4. Java OOP

6. Inner Classes

Nested Classes (1 of 2)

- A *nested class* is a class defined within another class:

```
class OuterClass {
    ...
  class NestedClass {
      ...
  }
}
```

Nested Classes (2 of 2)

- A nested class is a member of its enclosing class
- Non-static nested classes (**inner classes**) have access to other members of the enclosing class, even if they are declared private
- Static nested classes do not have access to other instance members of the enclosing class

Why Use Nested Classes?

- It is a way of logically grouping classes that are only used in one place.
- It increases encapsulation.
- Nested classes can lead to more readable and maintainable code (places the code closer to where it is used)

Static Nested Classes (1 of 2)

- A static nested class is associated with its outer class
- Like static class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class - it can use them only through an object reference

Static Nested Classes (2 of 2)

- Static nested classes are accessed using the enclosing class name:

  OuterClass.StaticNestedClass

- To create an object for the static nested class, use this syntax:

```
OuterClass.StaticNestedClass nestedObject =
     new OuterClass.StaticNestedClass();
```

Inner Classes (1 of 2)

- An inner class has direct access to that object's methods and fields
- It cannot define any static members itself
- Objects that are instances of an inner class exist *within* an instance of the outer class

Inner Classes (2 of 2)

- To instantiate an inner class, you must first instantiate the outer class. Then, create the inner object within the outer object with this syntax:

```
outerClass.InnerClass innerObject = outerObject.new
InnerClass();
```

Local Inner Classes

- Inner classes can be created inside code blocks, typically inside the body of a method
- A local inner class cannot have an access specifier
- It does have access to the final variables in the current code block and all the members of the enclosing class

Anonymous Classes

- Anonymous classes combine the process of definition and instantiation into a single step
- As these classes do not have a name, an instance of the class can only be created together with the definition

Anonymous Class Example I

```
new Thread(new Runnable() {
   public void run() {
      ...
   }
}).start();
```

Anonymous Class Example II

```
JFrame frame = new JFrame("AnonimDemo2");
frame.addWindowListener(new WindowAdapter() {
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
});
```

Anonymous Classes Use

- For creating objects on the fly in contexts such as:
  - the value in a return statement
  - an argument in a method call
  - in initialization of variables
  - to implement event listeners in GUI-based applications

Manuals

- http://docs.oracle.com/javase/tutorial/java/javaOO/nested.html