Anonymous Inner Class

Thanks to Nick Parlante for much of this handout

Anonymous Inner Class (function pointer)

- An "anonymous" inner class is a type of inner class created on the fly with a quick-and-dirty syntax. Anonymous inner classes are convenient for creating small inner classes. An anonymous inner class can implement a superclass or interface, just like a regular inner class.
- Anonymous Inner Classes as shown below are essentially a way of making a cheap little function pointer objects to pass back to the client so they can invoke the code later on.
- As a matter of style, an anonymous inner class is appropriate for small inner classes. If the inner class involves longer code, then a regular named inner class is a better choice.
- When compiled, anonymous inner classes are given names like Outer$1, Outer$2 by the compiler.
- An anonymous inner class cannot have a constructor. It must rely on the default object initialization behavior.
- An anonymous inner class does not have a name, but it may be stored in a Superclass type pointer. The anonymous inner class has access to the outer class ivars, as usual for an inner class.
- The anonymous inner class does not have access to local stack vars from where it is declared, unless they are declared final. (details below)

Modularity With Interfaces

- Note that we have a known public interface, e.g. `Clicker` or `Collection`, that the client knows about
- Our inner class implements the interface, so we can pass it back to the client and they can store it in the known public interface, but the client never knows about the specific inner class we use as implementation, and in fact we can change it later, and the client will never know.
- This is a classic modularity strategy -- decoupling the client so it just depends on the minimum necessary.

VoteMachine Example

- Suppose we are making a voting machine that encapsulates a count of votes. The `Clicker` interface defines a `click()` method that votes once.
- The `VoteMachine` method `createClicker()` returns a `Clicker` object that can be used to make votes on that machine. This fits the "function pointer" interpretation of inner classes -- we pass back a thing to the client, and later, the client can use it to call our code.
- Inside `getClicker()`, the `Clicker` object is built as an anonymous inner class that is returned to the caller.

```
// Demonstrates anonymous inner classes.
// VoteMachine encapsulates a number of votes -- getClicker() yields
// a  Clicker object that you can call click() on to vote.

public interface Clicker {
    public void click();
}

public class VoteMachine {
    private int votes;

    public VoteMachine() {
        votes = 0;
    }

    public int getVotes() {
```

```
        return votes;
    }

    // Creates a regular Clicker object -- one click, one vote.
    Clicker createClicker() {
        // Make a little anonymous inner class object implementing Clicker.
        Clicker anon = new Clicker() {
            public void click() {
                votes++;
            }
        };
        return anon;
    }

    // Creates a special clicker object -- one click, N votes.
    Clicker createClicker(int change) {
        // Make Clicker anon inner class.
        // TRICK: Anon inner code cannot refer to local variables
        // from where it is defined.
        // However, anon inner can refer to FINAL local variables.
        final int finalChange = change;
        Clicker clicker = new Clicker() {
            public void click() {
                //votes += change; // NO does not compile
                votes += finalChange;  // ok, this works
            }
        };
        return clicker;
    }

    public static void main(String[] args) {
        VoteMachine voteMachine = new VoteMachine();

        // Get a couple clickers
        Clicker a = voteMachine.createClicker();
        Clicker b = voteMachine.createClicker(10);

        // a and b are objects -- can store them, pass them around, etc.

        // Call them
        a.click();
        a.click();
        b.click();

        System.out.println(voteMachine.getVotes());  // 12
    }
}
```

final var trick

- Anonymous inner classes can see ivars of their outer object, as usual for an inner class.
- Anonymous inner classes **cannot** see local variables from where they are created (i.e. stack allocated variables).
- This is a matter of timing: the local stack variables exists at the time of the call to new to create the inner class -- e.g. the local variable `change` in the `createClicker(int)` method. The client keeps a pointer to the clicker inner class, and sends it the `click()` message some time in the future. At the time `click()` runs, notice that `createClicker(int)` is not running. It exited earlier, and its local variables/stack no longer exist. That's why you can't refer to them from inside the anonymous inner code. When the anonymous inner code runs, that context no longer exists.
- However, inner classes can see `final` stack vars from where they are created -- so create `final` stack vars to communicate values to an anonymous inner class. The compiler makes the `final` variant work by copying the value at the time the anon inner is created and giving the inner class access to the copy. This can work, since the `final` means that the value will not change.
- Use `VoteMachine.this` to refer to the `this` pointer of the outer object. Necessary in some cases if the compiler cannot distinguish that an ivar should be available from the outer object.