Anonymous Classes for Event Handlers

Week 5 Presentation 2

Nested Classes

```
class OuterClass {
        …
        class NestedClass {
           …
        }
}
```

```
class OuterClass {
        …
        public void doSomething() {
                class LocalClass {
                   …
                }
        }
}
```

Nested classes are classes declared inside other classes….

In Java you can also have local classes which are classes declared inside a method.

Why?

- Encapsulation
- Grouping related functionality together

- These nested or local objects are not needed outside the context in which they are declared (within another class or in a method)

It is useful for code clarity, but it also means that the objects are kind of "technical objects" that aren't expected to be used anywhere else and you won't have a reference to them generally available elsewhere.

Anonymous Classes

```
Class NamedClass implements Interface{
    …
}

Interface anObject = new NamedClass(…)
```

We saw a similar case with anonymous classes where some functionality requires an interface. For example an object that implements the “comparable” interface is often not really needed by a caller or other program parts generally – all that is needed here often is to provide a sorting utility method with a comparator. Other interfaces may have additional attributes and methods that are also not needed by the caller so would be unnecessary work to implement the object completely.

Of course one can however define everything and create an object however…

Anonymous Classes

```
Interface anObject = new Interface() {
                    // attribute and method implementation
                    };
```

And the Java syntax also allows creating an interface on the fly, and you can also directly instantiate an object by implementing the interface methods here...

This is very convenient for passing as a method parameter

Anonymous Classes

Also works for inheritance

```
class NamedClass extends ParentClass{
   …
}
NamedClass anObject = new NamedClass(…);

or
ParentClass anObject = new ParentClass(){
                                   // attribute and method defitions
                                          };
```

GUI: Event Handlers

This is common for graphical user interface programming for instance event listers. Here is an example of listening for a button press…

```
…
Button but = new Button();
but.setText(”Say ‘Hello’”);
But.setOnAction(new EventHandler<ActionEvent>(){
          @Override
          public void handle(ActionEvent e){
                    System.out.println(“Hi”);
          }
});
…
```

Only one method!

Only one method!

- If there is only one method then we can use Lambda expressions instead

In the very common case where your interface requires a single method, you can use lambda expressions. Lambda expressions come from functional programming, where you try not to store any state (which is completely opposed to attributes that store the state of an object ...)

- Cool!

Lambda Expressions

- Lambda expression

(parameter list) -> {statements}

- Recall that Lambda expressions are assigned to functional interfaces

@FunctionInterface

Functional interfaces have only one abstract method

- If there is only one method to define for an interface you don’t have to give it a name

Using Lambda Expression in Event Listener

```
…
Button but = new Button();
but.setText(”Say ‘Hello’”);
But.setOnAction(new EventHandler<ActionEvent>(){
          @Override
          public void handle(ActionEvent e){
                    System.out.println(“Hi”);
          }
});
…
```

Anonymous Class

Using Lambda Expression in Event Listener

Using Lambda Expression in Event Listener

```
…
Button but = new Button();
but.setText("Say 'Hello'");
But.setOnAction(new EventHandler<ActionEvent>(){
        @Override
        public void handle(ActionEvent e){
                System.out.println("Hi");
        }
});
…
```

Anonymous Class

As "handle()" is the only method of an event handler, it can also be written like this:

```
Button btn = new Button(); btn.setText("Say 'Hi'");
btn.setOnAction((e)->{
        System.out.println("Hi!"); });
```

Much shorter

Other common uses are in searching and sorting collections where if you recall the comparable interface has only o
method.