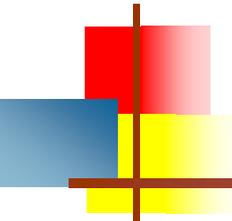


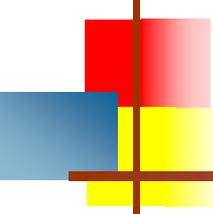
Inner Classes





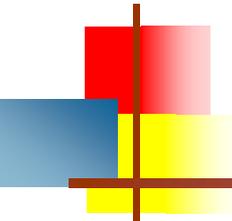
Inner classes

- All the classes so far have been “top level”
- It is possible (and useful) to define a class inside another class
 - The usual access modifiers (**public**, **protected**, **private**) can be used
- Inner classes were not in Java 1.0
 - They had to be added in later
 - As a result, inner classes are not as well done as some other aspects of the language



Four kinds of inner classes

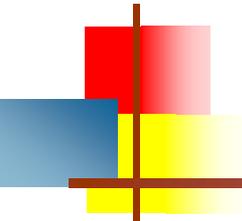
- **Member classes**
 - Simple and useful
- **Anonymous classes**
 - Useful, but syntax is ugly
- **Static member classes** (not too useful)
- **Local classes** (not too useful)
- *Every* class compiles to a separate **.class** file
- Inner classes compile to files with a **\$** in their names



Member classes

- A member class is an “ordinary” inner class

```
class Outer {  
    int n;  
  
    class Inner {  
        int ten = 10;  
        void setNToTen( ) { n = ten; }  
    }  
  
    void setN ( ) {  
        new Inner( ).setNToTen( );  
    }  
}
```

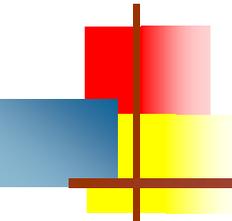


Member classes II

- Member classes are often used to handle events:

```
Button b = new Button ("Click Me");  
b.addActionListener (new Clicker( ));  
...  
class Clicker implements ActionListener { ... }
```

- A member class can access the variables of the enclosing class
 - This is what makes them so useful!
- Member classes are very easy
 - Declare them where you would declare a field or a method



Anonymous inner classes

- Anonymous inner classes are convenient for short code (typically a single method)

```
b.addActionListener(anonymous inner class);
```

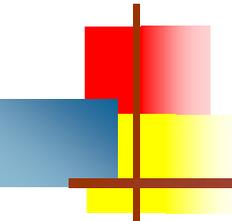
- The *anonymous inner class* can be either:

```
new Superclass(args) { body }
```

or

```
new Interface() { body }
```

- Notice that no class name is given--only the name of the superclass or interface
 - If it had a name, it wouldn't be anonymous, now would it?
- The *args* are arguments to the superclass's constructor (interfaces don't have constructors)



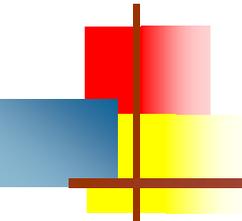
Example anonymous inner class

- An **ActionListener** is a Java-supplied interface for listening to Buttons and some other things
- The format (from the previous slide) is

```
new Interface () { body }
```

```
b.addActionListener (new ActionListener() {  
    public void actionPerformed (ActionEvent e) {  
        System.out.println ("Ouch!");  
    }  
});
```

- Like member classes, anonymous inner classes have full access to the fields and methods of the containing class



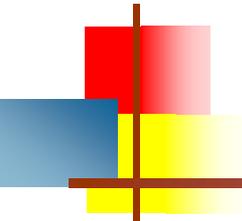
Static member classes

- `static class StaticMember { ... }`
- A static member class can access *only static variables* of the outer class
- A static member class isn't “really” an inner class, but a top-level class that happens to be written inside another class
- Static member classes are not too useful



Local classes

- A local class is a class defined inside a method
 - Like any other local declarations, the class *declaration* is available only within that method
 - However, objects created from that local class can “escape” the class by being assigned to nonlocal variables
- Because its instances may exist after the method exits, code in the local class cannot access variables declared in the method unless they are declared **final**
 - This makes them practically useless
- There are many other restrictions on local classes



Summary

- **Member classes**
 - An ordinary class, just defined within another
 - Has full access to the variables of the enclosing class
- **Anonymous classes**
 - Useful for short Listeners used in only one place
 - Has full access to the variables of the enclosing class
- **Static member classes**
 - Defined inside another class, but acts like an outer class
- **Local classes**
 - Defined within a method
 - Can access **final** variables in the enclosing class



The End

I have always wished that my computer would be as easy to use as my telephone. My wish has come true. I no longer know how to use my telephone.

— Bjarne Stroustrup