Java Object-Oriented Concepts Unit

Lesson 1.4: Nested Classes

The Learning House

427 S 4th Street #300

Louisville KY 40202

Lesson 1.4: Nested Classes

Overview

In this portion of Lesson 01, we'll take a look at Nested Classes. A nested class is a class that is defined within another class. Nested classes can help with code organization and grouping, encapsulation, and the readability/maintainability of your code. We will look at the following:

1. Static nested classes
2. Inner classes
3. Local classes
4. Anonymous classes
5. Lambda expressions
6. Guidelines for using nested classes

Important Definition

Nested classes come in two categories: static and non-static. Nested classes that are declared static are called **static nested classes**. Non-static nested classes are called **inner classes**. By definition, *there is no such thing as a static inner class in Java*.

Static Nested Classes

A static nested class is always associated with its enclosing class. A static nested class cannot directly refer to any instance variables or methods defined in the enclosing class — it must use them through an object reference. Effectively, a static nested class behaves like a top-level class that has just been nested in another class for convenience.

Inner Classes

Inner classes are associated with an instance of the enclosing class. The inner class has direct access to the enclosing object's methods and properties. Because it is associated with a particular instance of the enclosing object, an inner class cannot have any static members.

Shadowing

If you declare a variable that has the same name as variable declared in the enclosing scope, your variable will **shadow** the variable in the enclosing scope. In other words, when you refer to the variable simply by its name, you will access/manipulate the local version of that variable. In order to access the variable in the enclosing scope, you cannot use the variable name alone. A common example is the use of the **this** keyword when referring to class-level variables to distinguish the class-level variable from the local version.

Local Classes

Java allows developers to define classes inside any block of code. Classes defined in such a way are called **local classes**. Typically, local classes are defined within a method, but they can be defined in any block of code (a for loop, for example). Local classes (like inner classes) have access to the members of the enclosing class and cannot have any static members. Additionally, they have access to the local variables of the surrounding block as long as the local variables have been declared final (Java 7 and earlier) or are effectively final (Java 8 only).

Anonymous Classes

Anonymous classes are similar to local classes. The big difference is that anonymous classes are declared and instantiated at the same time and they do not have a name. You would use an anonymous class only in cases where you need to use the class exactly once. Anonymous classes have the same access to local variables of the enclosing scope as local classes do.

Lambda Expressions

Many times, anonymous classes are used in Java as a way to pass a method or behavior as a parameter into another method. Before Java 8, this was the only way for Java to approximate functional programming (functional programming languages treat methods/behaviors as first-class citizens). Although this works, the syntax is quite verbose and clunky. Lambda expressions were introduced into Java 8 to add some functional programming capabilities to the Java language. This allows Java developers to treat code as data.

The syntax of a lambda expression is as follows:

1. Comma-separated list of formal parameters inside parentheses
2. The arrow token: **->**
3. A body — a single expression or a block of code that may or may not return a value

Lambda expressions have the same access to local variables of the enclosing scope as do local and anonymous classes.

Guidelines

Here are some guidelines to help you determine when and what type of nested class you should use:

1. **Local Class** — Use a local class if you require more than one instance of the class, if you need access to the class's constructor, or if you wish to introduce a new named type so you can invoke the object's methods in later code.
2. **Anonymous Class** — Use an anonymous class if you only need one instance.
3. **Lambda Expression** — Use a lambda expression if you need to treat code as data as in a functional programming language.
4. **Nested Class** — Use a non-static nested class (aka inner class) if your class requires access to the enclosing instance's non-public fields and methods. If you don't require that level of access, use a static nested class instead.