

Nested Classes

reference: java.sun.com/docs/books/tutorial/java/javaOO/nested.html

Types of Classes

- Top Level
 - Static Nested
 - Inner
 - Member
 - Local
 - Anonymous
- ← Everything we have talked about so far...

Static Class

- Define one class inside the class definition of another class.
 - The nested class is available everywhere, although the name depends on the outer class

```
class MyOuterClass {  
    ...  
    static class Blah {  
        ...  
    }  
}
```

The name of this class (in other files) is `MyOuterClass.Blah`:

```
MyOuterClass.Blah b;
```

Static Nested Class Usage

- Really just a class naming issue...
 - The nested class can only make use of static methods/members of the outer class.
 - There doesn't need to be an object of the outer class in order to create an object of the nested class.

Inner Member Classes

- Inner Member Class:
 - Nested definition that is a *member* of the outer class.
 - The outer class can create instances of the inner class.
 - The inner class can access members and methods of the outer class.

Inner Member Class Example

```
class MyBackwardsList extends MyList {
```

```
...
```

MyIterator is a *member* of
MyBackwardsList



```
class MyIterator implements Iterator() {
```

```
    public MyIterator(...
```

```
        public boolean hasNext() {...
```

```
        public Object next() { ...
```

```
        ...
```

```
    }
```

```
public Iterator iterator() {
```

```
    return(new MyIterator(...));
```

```
}
```

Inner Local Classes

- Class definition happens inside a block of code (in a method).
 - The block of code that includes the inner class definition will instantiate (create) an object of the inner class.
 - Not a member of the outer class!
 - Inner class is not *visible* outside of the block it is defined in!
 - Inner class has access to all outer class data/methods.

Huh? Why? When?

- Lots of use in GUI programming!
- Typical use:
 - create an inner class that defines an object that will be used as a *callback*.
 - for example, the code that runs when a button is pressed.
 - This keeps the button handling code definition close to the button definition code.
 - Software Engineering Issue.

Inner Local Example

```
class MyBackwardsList extends MyList {  
  
    ...  
    public Iterator iterator() {  
        class MyIterator implements Iterator() {  
            public MyIterator(...  
            public boolean hasNext() {...  
            public Object next() { ...  
                ...  
            }  
            return (new MyIterator(...));  
        }  
    }  
}
```

Anonymous Classes

- You can create a class *on-the-go*, without even stopping to name the class.
 - This is a special kind of local class.
 - Often there is no advantage in actually coming up with a name!
 - Saves you those endless hours of pondering the best name for each of your classes.
 - I tried this with my 3rd child
 - His school hates me. I tell them "If it's cool in Java, it's cool at home". They take exception to this and say I have no class.

Creating an anonymous class

- You must create an object of the class.
 - The new class must extend or implement something...

```
blah = new Foo() {  
    FooMethod() {  
        System.out.println("I have no name");  
    }  
    AnotherMethod() { ... }  
}
```

This slide has no name

```
class MyBackwardsList extends MyList {  
  
    ...  
    public Iterator iterator() {  
        return new iterator() {  
            public MyIterator(...  
            public boolean hasNext() {...  
            public Object next() { ...  
                ...  
            }  
        }  
    }  
}
```

We will use inner classes!

- GUI code is full of this stuff!
 - it's hard to understand if you don't understand inner classes.
 - it's much harder to write if you don't understand *and love* inner classes.
- There are lots of other places where inner classes are useful.