Anonymous Classes

A short-cut for defining classes when you want to create only one object of the class.

Why Anonymous Class?

- Sometimes we define a class just to create only one instance of the class.

Example: a Comparator to sort strings ignoring case

```
Comparator<String> compIgnoreCase
                        = new MyComparator();
Collections.sort( list, compIgnoreCase );

/** Compare strings ignoring case. */
class MyComparator implements Comparator<String> {
    public int compare(String a, String b) {
        return a.compareToIgnoreCase(b);
    }
}
```

Anonymous Classes (2)

- Java lets you define the class and create one object from that class **at the same time**.

The class is ***anonymous*** -- it doesn't have a name.

```
Comparator<String> compIgnoreCase =
    new Comparator<String>() {
             A class that implements Comparator
        public int compare(String a, String b) {
           return a.compareToIgnoreCase(b);
        }
    };

Collections.sort( list, compIgnoreCase );
```

How to Create Object using an Anonymous Class

- An anonymous class always extends another class or implements an interface.

Name of existing class *to extend* or existing interface *to implement*

Parenthesis after name, but **no semi-colon**

```
new Interface_Or_Class( )
{
    class definition
}
```

class definition inside of { }

Example: implement interface

Create an object that implements the Comparator interface to compare Strings *by length*.

```
/** Compare strings by length. */
Comparator<String> compByLength =
          new Comparator<String>()
 { /* definition of anonymous class */
   public int Compare(String a, String b) {
        return a.length() - b.length();
   }
 };
Arrays.sort( strings, compByLength );
```

Example: extend a class

Create an object that **extends** MouseAdapter (a class) to override one method for mouse-click events. The other methods are inherited from MouseAdapter.

```
MouseAdapter click = new MouseAdapter( )
 {
  public void mouseClicked( MouseEvent evt ) {
     int x = evt.getX();
     int y = evt.getY();
     System.out.printf("mouse at (%d,%d)", x, y);
  }
 };
```

Example: interface with type param.

You can use type parameters in anonymous classes.

Example: a *Comparator* for the Color class to compare colors by amount of red.

```
Comparator<Color> comp
            = new Comparator<Color>( )
 {
  public int compare(Color c1, Color c2){
    return c1.getRed() - c2.getRed();
  }
 };
```

Rules for Anonymous Classes

May have:

- instance attributes
- instance methods

May **not** have:

- constructor
- static attributes
- static methods

$\left.\begin{matrix} \\ \\ \\ \end{matrix}\right\}$ This makes sense!

... the class doesn't have a name.

Parameter for Superclass Constructor

You can supply a *parameter* to Anonymous Class.

- parameters are passed to the ***superclass constructor***.
- in this case, anonymous class must extend a class.

```
// Anonymous class extends AbstractAction.
// "ON" is passed to AbstractAction
// constructor, like using super("ON")
Action on = new AbstractAction("ON")
{
   public void actionPerformed(ActionEvent evt) {
      //TODO perform action
   }
};
```

Rule: accessing outer attributes

An *anonymous class* can access attributes from the surrounding object.

```
// message is an attribute
private String message = "Wake Up!";
void wakeUpCall(Long delay) {
  // create a TimerTask that prints a msg
  TimerTask task = new TimerTask()
  {
      public void run( ) {
          System.out.println( message );
      }
  };
  Timer timer = new Timer();
  timer.schedule( task, delay );
```

Rule: accessing local variables

An *anonymous class* can access local variables from the surrounding scope **only if they are final**.

```
void wakeUpCall(Long delay) {
  final String message = "Wake Up!";
  // create a TimerTask that prints a msg
  TimerTask task = new TimerTask()
  {
      public void run( ) {
          System.out.println( message );
      }
  };
  Timer timer = new Timer();
  timer.schedule( task, delay );
```

GUI Code Builders & anonymous class

GUI code builders create anonymous classes and use the object all in one statement (no assignment to a variable).
This is a common example:

```
JButton mybutton = new JButton("Click Here!");
// define action listener and add it to mybutton
mybutton.addActionListener(
   new ActionListener() {
      public void actionPerformed(ActionEvent e){
        textfield.setText(
             "Ouch! Don't press so hard.");
      }
    }
);
```

JavaFX Example

All UI operations must be done on the Application thread.

For example, if you create and show a stage on an ordinary thread:

```
Stage mystage = new CounterView(counter);
mystage.show();
```

JavaFX will throw IllegalStateException:

This operation is permitted on the event thread only

Start task on Application Thread

A common solution for this is to create a *Runnable* object for your JavaFX code and run it using:

Platform.runLater(*Runnable task*);

runLater runs the task on the JavaFX Application thread.

? How do we create a *Runnable* for our CounterView ?

Anonymous Class for Runnable

The code we want to run:

```
Counter counter = new Counter();
Stage mystage = new CounterView(counter);
mystage.show();
```

Anonymous class & Platform.runLater(*Runnable*):

```
Counter counter = new Counter();
Platform.runLater( new Runnable() {
   public void run() {
      Stage mystage = new CounterView(counter);
      mystage.show();
   }
} );
```

Lambda for Runnable

The code we want to run:

```
Counter counter = new Counter();
Stage mystage = new CounterView(counter);
mystage.show();
```

using a Lambda for Platform.runLater(*Runnable*):

```
Counter counter = new Counter();
Platform.runLater(
    () -> {
        Stage mystage = new CounterView(counter);
        mystage.show();
    }
);
```

Summary

Use anonymous classes to reduce code when...

1. need only one object of the class
2. the code is short
3. you don't need a constructor, no static fields

Guidance:

Assign anonymous object to a variable for readability.

Don't assign-and-use in one statement.

For class with a single method, a Java 8 lambda expression is usually shorter.