Java inner class and static nested class

What is the main difference between a inner class and a static nested class in Java? Does design /implementation play a role in choosing any of these?

java inner-classes

edited Nov 7 '08 at 15:50 skiphoppy 18.7k 41 129 177

asked Sep 16 '08 at 8:22 Omnipotent 4,722 6 22 33

16 Answers

Nested classes are divided into two categories: static and non-static. Nested classes that are declared static are simply called static nested classes. Non-static nested classes are called inner classes.

Static nested classes are accessed using the enclosing class name:

```
OuterClass.StaticNestedClass
```

For example, to create an object for the static nested class, use this syntax:

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

Objects that are instances of an inner class exist within an instance of the outer class. Consider the following classes:

```
class OuterClass {
    ...
    class InnerClass {
        ...
    }
}
```

An instance of InnerClass can exist only within an instance of OuterClass and has direct access to the methods and fields of its enclosing instance.

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();
```

see: Java Tutorial - Nested Classes

For completeness note that there is also such a thing as an inner class *without* an enclosing instance:

```
class A {
  int t() { return 1; }
  static A a =  new A() { int t() { return 2; } };
}
```

Here, `new A() { ... }` is an *inner class defined in a static context* and does not have an enclosing instance.

edited Aug 8 '14 at 7:06 Marko Topolnik 84.5k 7 87 154

answered Sep 16 '08 at 8:28 Martin 9,766 1 11 15

67 +1 less is more: explained exactly what I wanted in as short an answer as possible. – Cornelius Jan 19 '11 at 7:40

38 Mind that you can also import a static nested class directly, i.e. you could do (at the top of the file): `import OuterClass.StaticNestedClass;` then reference the class *just* as OuterClass. – Camilo Díaz Repka Jul 16 '11 at 0:22

14 Good answer, good write-up, -1 for your first paragraph being an un-credited direct quote from the Java Tutorial. Quote it & source it, and make it that much better!. – Richard Sitze Aug 3 '12 at 3:31

1 outerObject.new InnerClass(); thanks for this – Diljeet Mar 1 '13 at 19:29

1 @Ilya Kogan: I meant that I know we can access parent's class fields static or non-static (if the child class is non-static class) i have utilized their features couple of times, but I was wondering that what kind of memory model they follow? does OOP concepts cover them as a separate concept? if not where they really fit in within OOP. Seems like a friend class to me :) – Mubashar Ahmad Sep 9 '13 at 2:24

The Java tutorial says:

> Terminology: Nested classes are divided into two categories: static and non-static. Nested classes that are declared static are simply called static nested classes. Non-static nested classes are called inner classes.

In common parlance, the terms "nested" and "inner" are used interchangeably by most programmers, but I'll use the correct term "nested class" which covers both inner and static.

Classes can be nested *ad infinitum*, e.g. class A can contain class B which contains class C which contains class D, etc. However, more than one level of class nesting is rare, as it is generally bad design.

There are three reasons you might create a nested class:

- organization: sometimes it seems most sensible to sort a class into the namespace of another class, especially when it won't be used in any other context
- access: nested classes have special access to the variables/fields of their containing classes (precisely which variables/fields depends on the kind of nested class, whether inner or static).
- convenience: having to create a new file for every new type is bothersome, again, especially when the type will only be used in one context

There are **four kinds of nested class in Java**. In brief, they are:

- **static class**: declared as a static member of another class
- **inner class**: declared as an instance member of another class
- **local inner class**: declared inside an instance method of another class
- **anonymous inner class**: like a local inner class, but written as an expression which returns a one-off object

In more detail:

static classes

Static classes are the easiest kind to understand because they have nothing to do with instances of the containing class.

A static class is a class declared as a static member of another class. Just like other static members, such a class is really just a hanger on that uses the containing class as its namespace, *e.g.* the class *Goat* declared as a static member of class *Rhino* in the package *pizza* is known by the name *pizza.Rhino.Goat*.

```
package pizza;

public class Rhino {

    ...

    public static class Goat {
        ...
    }
}
```

Frankly, static classes are a pretty worthless feature because classes are already divided into namespaces by packages. The only real conceivable reason to create a static class is that

such a class has access to its containing class's private static members, but I find this to be a pretty lame justification for the static class feature to exist.

inner classes

An inner class is a class declared as a non-static member of another class:

```
package pizza;

public class Rhino {

    public class Goat {
        ...
    }

    private void jerry() {
        Goat g = new Goat();
    }
}
```

Like with a static class, the inner class is known as qualified by its containing class name, *pizza.Rhino.Goat*, but inside the containing class, it can be known by its simple name. However, every instance of an inner class is tied to a particular instance of its containing class: above, the *Goat* created in *jerry*, is implicitly tied to the *Rhino* instance *this* in *jerry*. Otherwise, we make the associated *Rhino* instance explicit when we instantiate *Goat*:

```
Rhino rhino = new Rhino();
Rhino.Goat goat = rhino.new Goat();
```

(Notice you refer to the inner type as just *Goat* in the weird *new* syntax: Java infers the containing type from the *rhino* part. And, yes *new rhino.Goat()* would have made more sense to me too.)

So what does this gain us? Well, the inner class instance has access to the instance members of the containing class instance. These enclosing instance members are referred to inside the inner class *via* just their simple names, not *via this* (*this* in the inner class refers to the inner class instance, not the associated containing class instance):

```
public class Rhino {

    private String barry;

    public class Goat {
        public void colin() {
            System.out.println(barry);
        }
    }
}
```

In the inner class, you can refer to *this* of the containing class as *Rhino.this*, and you can use *this* to refer to its members, *e.g. Rhino.this.barry*.

local inner classes

A local inner class is a class declared in the body of a method. Such a class is only known within its containing method, so it can only be instantiated and have its members accessed within its containing method. The gain is that a local inner class instance is tied to and can access the final local variables of its containing method. When the instance uses a final local of its containing method, the variable retains the value it held at the time of the instance's creation, even if the variable has gone out of scope (this is effectively Java's crude, limited version of closures).

Because a local inner class is neither the member of a class or package, it is not declared with an access level. (Be clear, however, that its own members have access levels like in a normal class.)

If a local inner class is declared in an instance method, an instantiation of the inner class is tied to the instance held by the containing method's *this* at the time of the instance's creation, and so the containing class's instance members are accessible like in an instance inner class. A local inner class is instantiated simply *via* its name, *e.g.* local inner class *Cat* is instantiated as *new Cat()*, not new this.Cat() as you might expect.

anonymous inner classes

An anonymous inner class is a syntactically convenient way of writing a local inner class. Most commonly, a local inner class is instantiated at most just once each time its containing method

is run. It would be nice, then, if we could combine the local inner class definition and its single instantiation into one convenient syntax form, and it would also be nice if we didn't have to think up a name for the class (the fewer unhelpful names your code contains, the better). An anonymous inner class allows both these things:

```
new *ParentClassName*(*constructorArgs*) {*members*}
```

This is an expression returning a new instance of an unnamed class which extends *ParentClassName*. You cannot supply your own constructor; rather, one is implicitly supplied which simply calls the super constructor, so the arguments supplied must fit the super constructor. (If the parent contains multiple constructors, the “simplest” one is called, “simplest” as determined by a rather complex set of rules not worth bothering to learn in detail--just pay attention to what NetBeans or Eclipse tell you.)

Alternatively, you can specify an interface to implement:

```
new *InterfaceName*() {*members*}
```

Such a declaration creates a new instance of an unnamed class which extends Object and implements *InterfaceName*. Again, you cannot supply your own constructor; in this case, Java implicitly supplies a no-arg, do-nothing constructor (so there will never be constructor arguments in this case).

Even though you can't give an anonymous inner class a constructor, you can still do any setup you want using an initializer block (a {} block placed outside any method).

Be clear that an anonymous inner class is simply a less flexible way of creating a local inner class with one instance. If you want a local inner class which implements multiple interfaces or which implements interfaces while extending some class other than *Object* or which specifies its own constructor, you're stuck creating a regular named local inner class.

edited Nov 28 '14 at 3:45 — EJP — 130k 11 83 145

answered Sep 16 '08 at 9:24 — Jegschemesch — 4,533 3 16 27

19 Great story, thanks. It has one mistake though. You can access the fields of an outer class from an instance inner class by Rhino.this.variableName . – Thirler Jul 30 '09 at 7:44

1 While you can't provide your own constructor for anonymous inner classes, you can use double brace initialisation. c2.com/cgi/wiki?DoubleBraceInitialization – Casebash Mar 2 '10 at 22:48

7 Great explanation, but I disagree with static inner classes being worthless. See rwhansen.blogspot.com/2007/07/… for a great variation of the builder pattern that depends heavily on the use of static inner classes. – Mansoor Siddiqui Jun 23 '11 at 21:14

2 I also disagree that static inner class is worthless: if you want an enum in the inner class you'll have to make the inner class static. – Someone Somewhere Feb 24 '12 at 18:52

1 Static inner classes also allow name-spacing of classes, similar to a static-import of data. One of the worst things about Java is the lack of aliasing. You either do a "*" import, or you import every single class. So all class names must be unique. Static inner classes can have shorter names. Also, they're all in one file, which some people like and some people hate. I like. – cdunn2001 Jan 23 '14 at 1:21

I don't think the real difference became clear in the above answers.

First to get the terms right:

- A nested class is a class which is contained in another class at the source code level.
- It is static if you declare it with the **static** modifier.
- A non-static nested class is called inner class. (I stay with non-static nested class.)

Martin's answer is right so far. However, the actual question is: What is the purpose of declaring a nested class static or not?

You use **static nested classes** if you just want to keep your classes together if they belong topically together or if the nested class is exclusively used in the enclosing class. There is no semantic difference between a static nested class and every other class.

Non-static nested classes are a different beast. Similar to anonymous inner classes, such nested classes are actually closures. That means they capture their surrounding scope and their enclosing instance and make that accessible. Perhaps an example will clarify that. See this stub of a Container:

```
public class Container {
    public class Item{
        Object data;
        public Container getContainer(){
            return Container.this;
        }
        public Item(Object data) {
            super();
            this.data = data;
        }

    }

    public static Item create(Object data){
        // does not compile since no instance of Container is available
        return new Item(data);
    }
    public Item createSubItem(Object data){
        // compiles, since 'this' Container is available
        return new Item(data);
    }
}
```

In this case you want to have a reference from a child item to the parent container. Using a non-static nested class, this works without some work. You can access the enclosing instance of Container with the syntax `Container.this`.

More hardcore explanations following:

If you look at the Java bytecodes the compiler generates for an (non-static) nested class it might become even clearer:

```
// class version 49.0 (49)
// access flags 33
public class Container$Item {

  // compiled from: Container.java
  // access flags 1
  public INNERCLASS Container$Item Container Item

  // access flags 0
  Object data

  // access flags 4112
  final Container this$0

  // access flags 1
  public getContainer() : Container
   L0
    LINENUMBER 7 L0
    ALOAD 0: this
    GETFIELD Container$Item.this$0 : Container
    ARETURN
   L1
    LOCALVARIABLE this Container$Item L0 L1 0
    MAXSTACK = 1
    MAXLOCALS = 1

  // access flags 1
  public <init>(Container,Object) : void
   L0
    LINENUMBER 12 L0
    ALOAD 0: this
    ALOAD 1
    PUTFIELD Container$Item.this$0 : Container
   L1
    LINENUMBER 10 L1
    ALOAD 0: this
    INVOKESPECIAL Object.<init>() : void
   L2
    LINENUMBER 11 L2
    ALOAD 0: this
    ALOAD 2: data
    PUTFIELD Container$Item.data : Object
    RETURN
   L3
    LOCALVARIABLE this Container$Item L0 L3 0
    LOCALVARIABLE data Object L0 L3 2
    MAXSTACK = 2
    MAXLOCALS = 3
}
```

As you can see the compiler creates a hidden field `Container this$0`. This is set in the constructor which has an additional parameter of type Container to specify the enclosing

instance. You can't see this parameter in the source but the compiler implicitly generates it for a nested class.

Martin's example

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

would so be compiled to a call of something like (in bytecodes)

```
new InnerClass(outerObject)
```

For the sake of completeness:

An anonymous class **is** a perfect example of a non-static nested class which just has no name associated with it and can't be referenced later.

edited Aug 27 '13 at 7:53

answered Sep 16 '08 at 9:12

jrudolph

4,251 2 12 25

7 "There is no semantic difference between a static nested class and every other class." Except the nested class can see the parent's private fields/methods and the parent class can see the nested's private fields/methods. – Brad Cupit Jun 22 '10 at 14:56

Wouldn't the non-static inner class potentially cause massive memory leaks? As in, every time you create a listener, you create a leak? – G_V Dec 15 '14 at 8:24

2 @G_V there's definitely potential for memory leaks because an instance of the inner class keeps a reference to the outer class. Whether this is an actual problem depends on where and how references to instances of the outer and the inner classes are held. – jrudolph Dec 17 '14 at 9:09

I think that none of the above answers explain to you the real difference between a nested class and a static nested class in term of application design :

OverView

A nested class could be nonstatic or static and in each case **is a class defined within another class**. **A nested class should exist only to serve is enclosing class**, if a nested class is useful by other classes (not only the enclosing), should be declared as a top level class.

Difference

Nonstatic Nested class : is implicitly associated with with the enclosing instance of the containing class, this means that it is possible to invoke methods and access variables of the enclosing instance. One common use of a nonstatic nested class is to define an Adapter class.

Static Nested Class : can't access enclosing class instance and invoke methods on it, so should be used when the nested class doesn't require access to an instance of the enclosing class . A common use of static nested class is to implement a components of the outer object.

Conclusion

So the main difference between the two from a design standpoint is : *nonstatic nested class can access instance of the container class, while static can't* .

answered May 27 '12 at 7:43

aleroot

33.5k 7 68 123

6 +1 for well organized answer. – srk Mar 16 '13 at 13:48

I think, the convention that is generally followed is this:

- **static class** within a top level class is a **nested class**
- **non static class** within a top level class is a **inner class**, which further has two more form:

- **local class** - named classes declared inside of a block like a method or constructor body
- **anonymous class** - unnamed classes whose instances are created in expressions and statements

However, few other **points to remembers** are:

- Top level classes and static nested class are semantically same except that in case of static nested class it can make static reference to private static fields/methods of its Outer [parent] class and vice versa.
- Inner classes have access to instance variables of the enclosing instance of the Outer [parent] class. However, not all inner classes have enclosing instances, for example inner classes in static contexts, like an anonymous class used in a static initializer block, do not.
- Anonymous class by default extends the parent class or implements the parent interface and there is no further clause to extend any other class or implement any more interfaces. So,
  - `new YourClass(){};` means `class [Anonymous] extends YourClass {}`
  - `new YourInterface(){};` means `class [Anonymous] implements YourInterface {}`

I feel that the bigger question that remains open which one to use and when? Well that mostly depends on what scenario you are dealing with but reading the reply given by @jrudolph may help you making some decision.

edited Jan 24 '13 at 5:40 — Sumit Singh — 9,848 2 28 56

answered Dec 16 '10 at 11:38 — sactiw — 8,962 2 14 18

In simple terms we need `nested classes` primarily because Java does not provide `closures`.

`Nested Classes` are classes defined inside the body of another enclosing class. They are of two types - `static and non-static`.

They are treated as members of the enclosing class, hence you can specify any of the four access specifiers - `private, package, protected, or public`. We don't have this luxury with top-level classes, which can only be declared `public or package`.

`Inner classes aka Non-stack classes` have access to other members of the top class, even if they are declared private while `Static nested classes` do not have access to other members of the top class.

```
public class OuterClass {
    public static class Inner1 {
    }
    public class Inner2 {
    }
}
```

Inner1 is our static inner class and Inner2 is our inner class which is not static. The key difference between them, you can't create an Inner2 instance without an Outer where as you can create an Inner1 object independently.

When would you use Inner class?.

Think of a situation where `Class A` and `Class B` are related, `Class B` needs to access `Class A` members and also `Class B` is related only to `Class A`. **Inner classes comes into the picture**.

For creating an instance of inner class, you need to create an instance of your outer class.

```
OuterClass outer = new OuterClass();
OuterClass.Inner2 inner = OuterClass.new Inner2();
```

When would you use static Inner class?.

You would define a static inner class when you know that it does not have any relationship with the instance of the `enclosing class/top class`. If your inner class doesn't use methods or fields of the outer class, it's just a waste of space, so make it static.

For example, to create an object for the static nested class, use this syntax:

```
OuterClass.Inner1 nestedObject = new OuterClass.Inner1();
```

The advantage of a `static nested class` is that it doesn't need an object of the containing class/top class to work. This can help you to reduce the number of objects your application creates at runtime.

answered Oct 17 '13 at 22:35

Thalaivar
7,312 1 18 40

1 did you mean `OuterClass.Inner2 inner = outer.new Inner2();` ? – Erik Allik Mar 12 '14 at 2:57

The instance of the inner class is created when instance of the outer class is created. Therefore the members and methods of the inner class have access to the members and methods of the instance (object) of the outer class. When the instance of the outer class goes out of scope, also the inner class instances cease to exist.

The static nested class doesn't have a concrete instance. It's just loaded when it's used for the first time (just like the static methods). It's a completely independent entity, whose methods and variables doesn't have any access to the instances of the outer class.

The static nested classes are not coupled with the outer object, they are faster, and they don't take heap/stack memory, because its not necessary to create instance of such class. Therefore the rule of thumb is to try to define static nested class, with as limited scope as possible (private >= class >= protected >= public), and then convert it to inner class (by removing "static" identifier) and loosen the scope, if it's really necessary.

answered Sep 16 '08 at 8:58

rmaruszewski
1,037 1 12 17

There is a subtlety about the use of nested static classes that might be useful in certain situations.

Whereas static attributes get instantiated before the class gets instantiated via its constructor, static attributes inside of nested static classes don't seem to get instantiated until after the class's constructor gets invoked, or at least not until after the attributes are first referenced, even if they are marked as 'final'.

Consider this example:

```
public class C0 {

    static C0 instance = null;

    // Uncomment the following line and a null pointer exception will be
    // generated before anything gets printed.
    //public static final String outerItem = instance.makeString(98.6);

    public C0() {
        instance = this;
    }

    public String makeString(int i) {
        return ((new Integer(i)).toString());
    }

    public String makeString(double d) {
        return ((new Double(d)).toString());
    }

    public static final class nested {
        public static final String innerItem = instance.makeString(42);
    }

    static public void main(String[] argv) {
        System.out.println("start");
        // Comment out this line and a null pointer exception will be
        // generated after "start" prints and before the following
        // try/catch block even gets entered.
        new C0();
```

```
        try {
            System.out.println("retrieve item: " + nested.innerItem);
        }
        catch (Exception e) {
            System.out.println("failed to retrieve item: " + e.toString());
        }
        System.out.println("finish");
    }
}
```

Even though 'nested' and 'innerItem' are both declared as 'static final'. the setting of nested.innerItem doesn't take place until after the class is instantiated (or at least not until after the nested static item is first referenced), as you can see for yourself by commenting and uncommenting the lines that I refer to, above. The same does not hold true for 'outerItem'.

At least this is what I'm seeing in Java 6.0.

edited Mar 26 '10 at 12:48

answered Mar 26 '10 at 12:38

HippoMan

31 2

Nested class: class inside class

```
   Types: 1.Static nested class
          2.Non-static nested class [Inner class]
```

Difference:

Non-static nested class [Inner class]

In non-static nested class object of inner class exist within object of outer class. So that data member of outer class is accessible to inner class. So to create object of inner class we must create object of outer class first.

```
outerclass outerobject=new outerobject();
outerclass.innerclass innerobjcet=outerobject.new innerclass();
```

Static nested class

In static nested class object of inner class don't need object of outer class, because the word "static" indicate no need to create object.

```
class outerclass A {
    static class nestedclass B {
        static int x = 10;
    }
}
```

If you want to access x, then write the following inside method

```
  outerclass.nestedclass.x;  i.e. System.out.prinltn( outerclass.nestedclass.x);
```

edited May 21 '13 at 18:32

Ephemerality

606 2 8 22

answered Apr 4 '10 at 10:14

tejas

31 1

The terms are used interchangeably. If you want to be really pedantic about it, then you *could* define "nested class" to refer to a static inner class, one which has no enclosing instance. In code, you might have something like this:

```
public class Outer {
    public class Inner {}

    public static class Nested {}
}
```

That's not really a widely accepted definition though.

answered Sep 16 '08 at 8:25

Daniel Spiewak
32.8k 8 74 105

'static inner' is a contradiction in terms. – EJP Nov 28 '14 at 3:47

Nested class is a very general term: every class which is not top level is a nested class. An inner class is a non-static nested class. Joseph Darcy wrote a very nice explanation about Nested, Inner, Member, and Top-Level Classes.

edited Aug 3 '12 at 2:31

Bill the Lizard ♦
162k 109 391 674

answered Sep 16 '08 at 8:35

Wouter Coekaerts
3,814 1 15 24

I think people here should notice to Poster that : Static Nest Class just only the first inner class. For example:

```
public static class A {} //ERROR

public class A {
    public class B {
        public static class C {} //ERROR
    }
}

public class A {
    public static class B {} //COMPILE !!!

}
```

So, summarize, static class doesn't depend which class its contains. So, they cannot in normal class. (because normal class need an instance).

edited Aug 21 '13 at 18:10

Nikhil Agrawal
5,026 8 37 71

answered Mar 28 '12 at 9:09

hqt
6,320 17 58 138

Mostly incomprehensible. – EJP Nov 28 '14 at 3:48

In the case of creating instance, the instance of non static inner class is created with the reference of object of outer class in which it is defined. This means it have inclosing instance. But the instance of static inner class is created with the reference of Outer class, not with the reference of object of outer class. This means it have not inclosing instance.

For example:

```
class A
{
  class B
  {
    // static int x; not allowed here…..
  }
  static class C
  {
    static int x; // allowed here
  }
}

class Test
{
  public static void main(String… str)
  {
    A o=new A();
    A.B obj1 =o.new B();//need of inclosing instance

    A.C obj2 =new A.C();

    // not need of reference of object of outer class….
```

```
  }
}
```

edited Nov 14 '13 at 12:43

Anders Gustafsson
8,010 1 17 55

answered Nov 14 '13 at 12:29

Ankit Jain
792 1 5 15

Ummm... an inner class IS a nested class... do you mean anonymous class and inner class?

Edit: If you actually meant inner vs anonymous... an inner class is just a class defined within a class such as:

```
public class A {
    public class B {
    }
}
```

Whereas an anonymous class is an extension of a class defined anonymously, so no actual "class is defined, as in:

```
public class A {
}

A anon = new A() { /* you could change behavior of A here */ };
```

Further Edit:

Wikipedia claims there is a difference in Java, but I've been working with Java for 8 years, and it's the first I heard such a distinction... not to mention there are no references there to back up the claim... bottom line, an inner class is a class defined within a class (static or not), and nested is just another term to mean the same thing.

There is a subtle difference between static and non-static nested class... basically non-static inner classes have implicit access to instance fields and methods of the enclosing class (thus they cannot be constructed in a static context, it will be a compiler error). Static nested classes, on the other hand, don't have implicit access to instance fields and methods, and CAN be constructed in a static context.

edited Nov 28 '14 at 3:49

EJP
130k 11 83 145

answered Sep 16 '08 at 8:24

Mike Stone
22.6k 14 82 121

Thanks!! Thats exactly what i wanted ;-) – Omnipotent Sep 16 '08 at 8:48

1 According to the Java documentation, there is a difference between an inner class and a static nested class -- static nested classes don't have references to their enclosing class and are used primarily for organization purposes. You should see Jegschemesch's reply for a more in-depth description. – mipadi Nov 7 '08 at 16:05

I think the semantic difference is mostly historical. When I wrote a C#->Java 1.1 compiler, the Java language reference was very explicit: nested class is static, inner class isn't (and thus has this$0). At any rate it's confusing and I'm glad it's no longer an issue. – Tomer Gabel Mar 26 '09 at 13:41

---

static nested classes are not inner classes while the non-static nested class are inner classes.

answered Jun 3 '14 at 17:35

saurabh kumar
212 3 13

First of all There is no such class called Static class.The Static modifier use with inner class (called as Nested Class) says that it is a static member of Outer Class which means we can access it as with other static members and without having any instance of Outer class. (Which is benefit of static originally.)

Difference between using Nested class and regular Inner class is:

OuterClass.InnerClass inner = new OuterClass().new InnerClass();

First We can to instantiate Outerclass then we Can access Inner.

But if Class is Nested then synatx is:

OuterClass.InnerClass inner = new OuterClass.InnerClass();

Which uses the static Syntax as normal implementation of static keyword.

answered Sep 10 '13 at 14:06

Sohi

30 4

"...says that it is a static member of Outer Class which means....": It is not incorrect to think of a static nested class as a "member class" of Outer Class, but the similarities with static fields and methods end there. A static nested class does not "belong" to Outer Class. In almost every way that matters, a static nested class is a free standing top level class whose class definition has been nested inside that of Outer Class for packaging convenience (and, hopefully, because there is a logical association between the nested class and Outer Class ... although there needn't be one). – scottb Oct 18 '13 at 0:03

'static inner' is a contradiction in terms. Static classes do exist, at the first nesting level, and they are not inner classes, by definition. Very confused. – EJP Nov 28 '14 at 3:50

protected by Brad Larson ♦ Feb 21 '14 at 18:33

Thank you for your interest in this question. Because it has attracted low-quality answers, posting an answer now requires 10 reputation on this site.

Would you like to answer one of these unanswered questions instead?