Nested Classes in java

- In Java, it is possible to define a class within another class, such classes are known as nested classes. They enable you to logically group classes that are only used in one place, thus this increases the use of encapsulation, and creates more readable and maintainable code.
- A nested class has access to the members, including private members, of the class in which it is nested. However, the reverse is not true i.e., the enclosing class does not have access to the members of the nested class.
- A nested class is also a member of its enclosing class.
- As a member of its enclosing class, a nested class can be declared private, public, protected, or package private(default).

Why Use Nested Classes?

- **Logical grouping of classes**—If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together.

- **Increased encapsulation**—Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.

- **More readable, maintainable code**—Nesting small classes within top-level classes places the code closer to where it is used.

Nested classes are divided into two categories:

static nested class :

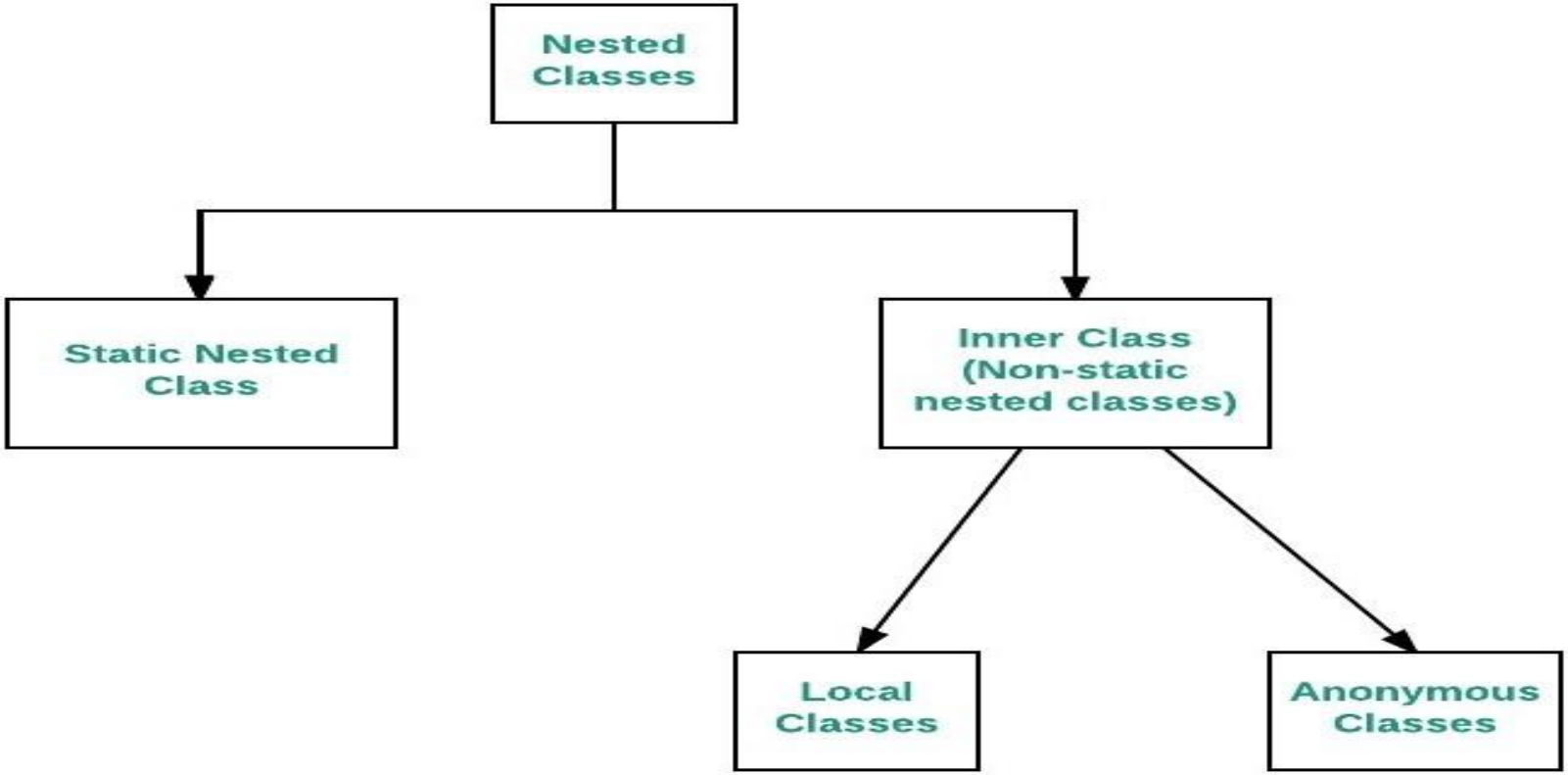
Nested classes that are declared static are called static nested classes.

inner class :

An inner class is a non-static nested class.

Syntax:

```
class OuterClass
{
...
    class NestedClass
    {
        ...
    }
}
```



Static Nested Classes

- A static nested class is associated with its outer class similar to class methods and variables.
- A static nested class cannot refer directly to instance variables or methods defined in its enclosing class.
- It can use them only through an object reference.
- Static nested classes are accessed using the enclosing class name:

  OuterClass.StaticNestedClass

- For example, to create an object for the static nested class, use this syntax:

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

```
// a static nested class
// outer class
class OuterClass
{
    static int outer_x = 10;
    int outer_y = 20;
private static int outer_private = 30;
   static class StaticNestedClass
   {
      void display()
      {
         // can access static member of outer class
         System.out.println("outer_x = " + outer_x);

         // can access display private static member of outer class
         System.out.println("outer_private = " + outer_private);

         // The following statement will give compilation error
  // as static nested class cannot directly access non-static members
         // System.out.println("outer_y = " + outer_y);

      }
   }
}
```

```
public class StaticNestedClassDemo
{
   public static void main(String[] args)
   {
OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();

       nestedObject.display();

   }
}
```

Inner Classes(Non-static nested classes)

- An inner class is associated with an instance of its enclosing class and has direct access to that object's methods and fields.

- Because an inner class is associated with an instance, it cannot define any static members itself.

- Objects that are instances of an inner class exist *within* an instance of the outer class.
- Consider the following classes:

```
class OuterClass {
                    ...
                    class InnerClass { ... }
}
```

- An instance of InnerClass can exist only within an instance of OuterClass and has direct access to the methods and fields of its enclosing instance.

- To instantiate an inner class, we must first instantiate the outer class. Then, create the inner object within the outer object.

- Syntax:

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

- Non-static nested classes(or inner classes comes in three variations):
  - **Member inner classes**(A class created within class and outside method.)
    - **Local inner classes**(A class created inside method/or inside any block)
    - **Anonymous classes** (also called anonymous inner classes)[class with no name]

Example 1

CODERINDEED

```
// Java program to demonstrate accessing  a member
inner class


class OuterClass
{
static int outer_x = 10;
 int outer_y = 20;
 private int outer_private = 30;
class InnerClass
   {
      void display()
      {
System.out.println("outer_x = " + outer_x);
System.out.println("outer_y = " + outer_y);
System.out.println("outer_private = " + outer_private);


      }
   }
}
```

```
// Driver class
public class InnerClassDemo
{
   public static void main(String[] args)
   {
OuterClass outerObject = new OuterClass();
OuterClass.InnerClass innerObject =
outerObject.new InnerClass();
     innerObject.display();


   }
}
```

Example 2

```
class TestMemberOuter1{
 private int data=30;
 class Inner{
  void msg(){System.out.println("data is "+data);}
 }

 void display(){
  Inner in=new Inner();
  in.msg();
 }
 public static void main(String args[]){
  TestMemberOuter1 obj=new TestMemberOuter1();
  obj.display();
 }
}
```

Local Inner Classes

- Local classes are *classes that are defined in a block*, which is a group of zero or more statements between balanced braces.

For example, we can define a local class in a method body, a for loop, or an if clause.

- A local class has access to the members of its enclosing class.
- The scope of local inner class is restricted to the block they are defined in.
- Local inner class cannot be instantiated from outside the block where it is created in.
- Till JDK 7,Local inner class can access only final local variable of the enclosing block. However From JDK 8, it is possible to access the non-final local variable of enclosing block in local inner class.

More points of Local Inner Class

- A local class has access to the members of its enclosing class.
- Local inner classes can extend an abstract class or can also implement an interface.
- Local inner could be abstract in nature also
- Local inner classes cannot be provided any access modifier like: private, public or protected

What happens at compile time?

When a program containing a local inner class is compiled, the compiler generate two .class files, one for the outer class and the other for the inner class that has the reference to the outer class. The two files are named by compiler as: Outer.class

Outer$1Inner.class

Example of Local class(Defined inside method)


```
class abc
{
   int a=12,b=34;
   void calculate()
   {
      class def
      {
         int sumCalculate()
         {
            return (a+b);
         }
         int mulCalculate()
         {
            return (a*b);
         }
      }
      def ob=new def();
      System.out.println("Sum is "+ ob.sumCalculate());
      System.out.println("Mul is "+ ob.mulCalculate());
   }
}
```

```
public class Main
{
        public static void main(String[] args)
        {
                abc ob = new abc();
                ob.calculate();
        }
}
```

The inner class cannot be declared as static. Inner classes are associated with the block they are defined within and not with the external class(Outer in this case).

Java code to demonstrate the scope of inner class

```
public class Outer
{               private void myMethod()
                {
                                class Inner
                                {
                                                private void innerMethod()
                                                {
                                                                System.out.println("Inside inner class");
                                                }
                                }
                }
                public static void main(String[] args)
                {
                                Outer outer = new Outer();
                                Inner inner = new Inner();
                                inner.innerMethod();
                }
}
```

The above program causes compilation error because the scope of inner classes are restricted to the block they are defined in.

Anonymous Classes

- Anonymous classes enable us to declare and instantiate a class at the same time.
- They are like local classes except that they do not have a name.
- The anonymous class expression consists of the following:

  1. *The new operator*

  2. The name of an interface to implement or a class to extend.

  3. Parentheses that contain the arguments to a constructor, just like a normal class instance creation expression.

  4. A body, which is a class declaration body. More specifically, in the body, method declarations are allowed but statements are not.

- *Anonymous classes have the same access to local variables of the enclosing scope as local classes:*

- An anonymous class has access to the members of its enclosing class.

- *Anonymous classes also have the same restrictions as local classes with respect to their members:*

- We cannot declare static initializers or member interfaces in an anonymous class.
- An anonymous class can have static members provided that they are constant variables.

- *Note that we can declare the following in anonymous classes*:

- Fields
- Extra methods (even if they do not implement any methods of the supertype)
- Local classes
- We cannot declare constructors in an anonymous class.

Note:

When we compile a nested class, two different class files will be created with names

Outerclass.class

Outerclass$Nestedclass.class

- Local Class is named as Outerclass$1Localclass.class
- Anonymous class is named as Outerclass$1.class

Syntax: The syntax of an anonymous class expression is like the invocation of a constructor, except that there is a class definition contained in a block of code.

```
// Test can be interface,abstract/concrete class
Test t = new Test()
{
   // data members and methods
   public void test_method()
   {
      .........
      .........
    }
};
```

Program to understand the need of anonymous class.


```
//Java program to demonstrate need for Anonymous Inner class
interface Age
{
        int x = 21;
        void getAge();
}
class AnonymousDemo
{
        public static void main(String[] args)
        {
                // Myclass is implementation class of Age interface
                MyClass obj=new MyClass();

                // calling getage() method implemented at Myclass
                obj.getAge();
        }
}

// Myclass implement the methods of Age Interface
class MyClass implements Age
{
        @Override
        public void getAge()
        {
                // printing the age
                System.out.print("Age is "+x);
        }
}
```

In the program, interface Age is created with getAge() method and x=21. Myclass is written as implementation class of Age interface. As done in Program, there is no need to write a separate class Myclass. Instead, directly copy the code of Myclass into this parameter, as shown here:

```
Age oj1 = new Age() {
        @Override
        public void getAge() {
            System.out.print("Age is "+x);
        }
    };
```

```
//Java program to demonstrate Anonymous inner class
interface Age
{
        int x = 21;
        void getAge();
}
class AnonymousDemo
{
        public static void main(String[] args) {
                Age oj1 = new Age() {
                        @Override
                        public void getAge() {
                                // printing age
                                System.out.print("Age is "+x);
                        }
                };
                oj1.getAge();
        }
}
```

Types of anonymous inner class

Based on declaration and behavior, there are 3 types of anonymous Inner classes:

1. Anonymous Inner class that extends a class :

We can have an anonymous inner class that extends a class.

```
class A
{
   int x=12;
   void show()
   {
      System.out.println("Hello World");
   }
}

class Main
{
            public static void main(String[] args)
            {
                        A ob = new A() // annoymous inner class
                        {
                           void show()
                           {
                              System.out.println("Overridden " +x);
                           }
                        };
                        ob.show();
            }
}
```

2. Anonymous Inner class that implements an interface

We can also have an anonymous inner class that implements an interface

```
interface  A
{
   int x=12;
   void show();
}

class Main
{
            public static void main(String[] args)
            {
                        A ob = new A() // annoymous inner class
                        {
                           public void show()
                           {
                              System.out.println("Overridden " +x);
                           }
                        };
                        ob.show();
            }
}
```

3. Anonymous class that extends abstract class


```
abstract class Example
{
   abstract void display();
}
class Main
{
        public static void main(String[] args)
        {
                Example ref = new Example() // annoymous inner class
                {
                   void display(){ System.out.print("Hello"); }
                };
                ref.display();
        }
}
```

Comparison between normal or regular class and static nested class

S.NO	Normal/Regular inner class	Static nested class
	Without an outer class object existing, there cannot be an inner class object. That is, the inner class object is always associated with the outer class object.	Without an outer class object existing, there may be a static nested class object. That is, static nested class object is not associated with the outer class object.
	Inside normal/regular inner class, static members can't be declared.	Inside static nested class, static members can be declared.
	Both static and non static members of outer class can be accessed directly.	Only a static member of outer class can be accessed directly.

Difference between Normal/Regular class and Anonymous Inner class:

- A normal class can implement any number of interfaces but anonymous inner class can implement only one interface at a time.
- A regular class can extend a class and implement any number of interface simultaneously. But anonymous Inner class can extend a class or can implement an interface but not both at a time.
- For regular/normal class, we can write any number of constructors but we cant write any constructor for anonymous Inner class because anonymous class does not have any name and while defining constructor class name and constructor name must be same.

Q1(Output??)

CODERINDEED

```
class A
{
   private static int x=100;
   private String y="Hello";
   static class B
   {
      void show()
      {
      System.out.print(x+" ");
      System.out.print(y);
      }
   }
}
public class Main
{
            public static void main(String[] args) {
                        A.B ref=new A.B();
                        ref.show();
            }
}
```

A. 100 Hello
B. 0 Hello
C. 100
D. Compile time error

Q2(Output??)

CODERINDEED

```
class A
{
   private static int x=100;
   class B
   {
   private int y=100;
   }
   void display()
   {
      System.out.println(x+y);
   }
}
public class Main
{
            public static void main(String[] args) {
                        A ref=new A();
                        ref.display();
            }
}
```

A. 100

B. 200

C. Compile time error

D. 0

Q3(Output??)

CODERINDEED

```
class A
{
   private static int x=10;
   class B
   {
   private int y=20;
   void display()
   {
      System.out.println(x+y);
   }
   }
}
public class Main
{
            public static void main(String[] args) {
                        A ref1=new A();
                        A.B ref2=ref1.new B();
                        ref2.display();
            }
}
```

A. 10
B. 20
C. 30
D. 0

Q4(Output??)


```
class Example
{
   void show()
   {
      class Test
      {
         static int a=13;
         void display()
         {
            System.out.println(a);
         }
      }
      Test ob=new Test();
      ob.display();
   }
}
public class Main
{
public static void main(String[] args)
                {
     Example obj=new Example();
     obj.show();
                }
}
```

A. 13
B. 0
C. Compile time error
D. Runtime error

Q5(Output??)

CODERINDEED

```
interface Example
{
   int a=100;
   void show();
}
class Main
{
        public static void main(String[] args)
        {
Example ob = new Example() // anonymous inner class
                {
                   static final int b=200;
                   public void show()
                   {
                      System.out.println(a+" "+b);
                   }
                };
                ob.show();
        }
}
```

A. 100 0
B. 0 200
C. Compile time error
D. 100 200

Q6(Output??)

CODERINDEED

```
abstract class ABC
{
   abstract void show1();
   abstract void show2();
}
class Main
{
            public static void main(String[] args)
            {
ABC ob = new ABC() // anonymous inner class
                        {
void show1(){ System.out.print("A "); }
void show2(){ System.out.print("B "); }
                        };
                        ob.show1();
                        ob.show2();
            }
}
```

OP1: A B

OP2: Compile time error

OP3:Runtime error

OP4:Blank output

Q7

Which of the following is true regarding anonymous inner class?

A. It can have constructors
B. We can create only one instance of anonymous class
C. It cannot have final static variables
D. It cannot have final non-static variables

Q8(Output??)

CODERINDEED

```
class outer
{
   int x=2;
   static class inner
   {
      int y=3;
      void display()
      {
      System.out.println(new outer().x+y);
      }
   }
}
public class Main
{
            public static void main(String[] args) {
            outer.inner ref=new outer.inner();
            ref.display();
            }
}
```

A. 5
B. 2
C. Compile time error
D. Blank output

Q9(Output??)

CODERINDEED

```
class A
{
   class B
   {
      static void methodB()
      {
         System.out.println("Method B");
      }
   }
}
public class Main
{
public static void main (String[] args)
{
A obj1=new A();
A.B obj2=obj1.new B();
}
}
```

A. Method B

B. Compile time error

C. Runtime error

D. Blank output

Q10(Output??)

CODERINDEED

```
class X
{
   static int x = 100;

   static class Y
   {
      static int y = x++;

      static class Z
      {
         static int z = y++;
      }
   }
}
public class Main
{
   public static void main(String[] args)
   {
      System.out.print(X.x+" ");
      System.out.print(X.Y.y+" ");
      System.out.print(X.Y.Z.z);
   }
}
```

A. 100 101 102
B. 100 100 100
C. 100 0 0
D. Compile time error

Q11

Which of the following is true about anonymous inner classes?

A. You can create ‘n’ number of objects to anonymous inner classes.

B. Anonymous inner classes will not have the name.

C. You can instantiate anonymous inner classes only once.

D. Both B and C

Q12(Output??)

CODERINDEED

```
class A
{
   static String s = "A";
   class B
   {
      String s = "B";
      void methodB()
      {
         System.out.print(s);
      }
   }
}
public class Main
{
   public static void main(String[] args)
   {
      A a = new A();
      System.out.print(a.s+" ");
      A.B b = a.new B();
      System.out.print(b.s+" ");
      b.methodB();
   }
}
```

A. A B A
B. A A B
C. A B B
D. B A A

Q13(Output??)


```
class A
{
   void methodOne()
   {
      class B
      {
         void methodTwo()
         {
            System.out.println("Method Two");
         }
      }
   }
   void methodThree()
   {
      new B().methodTwo();
   }
}
public class Main
{
   public static void main(String[] args)
   {
      A a = new A();
      a.methodThree();
   }
}
```

A. Method Two
B. Compile time error
C. Blank Output
D. Runtime error

Q14(Output??)

```
class X
{
   void methodX()
   {
      class Y
      {
         static void methodY()
         {
           System.out.println("Y");
         }
      }
      Y.methodY();
   }
}
public class Main
{
   public static void main(String[] args)
   {
     new X().methodX();
   }
}
```

A. Y
B. Compile time error
C. Runtime error
D. Blank Output

Which of the following members of outer class are accessible inside a member inner class?[Q15]

A. static members only
B. Non-static members only
C. Both A and B
D. None of these

Which of the following types of variables are not allowed to be declared inside member inner classes?[Q16]

A. Non-final static variables

B. Final static variables

C. Non-final non static variables

D. Final non static variables

Q17

How to create object of the inner class?

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

b) OuterClass.InnerClass innerObject = new InnerClass();

c) InnerClass innerObject = outerObject.new InnerClass();

d) OuterClass.InnerClass = outerObject.new InnerClass();

Q18

Non-static nested classes have access to _____ from enclosing class.

a) Private members

b) Protected members

c) Public members

d) All the members

Q19

All member functions of a local class must be _____

a) Defined outside class body

b) Defined outside the function definition

c) Defined inside the class body

d) Defined at starting of program

Q20

Which statement will instantiates an instance of the nested class?

```
public class MyOuter
{
   public static class MyInner
   {
      public static void display() { }
   }
}
```

A. MyOuter.MyInner m = new MyOuter.MyInner();

B. MyOuter.MyInner mi = new MyInner();

C. MyOuter m = new MyOuter();
MyOuter.MyInner mi = m.new MyOuter.MyInner();

D. MyInner mi = new MyOuter.MyInner();

Q21

Which statement is true about a static nested class?

A. You must have a reference to an instance of the enclosing class in order to instantiate it.

B. It does not have direct access to non-static members of the enclosing class.

C. It's variables and methods must be static.

D. It must extend the enclosing class.