Inner Class in Java

Difficulty Level : Medium Last Updated : 12 Jan, 2022

In Java, inner class refers to the class that is declared inside class or interface which were mainly introduced, to sum up, same logically relatable classes as Java is purely object-oriented so bringing it closer to the real world. Now geeks you must be wondering why they were introduced?

There are certain advantages associated with inner classes are as follows:

- Making code clean and readable.
- Private methods of the outer class can be accessed, so bringing a new dimension and making it closer to the real world.
- Optimizing the code module.

> *We do use them often as we go advance in java object-oriented programming where we want certain operations to be performed, granting access to limited classes and many more which will be clear as we do discuss and implement all types of inner classes in Java.*

Types of Inner Classes

There are basically four types of inner classes in java.

1. Nested Inner Class
2. Method Local Inner Classes
3. Static Nested Classes
4. Anonymous Inner Classes

Let us discuss each of the above following types sequentially in-depth alongside a clean java program which is very crucial at every step as it becomes quite tricky as we adhere forwards.

Type 1: Nested Inner Class

It can access any private instance variable of the outer class. Like any other instance variable, we can have access modifier private, protected, public, and default modifier. Like class, an interface can also be nested and can have access specifiers.

Example 1A

```java
// Java Program to Demonstrate Nested class

// Class 1
// Helper classes
class Outer {

    // Class 2
    // Simple nested inner class
    class Inner {

        // show() method of inner class
        public void show()
        {

            // Print statement
            System.out.println("In a nested class method");
        }
    }
}

// Class 2
// Main class
class Main {

    // Main driver method
    public static void main(String[] args)
    {

        // Note how inner class object is created inside
        // main()
        Outer.Inner in = new Outer().new Inner();

        // Calling show() method over above object created
        in.show();
    }
}
```

Output

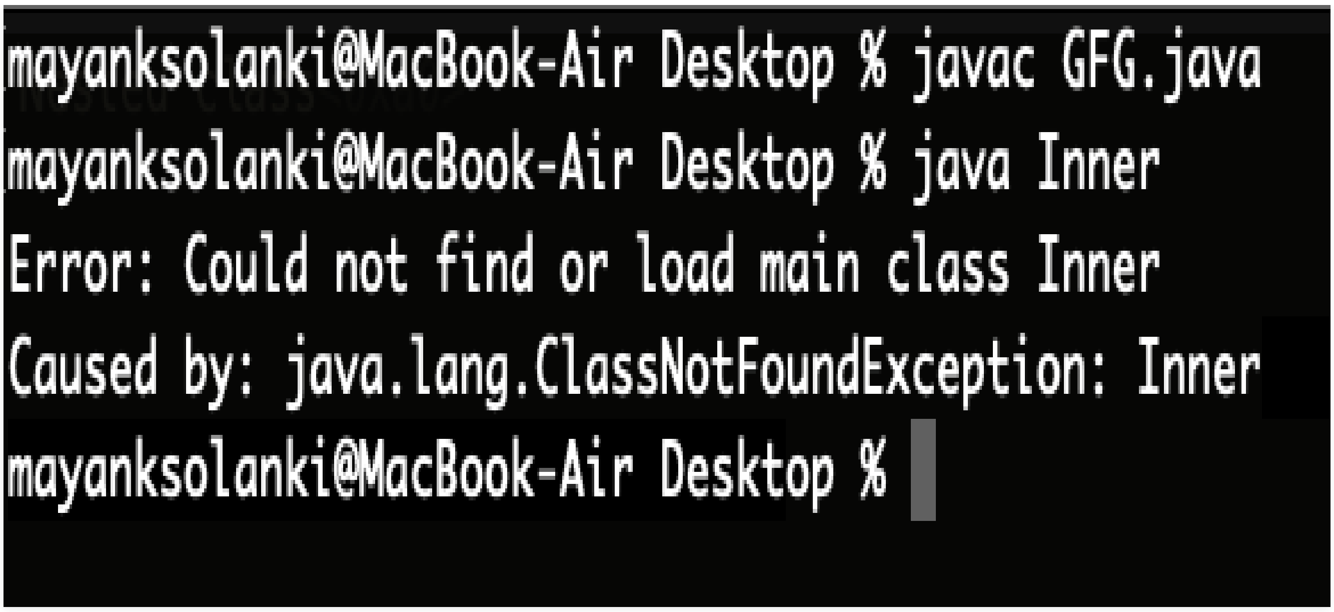
```
In a nested class method
```

Note: *We can not have a static method in a nested inner class because an inner class is implicitly associated with an object of its outer class so it cannot define any static method for itself. For example, the following program doesn't compile.*

Example 1B

```java
// Java Program to Demonstrate Nested class
// Where Error is thrown

// Class 1
// Outer class
class Outer {

    // Method defined inside outer class
    void outerMethod()
    {

        // Print statement
        System.out.println("inside outerMethod");
    }

    // Class 2
    // Inner class
    class Inner {

        // Main driver method
        public static void main(String[] args)
        {

            // Display message for better readability
            System.out.println("inside inner class Method");
        }
    }
}
```

Output:



```
mayanksolanki@MacBook-Air Desktop % javac GFG.java
mayanksolanki@MacBook-Air Desktop % java Inner
Error: Could not find or load main class Inner
Caused by: java.lang.ClassNotFoundException: Inner
mayanksolanki@MacBook-Air Desktop %
```

An interface can also be nested and nested interfaces have some interesting propertie s. We will be covering nested interfaces in the next post.

Type 2: Method Local Inner Classes

Inner class can be declared within a method of an outer class which we will be illustrating in the below example where Inner is an inner class in outerMethod().

Example 1

```java
// Java Program to Illustrate Inner class can be
// declared within a method of outer class

// Class 1
// Outer class
class Outer {

    // Method inside outer class
    void outerMethod()
    {

        // Print statement
        System.out.println("inside outerMethod");

        // Class 2
        // Inner class
        // It is local to outerMethod()
        class Inner {

            // Method defined inside inner class
            void innerMethod()
            {

                // Print statement whenever inner class is
                // called
                System.out.println("inside innerMethod");
            }
        }

        // Creating object of inner class
        Inner y = new Inner();

        // Calling over method defined inside it
        y.innerMethod();
    }
}

// Class 3
// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Creating object of outer class inside main()
        // method
        Outer x = new Outer();

        // Calling over the same method
        // as we did for inner class above
        x.outerMethod();
    }
}
```

Output

```
inside outerMethod
inside innerMethod
```

Method Local inner classes can't use a local variable of the outer method until that local variable is not declared as final. For example, the following code generates a compiler error.

> ***Note:*** *"x" is not final in outerMethod() and innerMethod() tries to access it.*

Example 2

```java
class Outer {
   void outerMethod() {
      int x = 98;
      System.out.println("inside outerMethod");
      class Inner {
         void innerMethod() {
            System.out.println("x= "+x);
         }
      }
      Inner y = new Inner();
      y.innerMethod();
   }
}
class MethodLocalVariableDemo {
   public static void main(String[] args) {
      Outer x=new Outer();
      x.outerMethod();
   }
}
```

Output

```
inside outerMethod
x= 98
```

> ***Note:*** *Local inner class cannot access non-final local variable till JDK 1.7. Since JDK 1.8, it is possible to access the non-final local variable in method local inner class.*

But the following code compiles and runs fine (Note that x is final this time)

Example 3

```
class Outer {
   void outerMethod() {
      final int x=98;
      System.out.println("inside outerMethod");
      class Inner {
         void innerMethod() {
            System.out.println("x = "+x);
         }
      }
      Inner y = new Inner();
      y.innerMethod();
   }
}
class MethodLocalVariableDemo {
    public static void main(String[] args){
      Outer x = new Outer();
      x.outerMethod();
    }
}
```

Output

```
inside outerMethod
x = 98
```

The main reason we need to declare a local variable as a final is that the local variable lives on the stack till the method is on the stack but there might be a case the object of the inner class still lives on the heap.
Method local inner class can't be marked as private, protected, static, and transient but can be marked as abstract and final, but not both at the same time.

Type 3: Static Nested Classes

Static nested classes are not technically inner classes. They are like a static member of outer class.

Example

```
// Java Program to Illustrate Static Nested Classes
// Importing required classes
import java.util.*;

// Class 1
// Outer class
class Outer {

    // Method
    private static void outerMethod()
    {

        // Print statement
        System.out.println("inside outerMethod");
    }

    // Class 2
    // Static inner class
    static class Inner {

        public static void display()
        {

            // Print statement
            System.out.println("inside inner class Method");

            // Calling method insid main() method
            outerMethod();
        }
    }
}

// Class 3
// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {

        Outer.Inner obj = new Outer.Inner();

        // Calling method via above instance created
        obj.display();
    }
}
```

Output

```
inside inner class Method
inside outerMethod
```

Type 4: Anonymous Inner Classes

Anonymous inner classes are declared without any name at all. They are created in two ways.

- As a subclass of the specified type
- As an implementer of the specified interface

Way 1: As a subclass of the specified type

Example:

```
// Java Program to Illustrate Anonymous Inner classes
// Declaration Without any Name
// As a subclass of the specified type

// Importing required classes
import java.util.*;

// Class 1
// Helper class
class Demo {

    // Method of helper class
    void show()
    {
        // Print statement
        System.out.println(
            "i am in show method of super class");
    }
}

// Class 2
// Main class
class Flavor1Demo {

    //  An anonymous class with Demo as base class
    static Demo d = new Demo() {
        // Method 1
        // show() method
        void show()
        {
            // Calling method show() via super keyword
            // which refers to parent class
            super.show();

            // Print statement
            System.out.println("i am in Flavor1Demo class");
        }
    };

    // Method 2
    // Main driver method
    public static void main(String[] args)
    {
        // Calling show() method inside main() method
        d.show();
    }
}
```

Output

```
i am in show method of super class
i am in Flavor1Demo class
```

In the above code, we have two classes Demo and Flavor1Demo. Here demo act as a super-class and the anonymous class acts as a subclass, both classes have a method show(). In anonymous class show() method is overridden.

Way 2: As an implementer of the specified interface

Example:

```
// Java Program to Illustrate Anonymous Inner Classes
// Declaration Without Any Name
// As an implementer of Specified interface

// Interface
interface Hello {

    // Method defined inside interface
    void show();
}

// Main class
class GFG {

    // Class implementing interface
    static Hello h = new Hello() {

        // Method 1
        // show() method inside main class
        public void show()
        {
            // Print statement
            System.out.println("i am in anonymous class");
        }
    };

    // Method 2
    // Main driver method
    public static void main(String[] args)
    {
        // Calling show() method inside main() method
        h.show();
    }
}
```

Output

```
i am in anonymous class
```

Output explanation:

In the above code, we create an object of anonymous inner class but this anonymous inner class is an implementer of the interface Hello. Any anonymous inner class can implement only one interface at one time. It can either extend a class or implement an interface at a time.

This article is contributed by **Pawan Kumar**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.