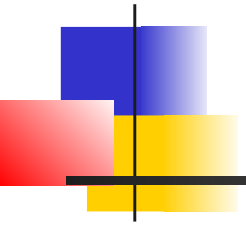

Anonymous Classes

A short-cut for 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> compareIgnoreCase
                   = new MyComparator();

/** 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> compareIgnoreCase =
    new Comparator<String>() {
          public int compare(String a, String b) {
             return a.compareToIgnoreCase(b);
          }
    };

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

A class that implements Comparator

How to Create Object using an Anonymous Class

- Write **`new Something() { /* class definition */ }`**
- Anonymous class always extends another class or implements an interface.

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

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

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

class definition inside of { }

Declaring a reference variable

- The declared type of the reference variable is the type of the superclass or interface.

task is declared to be type Runnable

```
Runnable task = new Runnable() {
    public void run() {
        // do something
    }
}
timeAndPrint( task );
```

Example: implement interface

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

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

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 event) {
      int x = event.getX();
      int y = event.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 objects.

```
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

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***.
- only applies for anonymous class that extends 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 "on" 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 );
```

Don't Write Code Like This

GUI code builders create an anonymous class and use it in one statement (no assignment to a variable).
This is an example:

```
Button mybutton = new Button("Click Here!");
// define action listener and add it to mybutton
mybutton.setOnAction(
    new Handler<ActionEvent>() {
        public void handle(ActionEvent e){
          textfield.setText(
                "Ouch! Don't press so hard.");
        }
    }
);
```

Please don't write code like this -- its harder to read.

Assign anonymous object to a variable with a descriptive name (as in examples on previous slides)

Summary

Use anonymous class when...

1. only need to create one object of the class
2. class is short
3. don't need a constructor or 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 Lambda expression is usually shorter.