

Scope 2a Example Continued

```
static class staticInner{
    int q = 15;
    int u = 22;
    void foop() {
        //System.out.println(p); //this is illegal; cannot refer to instance
        //variable within a static class
        System.out.println(" inner static foop prints outer static j " + j);
        // static var.
    }
}
}
```


Scope 3 Example

Explores inner class method syntax; inner classes have parents too; examines how to reference parental stuff, shadowed or not.

```
class A {int a = 11;}
class B {int a = 12;}
public class scope3 extends A {
    int k = 33;
    int a = 55;
    public static void main(String [] aa) {
        scope3 sc3 = new scope3();
        inner in = sc3.new inner();
        in.innerMethod();
    }

    class inner extends B {
        int a = 44;
        public void innerMethod() {
            System.out.println("inner a is " + a);
            System.out.println("parent a is " + super.a) ;

            // scope3.a would work if a was static in the outer class
            System.out.println("outer a is " + scope3.this.a + sc3.a );
            System.out.println("outer parent a is " + scope3.super.a);
        }
    }
}
```

Scope 4 Example

Explores access to inner class from outside the outer class

```
class outer {  
    int a = 3;  
    void outfoo() {System.out.println(a);}  
    class inner {  
        int a = 4;  
        void innfoo() {System.out.println(a);}  
    }  
}
```

```
public class scope4 {  
    public static void main(String [] aa) {  
        outer out = new outer();  
        outer.inner in = out.new inner(); in.innfoo();  
        // the outer class is initially anonymous in the next line  
        outer.inner in2 = (new outer()).new inner(); in2.innfoo();  
        System.out.println(in.a + " " + out.a);  
        // the 'a' in the outer part of the 'in2' instance might be hard to  
        // get at. The associated outer instance must be there, but it is  
        // hard to get to.  
    }  
}
```

Scope 5 Example

This example looks at local inner classes, which means classes that are declared as internal to a method. These classes can be named or anonymous, in this case named. One special syntax restriction is that variables local to the method, or parameters of the method, must be final in order to be accessed by the inner class. Local inner classes can not be static. Local inner classes cannot have access modifiers. Error messages for these violations can be less than perspicuous

```
import java.io.*;
public class scope5 {
    int a = 5;
    public static void main(String [] aa) {
        scope5 s = new scope5();
        s.foo(s.a,9,3000);
    }
    // in the call above, a var actual param is passed to a const
    // formal parameter b, and this is ok
    public void foo(final int b, final int c, int k) {
        int d = 10;
        final int e = 12;
        class LocalInner {
            void goo() {System.out.println (b + c + e);}
        }
        LocalInner li = new LocalInner(); // li has a longer lifetime than
        // any particular invocation of 'foo'; constants like b,c,e can
        // more easily be made to hang around for as long as a particular
        // li is alive; this is a scope-lifetime issue
        li.goo();
    }
}
```

Scope 6 Example

This example explores anonymous inner classes; these are local classes and as such have the usual restrictions--they cannot be static and cannot have access modifiers. An anonymous inner class can either extend a class, or implement an interface. We try to give both examples here.

In `new FaceOff { }` the stuff in the braces is an anon. class def. which implements `FaceOff` without using the word 'implements'; this is a dumb example, intended to show the syntax in `new ClassLess(666) { ... }`, the braces stuff is the anon. inner class, and the 666 is an argument for the superclass constructor (the superclass being `ClassLess`); the inner class extends `ClassLess` without using 'extends'

```
interface FaceOff {void foo();}
```

```
class ClassLess {  
    int zero = 6;  
    public ClassLess (int i) {zero = i;}  
    public ClassLess(){this(0);}  
}
```

```
public class scope6 {  
    public static void main(String [] aa) {  
        // the compiler ralph's if the word 'final' goes bye-bye  
        final int two = 3;  
        new FaceOff() {  
            public void foo(){System.out.println("You hockey puck!");}.foo();  
        }  
        new ClassLess(666) {int lucky = 7;  
            void moo() {System.out.println(lucky + zero + two);}.moo();  
        }  
    }  
}
```

Scope 7 Example

This program modifies Fram6.java to use the swing set. Most of the swing set is not 'peer-based'. This means that instead of translating java stuff into host machine window stuff, java just draws pictures onto blank host machine windows. This takes more time, but makes the look and feel of swing set applications much more platform independent. JFrame is actually a subclass of Frame. And JPanel is a subframe of JComponent, which is a subclass of Component. This program opens two frames of the same kind, and manages them by not shutting down until both frames are closed. The window closing event notifies the main method of a window closing. Thus 'main' acts as a window manager. The windows are treated identically by draw methods, i.e. by paint().

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class scope7 extends JFrame{
    static int i = 0;
    public scope7() {
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                dispose();
                scope7.dec();
            }
        });
        // the best way to place things on a JFrame is to create a contentPane,
        // which is like an overlay of the JFrame, and place stuff there!
        // The 'stuff' is usually an extension of a JPanel. So a JPanel goes
        // on a contentPane which overlays a JFrame
        Container contentPane = getContentPane();
        contentPane.add(new myPanel());
    }
```

Scope 7 Example Continued

```
public static void main(String args[]) throws InterruptedException{
    scope7 f = new scope7();
    i++;
    f.setTitle("ducks");
    f.setBounds(20,20,300,100);
    f.show();
    scope7 f2 = new scope7();
    i++;
    f2.setTitle("geese");
    f2.setBounds(350,20,200,200);
    f2.show();
    while (i > 0) {
        System.out.println("i in loop is " + i);
        Thread.sleep(1000);
    }
    System.exit(0); // needed because threads exist
}

public static void dec() {i--;}
}

class myPanel extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g); // this replaces 'paint' with JFrames
        g.drawString("brouhaha",100,50);
    }
}
```

Exception Example

Class Throwable has kids Error and Exception. Exception has the kid RuntimeException, with kids like ArithmeticException and NullPointerException. These are 'unchecked'.

User defined kids of Exception are known as 'checked' exceptions. There are also predefined kids of Exception, like IOException, which are also 'checked'.

```
public class exception {  
    public static void main(String [] aa) {  
        try { throw new NegException(new Integer(-17));  
        }  
        catch (NegException e) {  
            System.err.println(e);  
            System.err.println("The number was " + e.data.intValue());  
        }  
    }  
}  
  
class NegException extends Exception {  
    public Integer data;  
    NegException(Integer d){  
        super("some !@#$! sent me a negative number");  
        data = d;  
    }  
}
```

Exception 2 Example

The assumption below is that main does not want to deal with a `NegException`, but is handling it for, maybe, debugging reasons

```
public class exception2 {
    public static void main(String [] aa) {
        try {sub();}
        catch (NegException e) {
            System.err.println("Don't shine this exception on to me!");
        }
    }

    public static void sub () throws NegException {
        try { throw new NegException(new Integer(-17));
        }
        catch (NegException e) {
            System.err.println(e);
            System.err.println("The number was " + e.data.intValue());
            throw new NegException(e.data);
        }
    }
}

class NegException extends Exception {
    public Integer data;
    NegException(Integer d){
        super("some !@#$! sent me a negative number");
        data = d;
    }
}
```


Exception 3 Example

The assumption below is that the error object is to be passed up the food chain, with a little more information added each time until the right guy for handling the error is reached. 'sub' makes the initial throw, and would normally do so after, maybe, acquiring a negative value from a user

```
public class exception3 {  
    public static void main(String [] aa) {  
        try {sub();}  
        catch (NegException e) {  
            System.err.println("In main: " + e.ego);  
            System.err.println(e);  
            System.err.println("The offensive data was " + e.data.intValue());  
        }  
    }  
}
```

```
public static void sub () throws NegException {  
    try { throw new NegException("",new Integer(-17));  
    }  
    catch (NegException e) {  
        throw new NegException("In sub: ",e.data);  
    }  
}  
}
```

```
class NegException extends Exception {  
    public Integer data;  
    public String ego = "";  
    NegException(String s,Integer d){  
        super("some !@#$! sent me a negative number");  
        data = d;  
        ego = s + ego;  
    }  
}
```