Inner Classes

Mark Allen Weiss

Outline

- **History of inner classes**
- **Nested (static inner) classes**
- **The instance inner class and iterator pattern**
- **Inner classes inside of functions**
- **Anonymous inner classes**

History Lesson

- **Java design removes fringe C++ features**
- **C++ has nested classes to hide types:**

```
class List
{
    ...
  private:
    class Node
    {
      public:
        int data;
        Node *next;
        Node( int d = 0, Node *n = NULL )
          : data( d ), next( n ) { }
    };
};
```


Java 1.0

- **No nested classes in Java 1.0**
- **C++ nesting style achieved with packages**
  - **Can make `Node` class package private**
- **Speculation was that nested classes would needlessly complicate language**

Java 1.1

- **In Java 1.1, nested classes were added**
- **Needed especially to implement patterns associated with AWT event handling**
- **Also useful for several other patterns**
- **Designers went crazy and added lots of features**

What's Added

- **Nested classes (static inner classes), mimic C++ style of nesting a type inside of type**
- **Inner classes (new in Java), allows classes inside of classes and implies an association of objects of the inner class with an object of the outer class**
- **Classes inside of functions**
- **Classes inside of expressions**
- **Classes with no names**

Language Changes Required

- **Added lots of syntax for inner classes**
  - **lots of syntax rules for corner cases and silly code**
- **In addition to direct inner class syntax, added some new syntax as a result**
  - **final local variables**
  - **final parameters**
  - **instance initializers**
- **Java 1.1 compiler generates code for inner classes that can be run on a Java 1.0 VM**

Terminology

- **Use terms outer and inner class to refer to the two classes**
- **Nested classes are (static) inner classes**
- **Can have classes inside of classes inside of classes if we want**

Visibility Rules

- **Inner classes are members of the outer class**
  - **can be declared with any visibility**
  - **can see all members of the outer class**
  - **is in same package as outer class: implies that outer class can access non-private inner class members**
  - **class name is**
    - **Inner (from outer class)**
    - **Outer.Inner (if visible, from rest of world)**
- **Typical pattern:**
  - **declare inner class private**
  - **declare inner class members public or package visible**

Linked List

- **Hide the `Node` class inside of `MyList` class**

```
public class MyList
{
  private Node front = null;
    ...
  private static class Node
  {
    Object data;
    Node next;

    Node( Object d, Node n )
      { data = d; next = n; }
  }
}
```

Public Nested Class

- **Occasionally nested class is public**
  - **Done just for convenience**
  - **Example is `java.io.ObjectInputStream.GetField`**
- **Convention of lower case for package and upper case for classes makes it easy to see what class and package are**
- **If nested class has public constructor, can create with**

```
Outer.Inner in = new Outer.Inner( ... );
```

  - **Often created via factory method instead**

Access of Outer Class Members

- **Static inner class objects can**
  - **access static members of the outer class**
  - **access instance members of the outer class only through a reference to the outer class**
  - **rules similar to what static methods can access**
- **If inner and outer class have name clash**
  - **closest class name wins**
  - **access outer member explicitly with Outer.member or Inner.member**

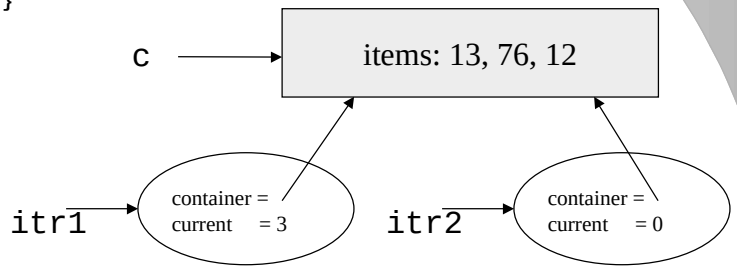
Instance Inner Class Motivation

- **Consider container/iterator interaction**

```
package pack;
public class MyContainer
{
  Object[] items;
  public Iterator iterator( )
    { return new MyContainerIterator( this ); }
     ...
}
class MyContainerIterator implements Iterator
{
  private int current = 0;
  private MyContainer container;
  public MyContainerIterator( MyContainer c )
    { container = c; }
  public boolean hasNext( )
    { return current < container.items.length; }
    ...
}
```


Using The Container/Iterator

```
import pack.MyContainer;
public class Demo {
  public static void main( String[] args ) {
    MyContainer c = new MyContainer( );
      ...  // populate c via adds
    Iterator itr1 = c.iterator( );
    Iterator itr2 = c.iterator( );
    while( itr1.hasNext( ) )
      System.out.println( itr1.next( ) );
  }
}
```



Observations

- **`items` array in `MyContainer` is not private**
- **`MyContainerIterator` is really part of `MyContainer`**
- **`MyContainerIterator` is package visible; private would be better; only use interface**
- **Following the container reference in `MyContainerIterator` is a little sloppy**
- **Every instance of `MyContainerIterator` is associated with exactly one `MyContainer`**

Revised With Inner Classes

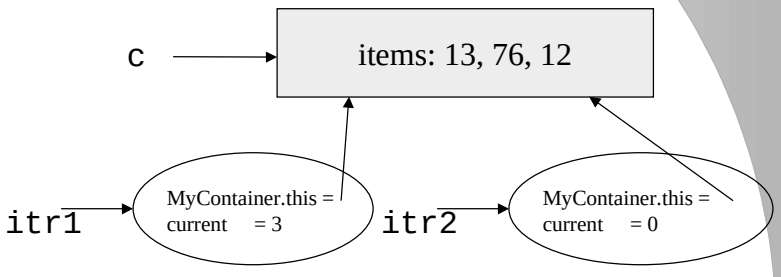
```
package pack;
public class MyContainer
{
  private Object[] items;
  public Iterator iterator( )
    { return new MyContainerIterator( ); }
     ...

  private class MyContainerIterator implements Iterator
  {
    private int current = 0;

    public MyContainerIterator( )
      { }
    public boolean hasNext( )
      { return current < items.length; }
      ...
  }
}
```

Benefits

- **`items` array is now private**
- **`MyContainerIterator` is now private**
- **Code is cleaner because `container` reference has mysteriously vanished and it appears we can access outer class members directly**
- **But what happened to that reference?**

Implicit Outer Reference

- **Instance inner class objects always have an implicit reference to the outer class object that caused its creation**
  - **Name is Outer.this**
  - **Can be omitted if no conflict with inner name**



Consequences of Implicit Outer Ref

- **Garbage collector will not reclaim an outer object if an inner object is around**
- **If an inner instance class is public enough, the following would not work:**

```
MyContainer.MyContainerIterator obj =
                        new MyContainer.MyContainerIterator();
```

  - **even though constructor appears to exist**
  - **no way to initialize outer reference in this code**
- **Cannot create inner instance class from *`static`* outer class method:**

```
  // Cannot do this inside static method of MyContainer
MyContainerIterator obj = new MyContainerIterator();
```


Inner Classes Complicate Things

- **Previous slide illustrates problem**
  - **Need syntax to create inner class from outside outer class**
  - **Have to be able to propagate a reference to some outer object**
  - **Code is stupid**
    - **Case 1: inner class should have been private and this wouldn't be an issue**
    - **Case 2: use an instance method**
  - **But language designers need a rule for all cases**

The Ugly Solutions

- **Outer reference invokes new as if it were a method:**

```
MyCollection c = new MyCollection( );
          ...
MyCollection.MyCollectionIterator = c.new MyCollectionIterator( );
```

- **Similar nonsense for extending an inner instance class:**

```
class Stupid extends Inner.Outer // Inner.Outer is instance inner class
{
  public Stupid( Outer obj )
  {
    obj.super( );
      ...
  }
}
```


Inner Classes With Factory Classes

- **If factory objects can never be created, inner classes and methods must of course be static:**

```
public interface Foo { ... }

public class FooFactory
{
  private static FooImpl { ... }
  private static FooProxy { ... }

  public static Foo allocateFoo( )
    { return new FooProxy( new FooImpl( ) ); }

  private FooFactory( ) { } // No FooFactory objects
}
```

Inner Classes

- **Can be static (of course)**
- **Can be final (not legal to extend)**
- **Can be abstract**
- **Can be interfaces**
- **Can define static final fields but no other statics**

Interfaces and Inner Classes

- **Same Outer.Inner used**
- **Inner interfaces of a class are legal**
- **Inner interfaces are always implicitly static**
- **Inner classes of an interface are legal**
- **Inner classes of an interface are always implicitly static**

Inner Class Files

- **Compiler:**
  - **generates a unique class file for each inner class.**
  - **Generally uses Outer$Inner.class as the class file**
  - **Internally generates a hidden implicit reference to outer class**
  - **Generates package private wrappers to access each accessed private member of the outer class**
    - **names such as access$001 (convention changed in Java 1.3)**
    - **can invoke with reflection**
    - **VM can verify that method only invoked from outer class, but doesn't right now**

Classes Inside of Methods

- **Can declare classes inside of methods**
- **Cannot use static**
  - **class is static if method is static**
  - **class is instance if method is instance**
- **Cannot use private**
  - **meaningless; class is not visible outside of the function**
- **Can also access the parameters of the enclosing method and local variables that are in scope**
  - **variables must be final**

Final Local Variables

- **Added in Java 1.1**
- **Value cannot change (for references, only value of reference can't change; object state can)**
- **Can only be initialized once (parameters already are); compiler does usual flow analysis**
- **Final variables (primitives and references) not reclaimed by GC until all anonymous inner objects in the method are reclaimed.**
  - **Implies that these variables are not on the stack**
  - **So not all local variables are on the stack**

Anonymous Inner Classes

- **Can create a class with no name**
- **Class implements an interface or extends an existing class**
- **Brutal syntax**
- **Useful for implementing short function objects**

Original Example From Interfaces Module With Function Object

```
class OrderRectByWidth implements Comparator {
  public int compare( Object obj1, Object obj2 ) {
    Rectangle r1 = (Rectangle) obj1;
    Rectangle r2 = (Rectangle) obj2;
    return( r1.getWidth() - r2.getWidth() );
  }
}

class Demo {
  public static void main( String [] args ) {
    Object [ ] rects = new Rectangle[ ] { ... };
    Object max = findMax( rects, new OrderRectByWidth( ) ) );
    Rectangle widest = (Rectangle) max;
       ...
  }
}
```


With Static Inner Class

```
class Demo {

  private static class OrderRectByWidth implements Comparator {
    public int compare( Object obj1, Object obj2 ) {
      Rectangle r1 = (Rectangle) obj1;
      Rectangle r2 = (Rectangle) obj2;
      return( r1.getWidth() - r2.getWidth() );
    }
  }

  public static void main( String [] args ) {
    Object [ ] rects = new Rectangle[ ] { ... };
    Object max = findMax( rects, new OrderRectByWidth( ) ) );
    Rectangle widest = (Rectangle) max;
      ...
  }
}
```

With Class Inside Of Method

```
class Demo {
 public static void main( String [] args ) {
    Object [ ] rects = new Rectangle[ ] { ... };

    class OrderRectByWidth implements Comparator {
      public int compare( Object obj1, Object obj2 ) {
        Rectangle r1 = (Rectangle) obj1;
        Rectangle r2 = (Rectangle) obj2;
        return( r1.getWidth() - r2.getWidth() );
      }
    }

    Object max = findMax( rects, new OrderRectByWidth( ) ) );
    Rectangle widest = (Rectangle) max;
      ...
  }
}
```


With Anonymous Inner Class

```
class Demo {
 public static void main( String [] args ) {
    Object [ ] rects = new Rectangle[ ] { ... };

    Object max = findMax( rects, new Comparator( ) {
        public int compare( Object obj1, Object obj2 ) {
          Rectangle r1 = (Rectangle) obj1;
          Rectangle r2 = (Rectangle) obj2;
          return( r1.getWidth() - r2.getWidth() );
        }
      }
    ) );
    Rectangle widest = (Rectangle) max;
      ...
  }
}
```

Anonymous Inner Class Hints

- **Classic signature will have lots of parentheses, semicolons, braces at the end**
  - **very hard to type**
- **Write the class outside first.**
- **After the `()` after the call to `new`, add some blank lines**
- **Replace `new Implementation` with `new Interface`**
- **Cut and paste the class declaration after the `()` that follow the call to `new`, and indent a little**

Anonymous Class Design

- **Useful for class that adds one short function**
- **Adding lots of stuff defeats readability**
- **Useful for accessing local variables that would be impossible without inner classes**

Anonymous Class Names

- **Compiler uses Outer$1.class, Outer$2.class**
- **Inner class declared in method gets Outer1Inner.class**
- **Can access inner classes -- even anonymous ones with reflection and `Class.forName`**
- **However, hard to know how compiler has numbered classes**

Anonymous Class Initialization

- **QUESTION: If an anonymous class is a class, how do you write its constructor?**
  - **Constructor has same name as the class**
- **ANSWER: You can't. So the language designers invented the initializer block**
  - **modeled to look like a static initializer block**
  - **almost never worth using, but more language complications**
  - **implies that you can only call a constructor defined in class that anonymous class extends or zero-parameter constructor if implementing an interface**

Inner Class Patterns

- **Static inner class is private with public/package data**
- **Inner class is private (possibly static), with public/package data and implements a public interface. Outer class has factory method that returns inner class instances using the interface type**
- **Both patterns used in linked lists and other data structures**
- **Other pattern is to use anonymous inner classes for function objects**

Class Relationships

- **Nested (static inner) relates TYPES**
  - **inner class type related to outer class type**
- **(Instance) Inner relates OBJECTS**
  - **Objects of the inner class type always associated with exactly one object of the outer type**

Summary

- **The speculation was correct: adding inner classes complicates the language**
- **Static inner classes (nested classes) just like C++ nested classes**
- **Instance inner class objects always have reference to outer object**
- **Inner classes generally private, so syntax ok**
- **Classes inside of functions can access final local variables and can even have no name**
- **Use anonymous classes judiciously**

