ARYATECHNO

Java Inner Classes

Topics : JAVA
Written on April 10, 2023

In Java, an inner class is a class defined within another class. There are four types of inner classes in Java:

1. Non-static nested class (also known as an inner class)
2. Static nested class
3. Local class
4. Anonymous class

Here is an example of a n**on-static nested class (inner class)** in Java:

```
public class OuterClass {
    private int outerVar;

    public void outerMethod() {
        InnerClass innerObj = new InnerClass();
        innerObj.innerMethod();
    }

    public class InnerClass {
        private int innerVar;

        public void innerMethod() {
            outerVar = 10;
            innerVar = 20;
            System.out.println("Inner class method: outerVar = " + outerVar +
", innerVar = " + innerVar);
        }
    }

public class Main {
    public static void main(String[] args) {
        OuterClass outerObj = new OuterClass();
        outerObj.outerMethod(); // Output: "Inner class method: outerVar =
10, innerVar = 20"
    }
}

}
```

In this example, `OuterClass` has an inner class called `InnerClass`. The `InnerClass` can access the members (fields and methods) of the `OuterClass`, including private members.

To create an instance of the `InnerClass`, we first need to create an instance of the `OuterClass`, and then use that instance to create an instance of the `InnerClass`. We can then call the `innerMethod()` of the `InnerClass`.

When we run the `main()` method, we create an instance of `OuterClass` and call its `outerMethod()`. Within `outerMethod()`, we create an instance of `InnerClass` and call its `innerMethod()`. The output shows that the `InnerClass` is able to access the `outerVar` variable of the `OuterClass`.

Non-static nested classes are useful for organizing code and making it more readable. They can also help to encapsulate implementation details and reduce the number of classes in a package.

In Java, **a static nested class** is a nested class that is declared with the `static` keyword. It is a class that is a static member of its outer class, and can be accessed using the name of the outer class. Here's an example of a static nested class in Java:

```
public class OuterClass {
    private static int outerVar = 10;

    public static class StaticNestedClass {
        private int innerVar;

        public void innerMethod() {
            innerVar = 20;
            System.out.println("Static nested class method: outerVar = " +
outerVar + ", innerVar = " + innerVar);
        }
    }
public class Main {
    public static void main(String[] args) {
        OuterClass.StaticNestedClass staticObj = new
OuterClass.StaticNestedClass();
        staticObj.innerMethod(); // Output: "Static nested class method:
outerVar = 10, innerVar = 20"
    }
}


}
```

When we run the `main()` method, we create an instance of `StaticNestedClass` using the syntax `OuterClass.StaticNestedClass`. We can then call the `innerMethod()` of the `StaticNestedClass`.

In Java, **a local class** is a class that is defined inside a method. Local classes have access to the variables and parameters of the enclosing method, but can only be accessed from within that

method. Here's an example of a local class in Java:

```
public class OuterClass {
    private int outerVar = 10;

    public void outerMethod() {
        int localVar = 20;

        class LocalClass {
            private int innerVar;

            public void innerMethod() {
                innerVar = 30;
                System.out.println("Local class method: outerVar = " +
outerVar + ", localVar = " + localVar + ", innerVar = " + innerVar);
            }
        }

        LocalClass localObj = new LocalClass();
        localObj.innerMethod(); // Output: "Local class method: outerVar =
10, localVar = 20, innerVar = 30"
    }
public class Main {
    public static void main(String[] args) {
        OuterClass outerObj = new OuterClass();
        outerObj.outerMethod(); // Output: "Local class method: outerVar =
10, localVar = 20, innerVar = 30"
    }
}

}
```

In this example, `OuterClass` has a method called `outerMethod()`, which contains a local class called `LocalClass`. The `LocalClass` can access the variables `outerVar` and `localVar` of the `outerMethod()`, as well as its own `innerVar` variable.

To create an instance of the `LocalClass`, we first need to call the `outerMethod()`, which will then create an instance of the `LocalClass`. We can then call the `innerMethod()` of the `LocalClass`.

In Java, **an anonymous class** is a class that is defined and instantiated at the same time, without giving it a name. Anonymous classes are often used for creating one-time use classes that implement an interface or extend a class. Here's an example of an anonymous class in Java:

```
public class OuterClass {
    private int outerVar = 10;
```

```
    public void outerMethod() {
        Interface innerObj = new Interface() {
            private int innerVar;

            @Override
            public void interfaceMethod() {
                innerVar = 20;
                System.out.println("Anonymous class method: outerVar = " +
outerVar + ", innerVar = " + innerVar);
            }
        };

        innerObj.interfaceMethod(); // Output: "Anonymous class method:
outerVar = 10, innerVar = 20"
    }
}

interface Interface {
    void interfaceMethod();
}


public class Main {
    public static void main(String[] args) {
        OuterClass outerObj = new OuterClass();
        outerObj.outerMethod(); // Output: "Anonymous class method: outerVar
= 10, innerVar = 20"
    }
}
```

Anonymous classes are useful when we need to create a small, one-time use class that implements an interface or extends a class. They can help to reduce the amount of code we need to write, and can make the code more concise and readable.