CSCI 251: Android App Development

Prof. Levy

Anonymous Inner Classes in Java

Motivation

- The View & Controller parts of the MVC pattern are supported by listeners and event-handlers.
- Although these concepts may be familiar, the Android code looks strange:

```
TextView txtTitle;
...
txtTitle.setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View v) {

   }
});
```

http://stackoverflow.com/questions/19265626/why-should-i-use-anonymous-classes-in-android-instead-of-class-redefinition

- Let’s break this down into parts ….

This is why it's called *anonymous*; usually we would assign the result of `new` to a new variable:

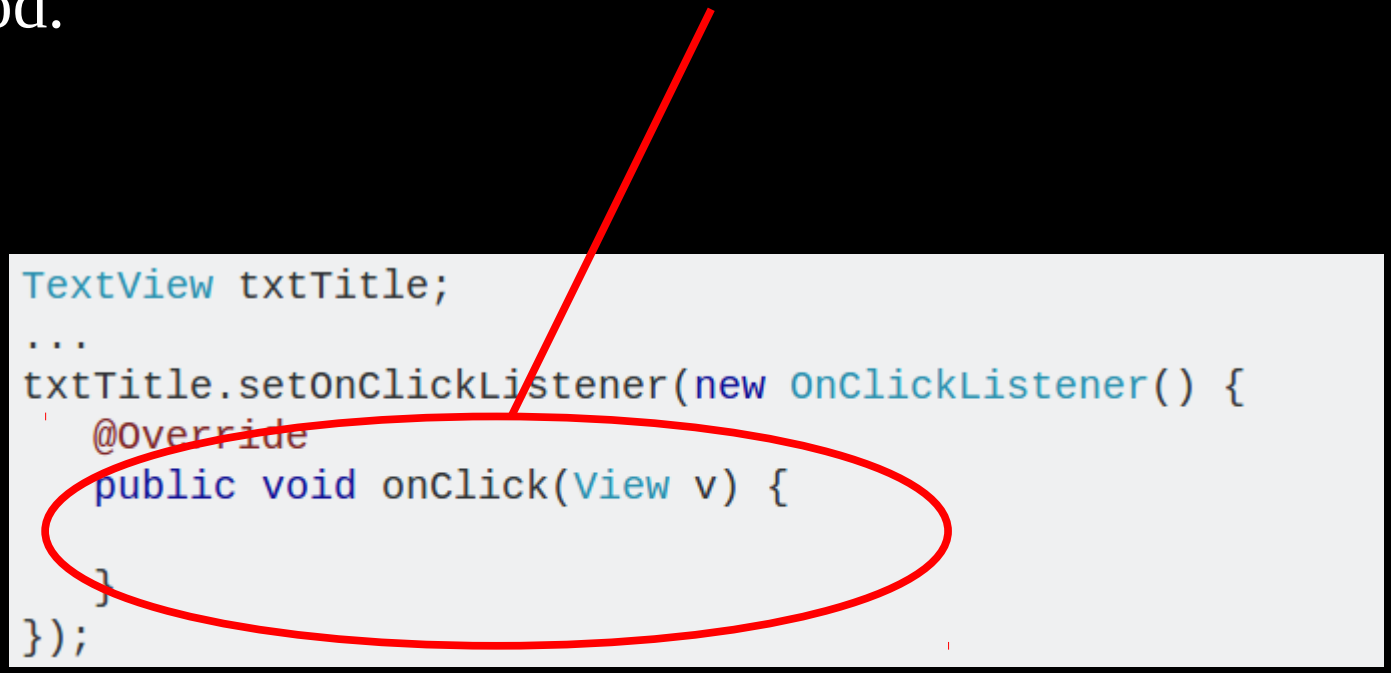
```
OnClickListener myListener = new OnClickListener();
txtTitle.setOnClickListener(myListener);
```

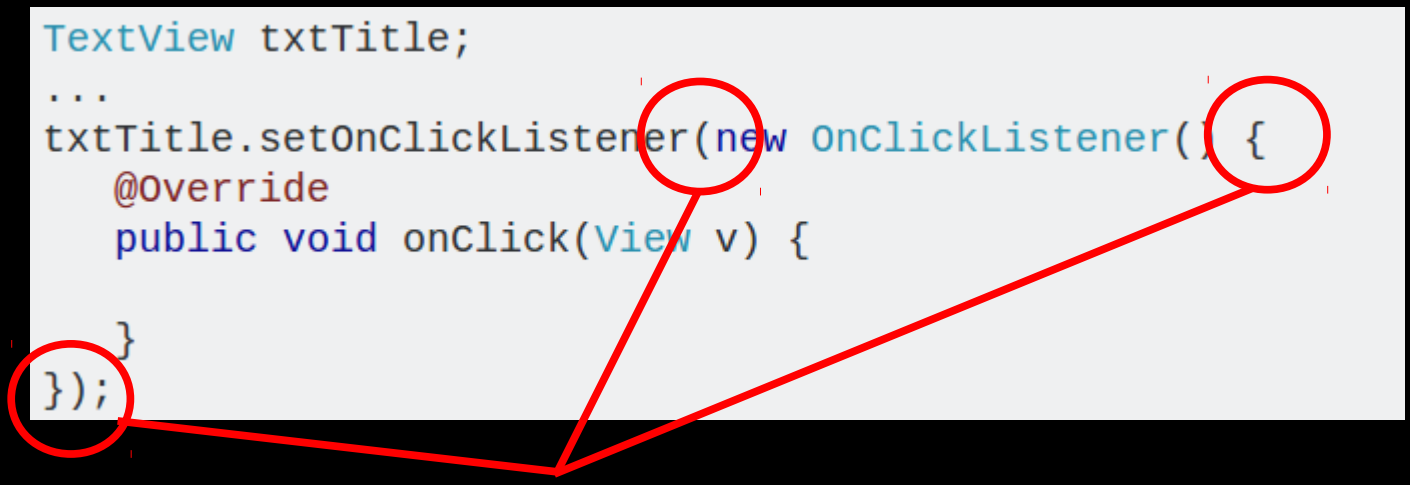
```
TextView txtTitle;
...
txtTitle.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {

    }
});
```

The `@override` annotation is a safety mechanism that tells the Java compiler to make sure that the class we are instantiating (`OnClickListener`) actually provides the method we want to override (`onClick`). If not, we'll get a compiler error. This trick helps us avoid failing to override the method by misspelling its name, then wondering why our app isn't responding!

```
TextView txtTitle;
...
txtTitle.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {

    }
});
```

This is why we call it an *inner* class: the method declaration (`onClick`) and body are inside a method invocation (`setOnClickListener`) of another class (`TextView`). In a real application, there would of course be code inside this `onClick` method.





Unusual syntax! The curly brace ends the `OnClickListener` class declaration. The parenthesis and semicolon end the `setOnClickListener` method invocation.