Theory: Inner classes

28 minutes 0 / 5 problems solved

Skip this topic Start practicing

If you would like to create your own superhero, what would you do? Of course, you would open a favorite development environment and create a class `Superhero`!

To be a proper one, our superhero will need a set of special items, including, for example, a magic cloak or a hammer. Here may come a problem. What is the best way to organize the classes describing the equipment? And how to mark that only one class, that is `Superhero`, can use them?

Here appears our savior — an instrument called **nested classes**. They help us to group classes logically and increase the encapsulation of our code.

§1. What is a nested class?

Basically, you can call a class **nested** when it is declared inside another class.

This is how our superhero would look like:

```
class Superhero {

    class MagicCloak {

    }

    class Hammer {

    }

}
```

Both classes `MagicCloak` and `Hammer` are nested classes. The `Superhero` class is often called an **outer class**, and a nested class is called a **member** of an outer class.

In this topic, we are going to talk about **non-static nested classes**, that are commonly known as **inner classes**.

§2. Inner class

Let's move to another example. Imagine that you are writing a class `Cat` representing cats. The cat may have a lot of fields and methods, but we may also use inner class structures. For example, let's say you want a cat to have a bow. Then you need to create a new class `Bow`. This class `Bow` would be quite small and specific, and you know you won't need a bow without a cat. The solution is to create a class `Bow` inside the class `Cat`:

Current topic:

Inner classes

Topic depends on:

- ✓ Instance methods (Stage 6)
- ✓ Access modifiers (Stage 6)

Topic is required for:

Nested classes

Table of contents:

```
public class Cat {

    private String name;

    public Cat(String name) {
        this.name = name;
    }

    public class Bow {

        String colour;



        public Bow(String colour) {

            this.colour = colour;

        }



        public void printColour() {

        System.out.println("Cat " + Cat.this.name + " has " + this.colour + " bow.");

        }

    }

}
```

Let's create a cat `Bob` with a red bow:

```
public class Main {

    public static void main(String[] args) {

        Cat cat = new Cat("Bob");
        Cat.Bow bow = cat.new Bow("red");

        bow.printColour();
    }

}
```

Look, we have created an instance of `Cat` and then created an instance of `Bow` using quite interesting syntax.

Here, the output will be:

```
Cat Bob has red bow.
```

Remember that to use inner classes we must *create an instance of the outer class.* In our example, we created a `Cat`.

§3. Scope of the inner class

Now let's discuss what we can see from the inner class and who can access the inner class from outside.

There is our class `Cat` with new method `sayMeow` and an inner class `Bow` with new method `putOnABow` .

```
public class Cat {

    private String name;

    public Cat(String name) {
        this.name = name;
    }

    private void sayMeow() {

        System.out.println(this.name + " says: \"Meow\".");

    }



    public class Bow {

        String colour;



        public Bow(String colour) {

            this.colour = colour;

        }



        public void putOnABow() {

            Cat.this.sayMeow();

            System.out.println("Bow is on!");

        }



        public void printColour() {

            System.out.println("Cat " + Cat.this.name + " has " + this.colour + " bow.\n");

        }

    }

}
```

Note, that inside the method `putOnABow` of class `Bow` we have access to private method `sayMeow()` of class `Cat` . We also have access to a private field `name` of class `Cat` — we get it in method `sayMeow`.

How about cat `Princess` with a golden bow to prove that our code works?

```
Cat cat = new Cat("Princess");
Cat.Bow bow = cat.new Bow("golden");

bow.printColour();
bow.putOnABow();
```

And, yes, the bow is on!

```
Cat Princess has golden bow.

Princess says: "Meow".
Bow is on!
```

As for the scope from the outside world: when you've instantiated an inner class, you can do whatever you want according to access modifiers.
Now let's collect all rules together and put them into a human "hard disk"!

§4. Rules for Inner classes

Inside the inner class, we can see all methods and fields of an outer class even if they are `private`. And don't forget that we can use everything else according to access modifiers as well.

An inner class is associated with an instance of its enclosing class. So to instantiate an inner class and get access to it you need to instantiate an outer class first:

```
Outer outer = new Outer();
Outer.InnerClass inner = outer.new InnerClass();
```

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

And be careful — there are always some restrictions!

Inside an inner class, you **cannot** define:

- Any static members;
- Enum;
- Interface.

§5. Reasons to use Inner Classes

Have you noticed what our examples about `Superhero` with magic items and `Cat` with a bow have in common? Sure you have — we hid our inner classes from the outside world. So that only `Superhero` may use a magic cloak and only a `Cat` may put on a bow. Also now it will be easier to navigate between classes and to understand the structure of your code.

And, at last, the formal (just a little bit) list of reasons:

1. they increase **encapsulation.** Our `Bow` is only for `Cat`. You can make a field (method) `private` and hide it from other classes, using only inside the inner class.
2. It will **organize** your code and help your packages be more reasonable, as with all magic equipment for `Superhero` being in one place.

§6. Summary

You can create a class within another class and such classes are called nested. A non-static nested class is called an inner class. We hope you are ready to use it. Just don't forget to instantiate an outer class first!

The main idea of Inner classes is to hide some code from other classes and increase encapsulation.

Report a typo

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

Start practicing

Comments (0) Hints (2) Useful links (0) Show discussion