## Nested Class:

#. A class defined within another class or interface is known as Nested class. The scope of the nested class is bounded by the scope of its enclosing class.

#. Static Nested class and Non-static class both treated as a data member of that class in which they are defined.

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

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

**1. Non-static nested class (inner class)**

1. Member Inner Class:
2. Local Inner Class:
2. Anonymous Inner Class

**2. Static nested class**

1. Static Nested Class:
2. Local Inner Class:
3. Anonymous Inner Class

## Static Nested Class

A static class created within class at class level. Static nested class can access all the static data members and member functions of a outer class without any inheritance even if that is private.

Static nested class cannot access non-static (instance) data member or method.

**IQ:** Can we keep main function in a static nested class.

Yes. Static nested class can be made executable by keeping a main function in it.

If we can make function as static in any class then main() can also keep in that class and it will also be executable.

If we have the static member inside static nested class, we don't need to create instance of static nested class. But if we have non-static function inside static class then it requires to make instance of that static class.

## Non-Static Nested Member Class:

A class created within a class at class level is called member inner class.

#. Non-static nested class can access all the data members and Member functions of its outer class without any inheritance.

#. Non- static Nested class can also do data shadowing. If both outer class and inner class have same name variable.

**IQ:** Can we keep main() function in a non-static class.

NO. Inner Non-static class cannot have it's own static data member and static member function.

#. If Inner Non-static data member is private then it can't go outside the outer 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.

#. 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.

## Static And Non-Static Local Class :

#. If a class is made inside block except class block then it is called local class. Local class applies rules that local variables applies.

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

#. If we make a local class within a static method of a outer class then it can use only the static things of a outer class.

# If we make local class within a non-static function of a outer class then it can access all the things of it's outer class.

#. Local inner class cannot be invoked from outside the method.

## Anonymous Class:

#. Anonymous class can be defined and instantiated simontaniously and it can not instantiated more than once.

#. Anonymous class does not have any name.

#. We can not create constructor in anonymous class explicitly but compiler adds implicitly.

#. A anonymous class must be having a parent either an interface or a class.

#. Java Anonymous inner class can be created by two ways:

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

#. While creating anonymous class, why there is () after "new My()" class name or interface. Because new operator demands constructor syntex to create memory in heap area.

#. We can also hold Reference ID of anonymous class.

# We can't make object of an interface. Interviewer trick like this.

```
My m = new My(){};

interface My{}
```

Don't consider it is making object of interface. That is syntex of anonymous class, where **My m** is holding refrence if of annonymous calss.

## Inheritance In Nested Class:

#. Nested class can also inherit other class and interfaces explicitly.

#. If a class inherits a class then child class Reference id can access everything of parent class including nested class. We don't need to make object of outer class.

#. Whenever we create object of inner class, it also send outer class Reference ID via constructor implicitly. That's why we are able to access class level non-static things inside non-static nested class.

Inside non-static Nested class "**this**" has its own class Reference Id. To access class level non-static things we can write syntex as "**OuterClassName.this.dataMemberName"** or functionName.

**Till JDK 7 local class variable were required to define as final.**

**Mine:**

#. Consider Nested class as Data members, every rule that a data member applied can applied on nested class.

#. Inner class also creates separate .class file. "OuterClassNam$InnerclassName.class".

#. Via Inheritance, we can inherit all nested class in child class because java simply treated nested class as Data members.

**IQ:** Can we declare nested class inside abstract class?

## Anonymous Nested Class Example Programs

**Way 1:**

```
interface My{
	void show();
}
class OuterClass{
	int x =100;
	static int y = 200;
// returning reference type data that's why "My" is return type
	My display(){
	// Anonymous Inner Class:  Class does have any name that's why to create memory giving iterface name with constructor systex
```

```
            return(
                new My(){
                    public void show(){
                        System.out.println("X is: " + x);
                        System.out.println("Y is: " + y);
                    }
                }
            );
        }
        public static void main(String...a){
            OuterClass o = new OuterClass();
            My m = o.display();
            m.show();
            // Can also do
            new OuterClass().display().show();
        }
}
```

**Way 2:** Making class and sending reference id. but that class will garbage after using

```
// Having parent is required
interface My{
    void show();
}
// Implementing and overriding the interface
class Temp implements My{
```

```
	public void show(){
		System.out.println("From Temp Class Show");
	}
}

class OuterClass{
	int x = 100;
	static int y = 200;
	// m is holding reference id of annonymous class
	void display(My m){
		m.show();
	}
	public static void main(String...a){
		// 1st Way
		OuterClass oc = new OuterClass();
		oc.display(new Temp());
		// Another Way:
		OuterClass oc1 = new OuterClass();
		oc1.display(
			new My(){
				public void show(){
					System.out.println("From Anonymous Class Show");
				}
			}
```

```
                );
                // Also can do
                new OuterClass().display(
                        new My(){
                                public void show(){
                                        System.out.println("From Anonymous Class Show");
                                }
                        }
                );
        }
}
```

**Way 3:** We can hold reference id of anonymous class

```
// Having parent is required
interface My{
        void show();
}
class OuterClass{
        public static void main(String...a){
                My m = new My(){
                        public void show(){
                                System.out.println("From Anonymous Class Show");
                        }
                };
```

```
            m.show();
        }
}
```

#. We can have local nested class in any block including init, static, if, for, and etc except class block.

```
class IfClass{
    void display(){
        if(1==1){
            class Inner{
                void show(){
                    System.out.println("I am Show");
                    for(int i =0; i<5; i++){
                        class InnerFor{
                            void show(){
                                System.out.println("Show From Loop");
                                }
                        }
                        new InnerFor().show();
                    }
                }
            }
            new Inner().show();
        }
```

```
    }
    final public static void main(String...a){
        new IfClass().display();
    }
}
```