**Nested Classes and Types**

A class can contain other types within the class body, such as other classes, interfaces, enums, and records.

These are called nested types, or nested classes.

4 types of nested classes in Java are

**The static nested class:**

It is declared in class body. Much like a static field, access to this class is through the Class name identifier.

The static nested class is a class enclosed in the structure of another class, declared as static.

This means the class, if accessed externally, requires the outer class name as part of the qualifying game.

This class has the advantage of being able to access private attributes on the outer class.

The enclosing class can access any attributes on the static nested class, including private attributes.

**Instance or inner class:**

Declared in class body. This type of class can only be accessed through an instance of the outerclass.

Inner classes are non-static classes, declared on an enclosing class at the member level.

Inner classes can have any of the four valid access modifiers.

An inner class has access to instance members, including private members of the enclosing class.

Instantiating an inner class from external code requires an instance of the Enclosing class. From that instance you call, new , followed by the inner class name and the parentheses, taking any constructor argument

EnclosingClass outerClass = new EnclosingClass();

EnclosingClass.InnerClass innerclass = outerClass.new Innerclass();

**Local class:**

Declared within a method body. They don’t access modifiers and are only accessible in that method body while executing. Like an inner class, they have access to all fields and methods on the enclosing class. They can also access local variables and method arguments, that are final or effectively final

**Anonymous class:**

Unnamed class, declared an instantiated in same statement. A local class that doesn’t have a name. All nested classes have been declared as with a class declaration. The anonymous class is never declared with a class declaration but its always instantiated as a part of expression. Anonymous classes are used less after the introduction of Lambda Expression in JDK 8. An anonymous class in instantiated and assigned in a single statement. {} will be the sign that it’s a class.

**Var c4 = new Comparator<StoreEmployee>() {};**

This is not the type of class being instantiated. It’s the super class of the anonymous class, or it’s the interface this anonymous class will implement as shown above.

**Var e1 = new Employee {};**

In the Employee example. The anonymous class is extending the Employee class. Meaning it’s a subclass of Employee.

In both of the examples above, there should be a semicolon at the end of the expression because it’s the expression not the declaration.