Java Inner Class

APRIL 3, 2018 BY PANKAJ — 10 COMMENTS

Java inner class is defined inside the body of another class. Java inner class can be declared private, public, protected, or with default access whereas an outer class can have only public or default access.

Java Nested classes are divided into two types.

1. **static nested class**

   If the nested class is static, then it's called static nested class. Static nested classes can access only static members of the outer class. Static nested class is same as any other top-level class and is nested for only packaging convenience.

   Static class object can be created with following statement.

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

2. **java inner class**

Any non-static nested class is known as inner class in java. Java inner class is associated with the object of the class and they can access all the variables and methods of the outer class.

Since inner classes are associated with instance, we can't have any static variables in them.

Object of java inner class are part of the outer class object and to create an instance of inner class, we first need to create instance of outer class.

Java inner class can be instantiated like this;

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

There are two special kinds of java inner classes.

1. **local inner class**

   If a class is defined in a method body, it's known as local inner class.

   Since local inner class is not associated with Object, we can't use private, public or protected access modifiers with it. The only allowed modifiers are abstract or final.

   A local inner class can access all the members of the enclosing class and local final variables in the scope it's defined.

   Local inner class can be defined as:

```
public void print() {
        //local inner class inside the method
        class Logger {
            String name;
        }
        //instantiate local inner class in the method to use
        Logger logger = new Logger();
```

2. **anonymous inner class**

   A local inner class without name is known as anonymous inner class. An anonymous class is defined and instantiated in a single statement.

   Anonymous inner class always extend a class or implement an interface. Since an anonymous class has no name, it is not possible to define a constructor for an anonymous class.

   Anonymous inner classes are accessible only at the point where it is defined.
   It's a bit hard to define how to create anonymous inner class, we will see it's real time usage in test program below.

Here is a java class showing how to define java inner class, static nested class, local inner class and anonymous inner class.

`OuterClass.java`

```
package com.journaldev.nested;

import java.io.File;
import java.io.FilenameFilter;

public class OuterClass {

    private static String name = "OuterClass";
    private int i;
    protected int j;
    int k;
    public int l;

    //OuterClass constructor
    public OuterClass(int i, int j, int k, int l) {
        this.i = i;
        this.j = j;
        this.k = k;
        this.l = l;
    }

    public int getI() {
        return this.i;
    }
```

Here is the test program showing how to instantiate and use inner class in java.

InnerClassTest.java

```
package com.journaldev.nested;

import java.util.Arrays;
//nested classes can be used in import for easy instantiation
import com.journaldev.nested.OuterClass.InnerClass;
import com.journaldev.nested.OuterClass.StaticNestedClass;

public class InnerClassTest {

    public static void main(String[] args) {
        OuterClass outer = new OuterClass(1,2,3,4);

        //static nested classes example
        StaticNestedClass staticNestedClass = new StaticNestedClass();
        StaticNestedClass staticNestedClass1 = new StaticNestedClass();

        System.out.println(staticNestedClass.getName());
        staticNestedClass.d=10;
        System.out.println(staticNestedClass.d);
        System.out.println(staticNestedClass1.d);

        //inner class example
        InnerClass innerClass = outer.new InnerClass();
        System.out.println(innerClass.getName());
        System.out.println(innerClass);
```

Here is the output of above java inner class example program.

```
OuterClass
10
0
OuterClass
w=0:x=0:y=0:z=0
w=1:x=2:y=3:z=4
Outer: OuterClass
Outer: 1
Outer: 2
Outer: 3
Outer: 4
[NestedClassTest.java, OuterClass.java]
[NestedClassTest.class, OuterClass$1.class, OuterClass$1Logger.class,
OuterClass$InnerClass.class, OuterClass$StaticNestedClass.class, OuterClass.class]
```

Notice that when OuterClass is compiled, separate class files are created for inner class, local inner class and static nested class.

Benefits of Java Inner Class

1. If a class is useful to only one class, it makes sense to keep it nested and together. It helps in packaging of the classes.
2. Java inner classes implements encapsulation. Note that inner classes can access outer class private members and at the same time we can hide inner class from outer world.
3. Keeping the small class within top-level classes places the code closer to where it is used and makes code more readable and maintainable.

That's all for java inner class.

FILED UNDER: JAVA

About Pankaj
If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.
Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

« Java continue statement Wrapper class in Java »

Comments

Amol Bhonsle says
JUNE 21, 2018 AT 9:28 PM
The statement about Method Local Inner Classes needs changes. It says "A local inner class can access all the members of the enclosing class and local final variables in the scope it's defined.".
The correct statement should be "A local inner class can access all the members of the enclosing class and local final variables in the scope it's defined. Additionally it can also access non final local variable of method in which it is defined, but it cannot modify them. So if you try to print non final local variable's value it will be allowed but if you try to change its value from inside method local inner class, you will get compile time Error"
Reply

Ganesh says
FEBRUARY 22, 2018 AT 2:12 AM
for local inner class we can access local non-final variable in the scope it's defined.(java version 8 and above)
Reply

Amol Bhonsle says
JUNE 21, 2018 AT 9:30 PM
True.....it can access only. But it cannot modify, thats the thing need to remember by developers.
Reply

Sundara Baskaran says
NOVEMBER 17, 2016 AT 6:10 PM
What are the benefits of annonymous inner class? Explain with example.
Reply

Purnendu Dutta says
OCTOBER 19, 2017 AT 1:14 PM
You forgot the magical word "Please".
Reply

Deepak says
JULY 17, 2015 AT 1:45 PM
Since inner classes are associated with instance, we can't have any static variables in them.... true but If we use 'static final int mytest=0 ;' it is allowed!
Reply

antoine cordier says
SEPTEMBER 10, 2014 AT 11:32 AM
Typo fix:
"Java nested classes are defined as class inside the body of another class. A nested class can be declared private, public, protected, or with default access whereas an outer class can have only **private** or default access."
private should be switched to private
Awesome work, Keep it up
Reply

Pankaj says
SEPTEMBER 10, 2014 AT 8:44 PM
Thanks for pointing out the typo error, fixed it.
Reply

Rishi Raj says
SEPTEMBER 30, 2013 AT 6:22 AM
Thanks, Pankaj, for sharing another helpful post.
But I want to add something to it:
In the first paragraph of this post, you have said: "... whereas an outer class can have only private or default access."
It should rather be: "... whereas an outer class can have only PUBLIC or default access." Do I make a point here?
Reply

chandrani says
APRIL 21, 2014 AT 11:48 AM
This is obviously an error, the author must have meant to write 'public' instead of 'private'.
Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Current ye@r *

5.2

Search for tutorials...

JET BRAINS

WebStor

Powerful IDE for mo
JavaScript develop

Try it now

DOWNLOAD ANDROID APP

GET IT ON Google Play

CORE JAVA TUTORIAL

- How to write a File in Java
- How to append data to a File in Java
- Converting InputStream to File
- Java Random Access File Example
- Download File from URL Example
- Java GZip Example
- Temp Files in Java
- Reading a CSV File using Java Scanner Class
- Java Property File Example

Java Exception Handling

- Exception Handling in Java
- java.lang.NoSuchMethodError
- java.lang.NullPointerException

MultiThreading and Concurrency

- Threads in Java
- Thread Life Cycle
- Thread Sleep Example
- Thread join Example
- Thread wait, notify, notifyAll Example
- Thread Safety in Java – Synchronization
- Java ThreadLocal Example
- Java Timer and TimerTask Example
- Java Thread Pool Example
- Java Callable Future Example
- Java FutureTask Example
- Java ScheduledThreadPoolExecutor Example
- Java Lock Example

Regular Expressions

- Regular Expressions in Java Example Tutorial
- Validating Email Address using Regular Expression in Java
- Validating Phone Number using Regex in Java

Advanced Java Concepts

- Java Heap Memory and Stack Memory
- Java is Pass by Value and not Pass by Reference
- JVM Memory Model and Garbage Collection
- Java Reflection API Tutorial
- Serialization in Java
- Java System Class
- Internationalization (i18n) in Java
- Atomic Operations in Java
- Thread Dump in Java
- Deadlocks in Java
- Sorting Objects in Java

- **Understanding JDK, JRE and JVM**
- **Java Classloader Example Tutorial**
- **Java clone object**

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial