

CMSC 132: Object-Oriented Programming II



Java Inner Classes

Department of Computer Science
University of Maryland, College Park

1

Overview

- **Classes**
 - Top-level vs. inner & nested
- **Inner classes**
 - Iterator example
 - Used inside outer class
 - Used outside outer class
- **Anonymous inner classes**
 - Syntax
 - Uses for GUIs
- **Nested classes**

2

Java Classes

- **Top level classes**
 - Declared inside package
 - Visible throughout package, perhaps further
 - Normally declared in their own file
 - Public classes must be defined in their own file
 - Not required for other classes
- **Inner and nested classes**
 - Declared inside class (or method)
 - Normally used only in **outer** (enclosing) class
 - Can have wider visibility

3

Inner / Nested Classes

- **Inner class**
- **Nested class**
- **Anonymous inner class**
- **Examples**

```
public class MyOuterClass {  
    public class MyInnerClass { ... }  
    static public class MyNestedClass { ... }  
    Iterator iterator() { return new Iterator() { ... } }  
}
```

4

Inner Classes

- **Description**
 - Class defined in scope of another class
 - May be named or anonymous
- **Useful property**
 - Outer & inner class **can directly access** each other's fields & methods (even if private)

5

Inner Classes

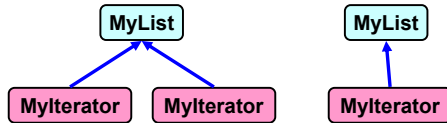
- **Example**

```
public class OuterClass {  
    private int x;  
    private class InnerClass {  
        private int y;  
        void foo() { x = 1; } // access private field  
    }  
    void bar() {  
        InnerClass ic = new InnerClass();  
        ic.y = 2; // access private field  
    }  
}
```

6

Inner Class Link To Outer Class

- Inner class **instance**
 - Has association to an instance of outer class
 - Must be instantiated with an enclosing instance
 - Is **typed** to outer class object at moment of creation (can not be changed)



7

Inner Classes

- Useful for
 - Private helper classes
 - Logical grouping of functionality
 - Data hiding
 - Linkage to outer class
 - Inner class object **typed** to outer class object
- Examples
 - **Iterator** for Java Collections
 - **ActionListener** for Java GUI widgets

8

Iterator Example

- **MyList**

```
public class MyList {
    private Object [] a;
    private int size;
}
```
- Want to make **MyList** implement **Iterable**
 - Skipping generic types at the moment
 - Need to be able to return an **Iterator**

9

(Problematic) MyIterator Design

```
public class MyIterator implements Iterator {
    private MyList list;
    private int pos;
    MyIterator(MyList list) {
        this.list = list;
        pos = 0;
    }
    public boolean hasNext() {
        return pos < list.size;
    }
    public Object next() {
        return list.a[pos++];
    }
    ...
}
```

10

MyIterator Design

- **Problems**
 - Need to maintain reference to **MyList**
 - Need to access **private** data in **MyList**
- **Solution**
 - Define **MyIterator** as inner class for **MyList**
 - Instance of **MyIterator** tied to instance of **MyList**
 - **MyIterator** methods can access **private** **MyList** fields
 - **MyIterator** objects can iterate through elements of **MyList**

11

(Successful) MyIterator Design

```
public class MyList implements Iterable {
    private Object [] a;
    private int size;
    public Iterator iterator() {
        return new MyIterator();
    }
    public class MyIterator implements Iterator {
        private int pos = 0;
        public boolean hasNext() { return pos < size; }
        public Object next() { return a[pos++]; }
        ...
    }
}
```

12

Instantiating Inner Class

- How to access instance of inner class?
- Common gimmick
 - Outer class method returns instance of inner class
 - Used by Java Collections Library for Iterators
- Example code

```
public class MyList {
    public class IC implements Iterator { ... }
    public Iterator iterator() {
        return new IC(); // creates instance of IC
    }
}
MyList m = new MyList();
Iterator it = m.iterator();
```

13

Using Inner Class *Inside* Outer Class

- Assume class OC defines an inner class IC
- Inside methods of OC, just use IC normally
 - Use `a = new IC();` to create instances of IC
 - `a` is associated with the specific IC object referenced by `this`

14

Using Inner Class *Inside* Outer Class

- Code

```
public class OC { // outer class
    private int x = 2;
    private class IC { // inner class
        private int y = 4;
        private int getSum() { return x + y; }
    }
    void bar() {
        IC z = new IC(); // create new IC assoc. w/ this
        z.getSum(); // treat z like normal object
    }
}
```

15

Using Inner Class *Outside* Outer Class

- Assume class OC defines an inner class IC
- Outside of OC, use `OC.IC` to name inner class
 - Use `a = b.new IC();` to create instances of IC
 - `a` is associated with a specific OC object `b`
 - It is very rare for you to need to do this

16

Using Inner Class *Outside* Outer Class

- Code

```
public class OC { // outer class
    private int x = 2; // don't forget private
    public class IC { // inner class
        int z = 4;
        public int getSum() {
            return x + z;
        }
    }
}
```

17

Using Inner Class *Outside* Outer Class

- Class referencing syntax
 - `OuterClass.InnerClass`
- Example

```
OC b = new OC();
OC.IC a; // name of inner class instance
a = b.new IC(); // instantiates inner class
// a = new OC.IC() wrong! Need instance of OC (b)
// a now will "know about" b, but not vice versa

a.getSum() yields 6 // can access private x in b
```

18

Accessing Outer Scope

■ Code

```
public class OC {           // outer class
    int x = 2;
    private class IC {     // inner class
        int x = 6;
        private void getX() { // inner class method
            int x = 8;
            System.out.println( x );           // prints 8
            System.out.println( this.x );      // prints 6
            System.out.println( OC.this.x );   // prints 2
        }
    }
}
```

19

Method Invocations

■ Method invocations on inner class

- Can be transparently redirected to outer instance

■ Resolving method call on unspecified object

1. See if method can be resolved on inner object
2. If not, see if method can be resolved on corresponding instance of outer object
3. If nested multiple levels, keep on looking

20

Anonymous Inner Class

■ Description

- Inner class without name
- Defined where you create an instance of it
 - In the middle of a method
 - Returns an instance of anonymous inner class
- Useful if the only thing you want to do with an inner class is create instances of it in one location

■ Syntax

```
new ReturnType() { // unnamed inner class
    body of class... // implementing ReturnType
};
```

21

Anonymous Inner Class

■ Code

```
public class MyList {
    public Iterator iterator() {
        return new Iterator() { // unnamed inner class
            ... // implementing Iterator
        };
    }
}
MyList m = new MyList();
Iterator it = m.iterator();
```

22

Example Anonymous Inner Classes

■ Use

```
new Foo() {
    public int one() { return 1; }
    public int add(int x, int y) { return x + y; }
};
```

■ To define an anonymous inner class that

- Extends class Foo / implements interface Foo
- Defines methods one & add

23

MyList With *Explicit* Inner Class

■ Code

```
public class MyList implements Iterable {
    private Object [] a;
    private int size;
    public Iterator iterator() {
        return new MyIterator();
    }
    public class MyIterator implements Iterator {
        private int pos = 0;
        public boolean hasNext() { return pos < size; }
        public Object next() { return a[pos++]; }
    }
}
```

24

MyList With Anonymous Inner Class

■ Code

```
public class MyList implements Iterable {
    private Object [] a;
    private int size;
    public Iterator iterator() {
        return new Iterator () {
            private int pos = 0;
            public boolean hasNext() { return pos < size; }
            public Object next() { return a[pos++]; }
        };
    }
}
```

25

Support For Java GUIs

■ Graphical User Interface (GUI)

- Java AWT & Swing libraries

■ Event-driven programming model

- Components may generate events
 - E.g., `ActionEvent`, `KeyEvent`, `MouseEvent`
- Requires **event listeners** to handle event

■ Event listeners frequently implemented using anonymous classes

- Used only in one location
- Implements event listener interfaces

26

Using Inner Classes in GUIs

```
javax.swing.SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        createAndDisplayGUI();
    }
});

button.addActionListener (new ActionListener() {
    public void actionPerformed (ActionEvent evt) {
        System.out.println("Button pushed");
    }
});
```

27

Nested Class

■ Description

- Similar to inner class, but declared as **static class**
- No link to an instance of the outer class
- Can directly access outer class fields & methods
- Useful if inner class object
 - Associated with different outer class objects
 - Survives longer than outer class object

■ Example

```
class LinkedList {
    static class Node { Node next; }
    Node head;
}
```

28

Summary of Inner / Nested Classes

■ All inner / nested classes

- Defined inside another class
- Can access private members of enclosing class

■ Inner class

- Each instance of an inner class is transparently associated with an instance of the outer class

■ Anonymous inner class

- Unnamed inner class defined & used in one place

■ Nested class

- Defined as static class

29