Inner Classes

Objectives

By the end of this module, you will have familiarity with the following concepts:

- Static member classes
- Member classes
- Local classes
- Anonymous classes

Definition of an Inner Class

- An inner class is simply a class within a class.

```
public class OuterClass {

        public class InnerClass {
        }
}
```

Inner Classes are Outer Class Members

- Inner classes are members of the outer class and therefore can have public, package, protected, or private visibility depending on the access modifier.
- Outer classes are only allowed to have access modifiers of public or package.
- Visibility for inner classes defines visibility within the package, or inside and outside the class.

Inner Class Visibility

```
public class OuterClass {
        public class PublicInnerClass {
        }
        private class PrivateInnerClass {
        }
         protected class ProtectedInnerClass{
        }
        class PackageInnerClass{
        }

 }
```

Resulting Classes from Compilation

- OuterClass.class
- OuterClass$PublicInnerClass.class
- OuterClass$PrivateInnerClass.class
- OuterClass$ProtectedInnerClass.class
- OuterClass$PackageInnerClass.class

Accessing Inner Classes

- The following syntax provides access to inner classes within proper access restrictions rules:

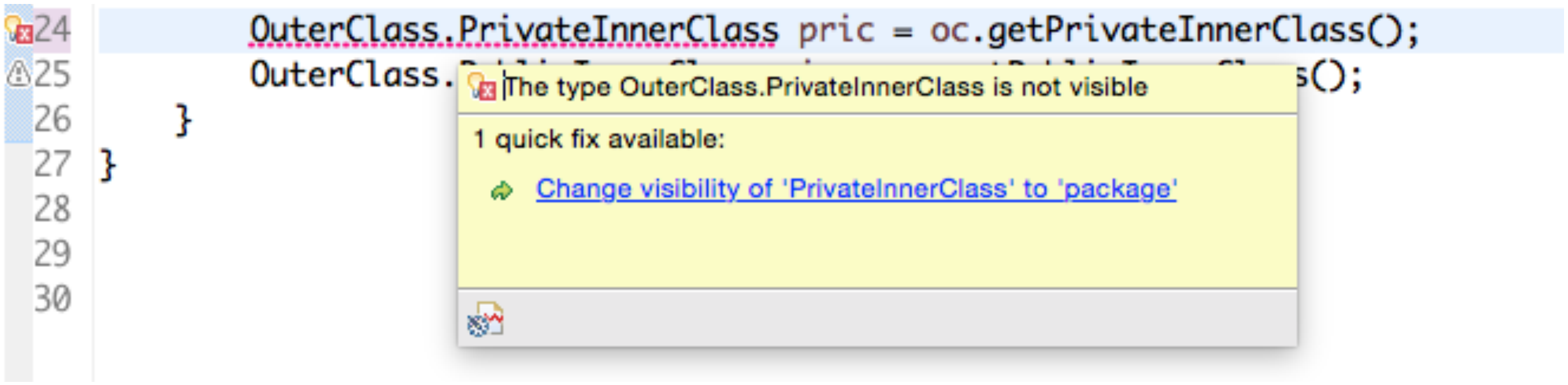
```
OuterClass oc = new OuterClass();

PublicInnerClass oPic1 = oc.new PublicInnerClass();

PublicInnerClass oPic2 = new OuterClass().new
                                        PublicInnerClass();
```

Private Inner Classes are Private

```
public PrivateInnerClass getPrivateInnerClass() {
                        return new PrivateInnerClass();
            }
 OuterClass.PrivateInnerClass pric = oc.getPrivateInnerClass();
```



```
        OuterClass.PrivateInnerClass pric = oc.getPrivateInnerClass();
        OuterClass.                                                s();
    }
}
```

The type OuterClass.PrivateInnerClass is not visible

1 quick fix available:

Change visibility of 'PrivateInnerClass' to 'package'

Fine-Grain Control

- Inner classes provide a program with finer-grained access and encapsulation.
- This enables better naming and access control.
- There is no limit to the number of inner classes a class may have.
- Inner classes may also have inner classes.
- Overuse can make code very hard to read and maintain.

Four Principles To Follow

- Cohesion – classes should contain related methods that cooperate to do one thing well.
- Understandability – being able to understand what a program does by reading its source code.
- Decoupling – if a change in one part of the program changes another part of the program, they are coupled.
- Testability – Write clean code.

Accessing Outer Class Variables

```
public class OuterClass {

      private List<Account> acctList = new
ArrayList<Account>();

      public class PublicInnerClass {
              List<Account> innerAccountList;

        private void
setOuterAccountList(ArrayList<Account> acctList){
                  OuterClass.this.acctList = acctList;
      }
              }
}
```

Circle Of Trust

- Each instance of a member inner class is bound to an instance of their outer class.
- Even though we access the inner class's *"this"* reference with the class name, it is still bound to a single instance.
- Inner classes are in the outer class's circle of trust

Private Stock Watcher Class

```
public class StockAccount extends Account  {
 private List<StockPosition> stocks = new
ArrayList<StockPosition>();
…
private class Watcher implements  StockWatcher{
       // This is the stock event listener change method
       @Override
       public void updateStockPrice(StockEvent se) {
              if (se.priceChanged()) {
              stocks.get(stocks.indexOf(se.getName())).
                            setLastPrice(se.getLastPrice());
              stocks.get(stocks.indexOf(se.getName())).
                            setNewPrice(se.getNewPrice());

                     }
              }
       }
}
```

Static Member Class Definition

- A static member class is a nested class that remains with the class definition.
- Unlike other static members, static nested classes can be instantiated.
- A non-static inner class can be referred to as a nested class, but a static nested class is not considered an inner class.
- Making a nested class static saves memory and improves performance.

Static Nested Classes

- Static nested classes cannot access members of their containing (outer) class.
- Since a static nested class does not have an implicit reference to the outer class, it would not know which instance to reference in order to access an instance member.
- Static nested classes are essentially and behaviorally a top-level class.

A Static “*this*”?

- Static classes have a reference to *this.*
- The reference refers to the instance of the static class.
- Static classes are allowed to have instance variables.
- Each instance of the LinkedList class contains an Entry class instance that is used to represent the head element in that specific list.

A Static “*this*”?

- the class OrderByHighestPosition is a static inner class with two static methods.
- The first method accepts a list of stocks as an argument.
- The method then uses a comparator to return the list sorted based on stock price.

```
public static final Comparator<StockPosition>
        PRICE_COMPARATOR = new Comparator<StockPosition>() {
        public int compare(StockPosition p1, StockPosition
p2) {
            return p1.getLastPrice().compareTo(p2.getLastPrice());
        }
};
```

OrderByHighestPosition

```
public class StockAccount extends Account {
              …
      public static final class OrderByHighestPosition {
         public static void
getHighestPostion(List<StockPosition> stockList) {
             Collections.sort(stockList, PRICE_COMPARATOR);
         }
      }
}
```

Top-Level Nested Classes

- Static inner classes are top-level nested classes.
- They are not tied to an instance of their outer class, and therefore do not have access to the outer class's members.
- We could have easily created a static inner class to represent our stock positions within our stock account and referenced the class from instances of the outer class, but not the other way around.
- We can use a static inner class directly without instantiation:

```
StockAccount.OrderByHighestPosition.gethighestPostion(
positions);
```

Local Inner Classes

- A local inner class is local to a block—usually a method.
- A local inner class can be defined inside any block of code, including a **for** loop, **if** statement, and **switch**.
- It has no reference outside the block and can only be instantiated within the block.
- Using a local inner class in a method allows that class to access the local final variables of that method.

Outer Class Variables Must Be Final

- Local inner classes can access any variables in the outer class regardless of access restriction, as long as the variables are final.
- Static variables can only be used in inner classes when they are declared final.
- Final or constant variables are limited to primitives and strings and must be initialized at compile time.
- Static methods are only allowed in top-level classes and static inner classes.

Local Inner Classes

- The only way to access our local inner class is when control is in the block containing the class.
- There is no way to access a local inner class without being in the method where the local inner class is declared.
- In the LocalInnerClass example, the classes that are generated are:
  - LocalInnerClass.class
  - LocalInnerClass$1UUIDUtils.class
  - BadUUIDException.class

Closure

- The advantage of local inner classes is the ability for the instance of the class to access final local variables in the containing method.
- When the local class instance uses a final variable, it retains the value of that variable, even if the variable goes out of scope.
- Some languages call this closure.

Anonymous Inner Class

- An anonymous inner class is a local inner class minus the name.
- Anonymous classes are created on the fly.
- They are a more convenient way to implement a quick interface or create a quick class to handle a specific task.
- The class is used only once and then discarded until a new one needs to be created.
- Once used there is no reference to the class and it will be garbage collected.

Quick and Easy

```
final StockPosition sp = new
StockPosition(tradedSymbol,
                        price, quantity);
sp.addStockWatchers(new StockWatcher() {
        @Override
        public void updateStockPrice(StockEvent
se) {
                sp.setNewPrice(se.getNewPrice());
        }
});

stocks.add(sp);
```

Closure

- In the example, we add a new StockEvent listener using the addStockWatcher() method.
- Instead of creating a concrete class that implements the StockWatcher interface, we create an anonymous class that overrides the single method in the interface.
- The anonymous class is essentially a local inner class and therefore can only use *final* variables.

Summary

We covered

- Static member classes
- Member classes
- Local classes
- Anonymous classes