Java Inner Classes (Nested Classes)

Java inner class or nested class is a class that is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place to be more readable and maintainable.

Additionally, it can access all the members of the outer class, including private data members and methods.

Syntax of Inner class

```
1. class Java_Outer_class{
2.  //code
3.  class Java_Inner_class{
4.   //code
5.  }
6. }
```

Advantage of Java inner classes

There are three advantages of inner classes in Java. They are as follows:

1. Nested classes represent a particular type of relationship that is **it can access all the members (data members and methods) of the outer class,** including private.
2. Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
3. **Code Optimization**: It requires less code to write.

Need of Java Inner class

Sometimes users need to program a class in such a way so that no other class can access it. Therefore, it would be better if you include it within other classes.

If all the class objects are a part of the outer object then it is easier to nest that class inside the outer class. That way all the outer class can access all the objects of the inner class.

Do You Know

- What is the internal code generated by the compiler for member inner class?
- What are the two ways to create an anonymous inner class?
- Can we access the non-final local variable inside the local inner class?
- How to access the static nested class?
- Can we define an interface within the class?
- Can we define a class within the interface?

Difference between nested class and inner class in Java

An inner class is a part of a nested class. Non-static nested classes are known as inner classes.

Types of Nested classes

There are two types of nested classes non-static and static nested classes. The non-static nested classes are also known as inner classes.

- Non-static nested class (inner class)
    1. Member inner class
    2. Anonymous inner class
    3. Local inner class
- Static nested class

Type	Description
Member Inner Class	A class created within class and outside method.

Anonymous Inner Class	A class created for implementing an interface or extending class. The java compiler decides its name.
Local Inner Class	A class was created within the method.
Static Nested Class	A static class was created within the class.
Nested Interface	An interface created within class or interface.

Java Member Inner class

A non-static class that is created inside a class but outside a method is called **member inner class**. It is also known as a **regular inner class**. It can be declared with access modifiers like public, default, private, and protected.

Syntax:

```
class Outer{
 //code
 class Inner{
  //code
 }
}
```

Java Member Inner Class Example

In this example, we are creating a msg() method in the member inner class that is accessing the private data member of the outer class.

TestMemberOuter1.java

```
class TestMemberOuter1{
 private int data=30;
 class Inner{
  void msg(){System.out.println("data is "+data);}
 }
 public static void main(String args[]){
  TestMemberOuter1 obj=new TestMemberOuter1();
  TestMemberOuter1.Inner in=obj.new Inner();
  in.msg();
 }
}
```

Output:

```
data is 30
```

How to instantiate Member Inner class in Java?

An object or instance of a member's inner class always exists within an object of its outer class. The new operator is used to create the object of member inner class with slightly different syntax.

The general form of syntax to create an object of the member inner class is as follows:

Syntax:

```
1. OuterClassReference.new MemberInnerClassConstructor();
```

Example:

```
1. obj.new Inner();
```

Here, OuterClassReference is the reference of the outer class followed by a dot which is followed by the new operator.

Internal working of Java member inner class

The java compiler creates two class files in the case of the inner class. The class file name of the inner class is "Outer$Inner". If you want to instantiate the inner class, you must have to create the instance of the outer class. In such a case, an instance of inner class is created inside the instance of the outer class.

Internal code generated by the compiler

The Java compiler creates a class file named Outer$Inner in this case. The Member inner class has the reference of Outer class that is why it can access all the data members of Outer class including private.

```
1. import java.io.PrintStream;
2. class Outer$Inner
3. {
4.    final Outer this$0;
5.    Outer$Inner()
6.    {  super();
7.       this$0 = Outer.this;
```

```
8.      }
9.      void msg()
10.     {
11.         System.out.println((new StringBuilder()).append("data is ")
12.                     .append(Outer.access$000(Outer.this)).toString());
13.     }
14. }
```

Java Anonymous inner class

Java anonymous inner class is an inner class without a name and for which only a single object is created. An anonymous inner class can be useful when making an instance of an object with certain "extras" such as overloading methods of a class or interface, without having to actually subclass a class.

In simple words, a class that has no name is known as an anonymous inner class in Java. It should be used if you have to override a method of class or interface. Java Anonymous inner class can be created in two ways:

1. Class (may be abstract or concrete).
2. Interface

Java anonymous inner class example using class

TestAnonymousInner.java

```
1. abstract class Person{
2.   abstract void eat();
3. }
4. class TestAnonymousInner{
5.  public static void main(String args[]){
6.   Person p=new Person(){
7.   void eat(){System.out.println("nice fruits");}
8.   };
9.   p.eat();
10.  }
11. }
```

Output:

```
nice fruits
```

Internal working of given code

```
1. Person p=new Person(){
2. void eat(){System.out.println("nice fruits");}
```

3. };

    1. A class is created, but its name is decided by the compiler, which extends the Person class and provides the implementation of the eat() method.
    2. An object of the Anonymous class is created that is referred to by 'p,' a reference variable of Person type.

Internal class generated by the compiler

```
1. import java.io.PrintStream;
2. static class TestAnonymousInner$1 extends Person
3. {
4.   TestAnonymousInner$1(){}
5.   void eat()
6.   {
7.       System.out.println("nice fruits");
8.   }
9. }
```

Java anonymous inner class example using interface

```
1. interface Eatable{
2.  void eat();
3. }
4. class TestAnnonymousInner1{
5.  public static void main(String args[]){
6.  Eatable e=new Eatable(){
7.   public void eat(){System.out.println("nice fruits");}
8.  };
9.  e.eat();
10. }
11. }
```

Output:

```
nice fruits
```

Internal working of given code

It performs two main tasks behind this code:

```
1. Eatable p=new Eatable(){
2. void eat(){System.out.println("nice fruits");}
3. };
```

1. A class is created, but its name is decided by the compiler, which implements the Eatable interface and provides the implementation of the eat() method.
2. An object of the Anonymous class is created that is referred to by 'p', a reference variable of the Eatable type.

Internal class generated by the compiler

```
1. import java.io.PrintStream;
2. static class TestAnonymousInner1$1 implements Eatable
3. {
4. TestAnonymousInner1$1(){}
5. void eat(){System.out.println("nice fruits");}
6. }
```

Java Local inner class

A class i.e., created inside a method, is called local inner class in java. Local Inner Classes are the inner classes that are defined inside a block. Generally, this block is a method body. Sometimes this block can be a for loop, or an if clause. Local Inner classes are not a member of any enclosing classes. They belong to the block they are defined within, due to which local inner classes cannot have any access modifiers associated with them. However, they can be marked as final or abstract. These classes have access to the fields of the class enclosing it.

If you want to invoke the methods of the local inner class, you must instantiate this class inside the method.

Java local inner class example

LocalInner1.java

```
1.  public class localInner1{
2.   private int data=30;//instance variable
3.   void display(){
4.    class Local{
5.     void msg(){System.out.println(data);}
6.    }
7.    Local l=new Local();
8.    l.msg();
9.   }
10.  public static void main(String args[]){
11.   localInner1 obj=new localInner1();
12.   obj.display();
13.  }
14. }
```

Output:

```
30
```

Internal class generated by the compiler

In such a case, the compiler creates a class named Simple$1Local that has the reference of the outer class.

```
1. import java.io.PrintStream;
2. class localInner1$Local
3. {
4.     final localInner1 this$0;
5.     localInner1$Local()
6.     {
7.         super();
8.         this$0 = Simple.this;
9.     }
10.    void msg()
11.    {
12.        System.out.println(localInner1.access$000(localInner1.this));
13.    }
14. }
```

Rule: Local variables can't be private, public, or protected.

Rules for Java Local Inner class

1) Local inner class cannot be invoked from outside the method.

2) 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 the local inner class.

Example of local inner class with local variable

LocalInner2.java

```
1. class localInner2{
2.  private int data=30;//instance variable
3.  void display(){
4.   int value=50;//local variable must be final till jdk 1.7 only
5.   class Local{
```

```
6.     void msg(){System.out.println(value);}
7.    }
8.    Local l=new Local();
9.    l.msg();
10.  }
11.  public static void main(String args[]){
12.   localInner2 obj=new localInner2();
13.   obj.display();
14.  }
15. }
```

Output:

```
50
```

Java static nested class

A static class is a class that is created inside a class, is called a static nested class in Java. It cannot access non-static data members and methods. It can be accessed by outer class name.

- It can access static data members of the outer class, including private.
- The static nested class cannot access non-static (instance) data members or

Java static nested class example with instance method

TestOuter1.java

```
1.  class TestOuter1{
2.    static int data=30;
3.    static class Inner{
4.     void msg(){System.out.println("data is "+data);}
5.    }
6.    public static void main(String args[]){
```

```
7.    TestOuter1.Inner obj=new TestOuter1.Inner();
8.    obj.msg();
9.    }
10. }
```

Output:

```
data is 30
```

In this example, you need to create the instance of static nested class because it has instance method msg(). But you don't need to create the object of the Outer class because the nested class is static and static properties, methods, or classes can be accessed without an object.

Internal class generated by the compiler

```
1. import java.io.PrintStream;
2. static class TestOuter1$Inner
3. {
4. TestOuter1$Inner(){}
5. void msg(){
6. System.out.println((new StringBuilder()).append("data is ")
7. .append(TestOuter1.data).toString());
8. }
9. }
```

Java static nested class example with a static method

If you have the static member inside the static nested class, you don't need to create an instance of the static nested class.

TestOuter2.java

```
1. public class TestOuter2{
2.   static int data=30;
3.   static class Inner{
4.    static void msg(){System.out.println("data is "+data);}
5.   }
```

```
6.    public static void main(String args[]){
7.    TestOuter2.Inner.msg();//no need to create the instance of static nested class
8.    }
9.  }
```

Output:

```
10.  data is 30
```

Java Nested Interface

An interface, i.e., declared within another interface or class, is known as a nested interface. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred to by the outer interface or class. It can't be accessed directly.

Points to remember for nested interfaces

There are given some points that should be remembered by the java programmer.

- The nested interface must be public if it is declared inside the interface, but it can have any access modifier if declared within the class.
- Nested interfaces are declared static

Syntax of nested interface which is declared within the interface

```
1. interface interface_name{
2.  ...
3.  interface nested_interface_name{
4.   ...
5.  }
6. }
```

Syntax of nested interface which is declared within the class

```
1. class class_name{
2.  ...
3.  interface nested_interface_name{
4.   ...
5.  }
6. }
```

Example of nested interface which is declared within the interface

In this example, we will learn how to declare the nested interface and how we can access it.

TestNestedInterface1.java