Inner and Anonymous Classes

- Inner Classes
- Anonymous Classes

Introduction to Java

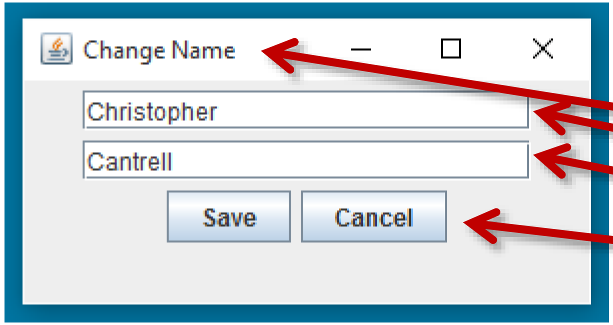
See Also

https://docs.oracle.com/javase/tutorial/java/javaOO/innerclasses.html

http://www.javaworld.com/article/2077411/core-java/inner-classes.html

http://stackoverflow.com/questions/355167/how-are-anonymous-inner-classes-used-in-java

Inner Classes



```
public class ClickHandler {

    private AppWindow app;

    public void buttonClick() {

        app.labelOne = "Input";
        app.clearInputs();
        app.setText("Your Name");;
        app.activateButton();

    }

}
```

- Objects can be tightly coupled
- For Example: Event Handlers
- Code gets tedious

Inner Classes

```
public class AppWindow {

    String labelOne;
    String labelTwo;

    public void setText(String text) {}

    public void activateButton() {}

    public void clearInputs() {}

    public void doAnimation() {}

}
```

```
        // Create the main window
        AppWindow w = new AppWindow();

        // Create a click listener for main
        ClickHandler h = new ClickHandler(w);

        h.buttonClick();
```

```
public class ClickHandler {

    private AppWindow app;

    public ClickHandler(AppWindow app) {
        this.app = app;
    }

    public void buttonClick() {
        app.labelOne = "Input";
        app.clearInputs();
        app.setText("Your Name");;
        app.activateButton();
    }

}
```

Inner Classes

```
public class AppWindow {
    String labelOne;
    String labelTwo;
    public void setText(String text) {}
    public void activateButton() {}
    public void clearInputs() {}
    public void doAnimation() {}

    class ClickHandler {
        //private AppWindow app;
        //public ClickHandler(AppWindow app) {
        //    this.app = app;
        //}

        public void buttonClick() {
            labelOne = "Input";
            clearInputs();
            setText("Your Name");
            activateButton();
        }

    }
}
```

```
// Create the main window
AppWindow w = new AppWindow();

// Create a click listener for main
ClickHandler h = new ClickHandler(w);

h.buttonClick();
```

- Put class inside class
- Compiler moves to separate class
- Adds constructor automatically
- Adds “app.” automatically

Inner Classes

```
public class AppWindow {
    String labelOne;
    String labelTwo;
    public void setText(String text) {}
    public void activateButton() {}
    public void clearInputs() {}
    public void doAnimation() {}

    class ClickHandler {

        public void buttonClick() {
            labelOne = "Input";
            clearInputs();
            setText("Your Name");
            activateButton();
        }

    }
}
```

```
AppWindow w = new AppWindow();

// ClickHandler h = new ClickHandler(w);

AppWindow.ClickHandler h =  w.new ClickHandler();
```

- Alternate syntax
- Remember: you must use a parent object instance in "new"

Inner Classes

```
public class AppWindow {

    public void doStuff( /*this*/ ) {
        ClickHandler h = new ClickHandler( /*this*/ );
    }

    class ClickHandler {

        public void buttonClick() {}

    }

}
```

- In a method the compiler will use "this" if you don't give an instance

Permissions

```
public class AppWindow {

    private void doAnimation() {}

    public class ClickHandler {

        public void buttonClick() {
            // I have permission
            doAnimation();
        }

    }

}
```

- Inner classes are “inside” the parent
- Access to “private” members
- Inner classes have permissions too

Static Inner Classes

```java
public class AppWindow {

    String labelOne;
    String labelTwo;

    public void setText(String text) {}
    public void activateButton() {}
    public void clearInputs() {}
    private void doAnimation() {}

    public static class ClickHandler {

        public void fun(AppWindow app) {
            app.doAnimation();
            System.out.println("I am here");

        }

    }

}
```

```java
AppWindow.ClickHandler c = new AppWindow.ClickHandler();

c.fun(w);
```

- Used for scoping (permissions)

Anonymous Classes

```
public class Point {

    int x;
    int y;

    @Override
    public String toString() {
        return "("+x+","+y+")";
    }

}
```

```
class SecretPoint extends Point {

    // Must copy/delegate constructors

    @Override
    public String toString() {
        return "*****";
    }

}
```

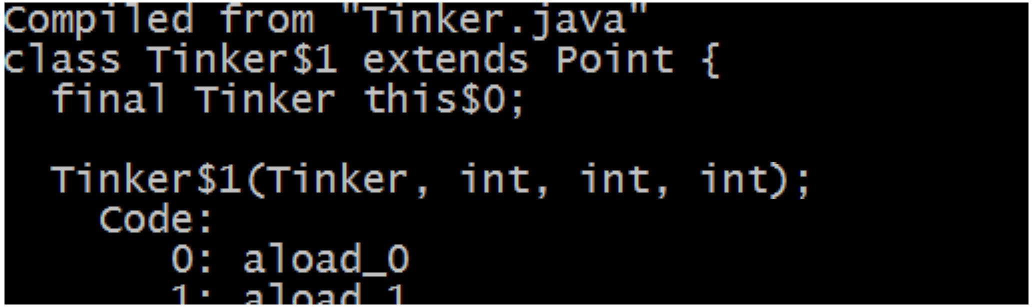
```
// Point s = new Point();
Point s = new SecretPoint();
System.out.println(s);
```

Anonymous Classes

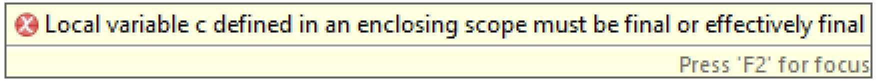
```
int a = 4;
int b = 5;

Point s = new Point(a,b) {
    @Override
    public String toString() {
        return "*****";
    }
};

int c = a + b;

System.out.println(s);
```

Anonymous Classes

```
 int myVal = 20;

public void fun() {

    int a = new Random().nextInt(5);
    int b = 5;
    int c = 10;

    Point s = new Point(a,b) {
        @Override
        public String toString() {
            System.out.println(myVal);
            System.out.println(a);
            System.out.println(b);
            System.out.println(c);
            return "*****";
        }
    };

    c = c + 1;

    System.out.println(s);
}
```



```
Compiled from "Tinker.java"
class Tinker$1 extends Point {
  final Tinker this$0;

  Tinker$1(Tinker, int, int, int);
    Code:
       0: aload_0
       1: aload_1
```



Local variable c defined in an enclosing scope must be final or effectively final

Press 'F2' for focus

Tinkering

- Code up a tinker with the “AppWindow” and “ClickHandler” inner class example.
- Try the Point anonymous class example. Why is it a good idea for “String” to reject this kind of override? How can you make Point reject it (like String does)?