Theory: Nested classes

As you know, there are inner classes that you create inside the outer classes. But have you wondered if there are any other ways to create a new class inside the other? Today you will meet the different two types of nested classes. Our lovely heroes are called **static nested class** and **local inner class.** What's more, after completing this topic, you'll have the whole picture of the hierarchy of nested classes.

Current topic:

Nested classes

Topic depends on:

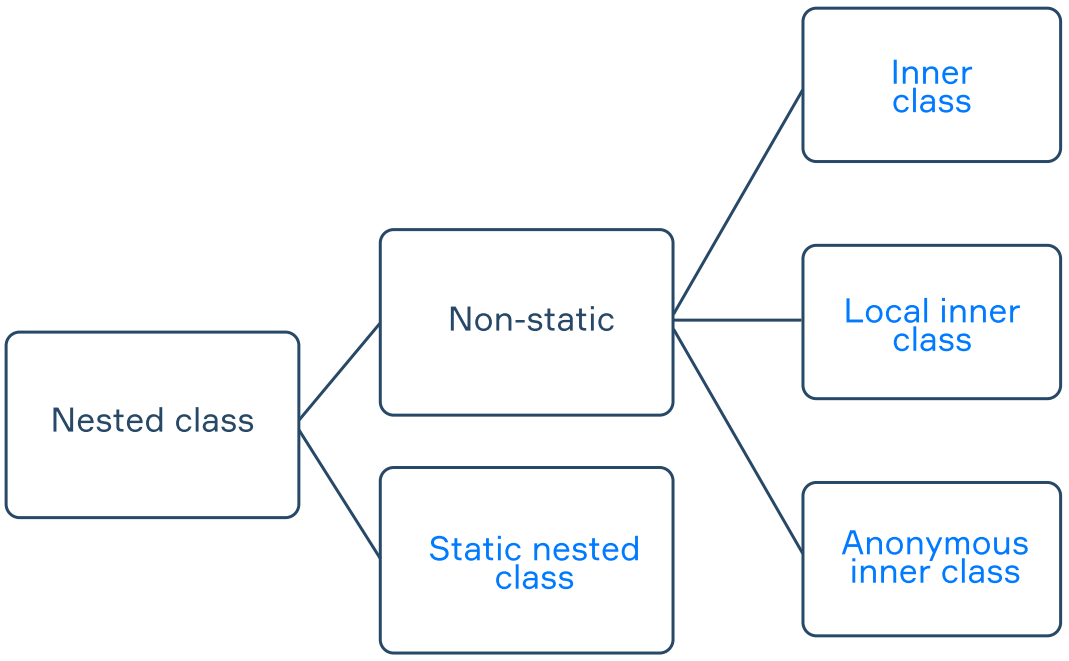
✓ Final variables

✓ Inner classes

§1. Types of nested classes

As you remember, a class is called **nested** when it's declared within another class. Let's look into the types of nested classes. There are four types of nested classes.

You can find the hierarchy in the picture, where the blue ones are these types.



First, all nested classes are divided into static and non-static ones. As you can see, only one type has the `static` keyword.

In Java documentation, you also may see that the non-static group includes local inner classes and anonymous inner classes along with the inner classes you know about.

Let's now take a closer look at **static nested classes** and **local inner classes**.

§2. Static nested class

Imagine, that one day you woke up and decided to draw. Obviously, with the help of some Java code. Your ultimate goal is to draw a painting. But how to create a masterpiece? That'll require doing some sketches first.

How would you organize your code? It's a good idea to use a nested class here. But if you choose to employ an inner class then a `Sketch` would only exist if the `Painting` was instantiated previously. So, we'd prefer something different. And the **static nested class** is going to help us.

It allows us to create a `Sketch` first, and then, only if you ready to become a masterpiece, you can create a `Painting`.

```
public class Painting {

    private String name;

    public static class Sketch {

        private int id;

        public Sketch(int id) {

            this.id = id;

        }



        public void drawSketch() {

            drawForest();

            drawBear();

        }



        private void drawForest() {

             System.out.println("Forest was drawn in sketch!");

            }

        }



        private void drawBear() {

            System.out.println("Bear was drawn in a sketch!");

        }

    }

}
```

Let's try it:

```
public class Main {
    public static void main(String[] args) {

        Painting.Sketch sketch = new Painting.Sketch(0);

        sketch.drawSketch();
    }
}
```

Greetings to our bear in a forest!

```
Forest was drawn in a sketch!
Bear was drawn in a sketch!
```

§3. Scope of static nested class

Let's modify our example a little bit with `Sketch` and `Painting` and talk about their scope.

```
public class Painting {

    private String name;

    private static double length;
    private static double width;

    public static void setLength(double length) {
        Painting.length = length;

    }



    public static void setWidth(double width) {

        Painting.width = width;

    }



    public static class Sketch {



        private int id;



        public Sketch(int id) {

            this.id = id;

        }



        public void drawSketch() {

            drawForest();

            drawBear();

        }



        private void drawForest() {

            if (Painting.length > 5 && Painting.width > 3) {

                System.out.println("Big forest was drawn in sketch!");

            } else {

                System.out.println("Small forest was drawn in a sketch!");

            }

        }



        private void drawBear() {

            System.out.println("Bear was drawn in a sketch!");

        }

    }

}
```

We've added two `static` fields into `Painting` class, namely `length` and `width`. And we've also added condition into the method `drawForest` of class `Sketch`.

With setters, we decide what sizes our `Painting` will be and then use that information inside the method `drawForest`.

```
public class Main {

    public static void main(String[] args) {

        Painting.setLength(10);
        Painting.setWidth(7);

        Painting.Sketch sketch = new Painting.Sketch();
        sketch.drawSketch();

    }

}
```

And here is a big forest with a bear:

```
Big forest was drawn in sketch!
Bear was drawn in a sketch!
```

So, we've got access to `private static` fields from static nested class!

And if there is something that we ***can't*** see? Yes, instance variables and methods of an outer class, including field `name` in our example.

From outside everything works as usual: we create an instance of a static nested class and good luck! Just mind the syntax:

```
OuterClass.NestedClass nested = new OuterClass.NestedClass();
```

Remember about *access modifiers*: if you make a static nested class `private`, then it only can be accessed inside the outer class. The same works with fields and methods.

§4. Local inner class

In real life, you won't face local inner classes often, but it is worth knowing how to use them if you want to be a proper programmer.

You can define a local inner class *inside any block*. But, usually, local inner classes are defined inside a method body.
Let's move to an example now:

```
public class Outer {

    private int number = 10;

    void someMethod() {

        class LocalInner {

            private void print() {

                System.out.println("number = " + Outer.this.number);

            }

        }


        LocalInner inner = new LocalInner();

        inner.print();

    }


    public static void main(String[] args) {

        Outer outer = new Outer();

        outer.someMethod();

    }

}
```

Here we have an outer class `Outer` and a method `someMethod` in it. We define our local inner class inside `someMethod` and also we create an instance of `LocalInner` there.

Have you noticed that our class `LocalInner` *doesn't have* an access modifier? And it can't!

There are other restrictions. You *cannot* define inside a local inner class: any static members, enums, interfaces.

§5. Scope of Local Inner class

The scope of the local inner class is restricted within the block, which is `someMethod` in our example.

```
public class Outer {

    private int number = 10;

    void someMethod() {
        final int x = 5;

        class Inner {
            private void print() {

                System.out.println("x = " + x);

                System.out.println("number = " + Outer.this.number);

            }

        }



        Inner inner = new Inner();

        inner.print();

    }



    public static void main(String[] args) {

        Outer outer = new Outer();

        outer.someMethod();

    }

}
```

What we *can* see inside the local inner class? Members of an outer class, including field `number`. And local variables of the enclosing block, such as void `someMethod`. Local variables must be declared as `final` or be effectively final, the latter means their value is never changed after initialization and there's no need for the keyword `final`.

Remember, that local inner class can be instantiated only *within the block* where the inner class is defined.
So other parts of the code *don't know* that it exists.

§6. Summary

If you want something close to a nested class, but don't want to instantiate an outer class — a static nested class is always there for you!
Static nested classes add functionality to an outer class and may be used for different purposes: usually, as a special structure that is connected with the outer class.

And have you heard about a class in a block? Yeah, you are a good student and you know, that it is called local inner class and its scope is restricted within a block.

Finally, nested classes **organize** code and help your package be more reasonable, increase **encapsulation**, since you can hide some code in a nested class, and the last point is that small classes may provide more **readable** code.

Report a typo

6 users liked this theory. **0** didn't like it. **What about you?**

Start practicing

This content was created about 2 months ago and updated 5 days ago. Share your feedback below in comments to help us improve it!

Comments (1) Hints (0) Useful links (0) Show discussion