Java Inner Classes

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

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

Additionally, it can access all the members of 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 basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of 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.

Difference between nested class and inner class in Java

Inner class is a part of 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 interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.
Static Nested Class	A static class created within 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.

Syntax:

```
1. class Outer{
2.  //code
3.  class Inner{
4.   //code
5.  }
6. }
```

Java Member inner class example

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

```
1. class TestMemberOuter1{
2.  private int data=30;
3.  class Inner{
4.   void msg(){System.out.println("data is "+data);}
5.  }
6.  public static void main(String args[]){
7.   TestMemberOuter1 obj=new TestMemberOuter1();
8.   TestMemberOuter1.Inner in=obj.new Inner();
9.   in.msg();
10.      }
11.     }
```

Internal working of Java member inner class

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

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

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

Java anonymous inner class example using class

```
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.     }
```

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 Anonymous class is created that is referred by p 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.     }
```

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 Anonymous class is created that is referred by p reference variable of 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. If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

Java local inner class example

```
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.     }
```

Internal class generated by the compiler

In such case, compiler creates a class named Simple$1Local that have 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 variable 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 local inner class.

Example of local inner class with local variable

```
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.     }
```

Java static nested class

A static class i.e. created inside a class is called 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 outer class including private.
- Static nested class cannot access non-static (instance) data member or method.

Java static nested class example with instance method

```
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.      }
```

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 Outer class because nested class is static and static properties, methods or classes can be accessed without 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 static method

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

```
1. 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. }
```

Java Nested Interface

An interface i.e. declared within another interface or class is known as 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 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.

- 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 implicitely.

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 are going to learn how to declare the nested interface and how we can access it.

```
1. interface Showable{
2.   void show();
3.   interface Message{
4.    void msg();
5.   }
6. }
7. class TestNestedInterface1 implements Showable.Message{
8.  public void msg(){System.out.println("Hello nested interface");}
9.
10.      public static void main(String args[]){
11.       Showable.Message message=new TestNestedInterface1();//upcasting here
12.       message.msg();
13.      }
14.     }
```

As you can see in the above example, we are acessing the Message interface by its outer interface Showable because it cannot be accessed directly. It is just like almirah inside the room, we cannot

access the almirah directly because we must enter the room first. In collection frameword, sun microsystem has provided a nested interface Entry. Entry is the subinterface of Map i.e. accessed by Map.Entry.

Internal code generated by the java compiler for nested interface Message

The java compiler internally creates public and static interface as displayed below:.

```
1. public static interface Showable$Message
2. {
3.   public abstract void msg();
4. }
```

Example of nested interface which is declared within the class

Let's see how can we define an interface inside the class and how can we access it.

```
1. class A{
2.   interface Message{
3.    void msg();
4.   }
5. }
6.
7. class TestNestedInterface2 implements A.Message{
8.  public void msg(){System.out.println("Hello nested interface");}
9.
10.       public static void main(String args[]){
11.        A.Message message=new TestNestedInterface2();//upcasting here
12.        message.msg();
13.       }
14.      }
```

Can we define a class inside the interface?

Yes, If we define a class inside the interface, java compiler creates a static nested class. Let's see how can we define a class within the interface:

```
1. interface M{
2.   class A{}
3. }
```