Inner Classes in Java

It was introduced in Java 1.1 version to fix GUI bugs as a part of event handling. Due to benefits and powerful features programmers started using this in regular coding also.

When should we go for Inner Classes in Java?

Without existing one type of object if there is no chance of existing another type of object then we should go for Inner Classes.

Examples

1. Without existing University there is no chance of existing department object. Hence we have to declare department class inside university class.

```
class University{
      // University is outer class
      class Department
      {
            // Department is inner class
      }
}
```

2. Without existing Car there is no chance of existing engine.
3. Map is a group of key- value pair and each key- value pair is called an Entry. Without existing of Map object there is no chance of existing Entry object hence Entry Interface is defined inside Map Interface.

```
Interface Map{
      // Map is outer Interface
      Interface Entry
      {
            // Entry is inner Interface
      }
}
```

Note

- Without existing of outer class object there is no chance of existing Inner class object.
- The relation between outer and inner class is not a IS-A relationship, it is HAS-A relationship. (Composition or Aggregation). University has a Department.

Types of Inner Classes

Based on position of declaration and behaviour all inner classes are divided into four types.

- Normal or Regular Inner Class
- Method Local Inner Class
- Anonymous Inner Class
- Static **Nested** Class
  Every class is named here as inner but fourth one is named as nested. There is a reason behind it.

Normal or Regular Class

If we are declaring any named class (Not Anonymous) directly inside a class (Not Method Local Inner Class-means not declare inside a method) without a static modifier (Not Static Nested class) such type of inner class is called normal/regular inner class.

```java
public class Outer {
	class Inner{
		
	}
	
	public static void main(String[] args) {
		System.out.println("Outer Class main method invoked");
	}

}
```

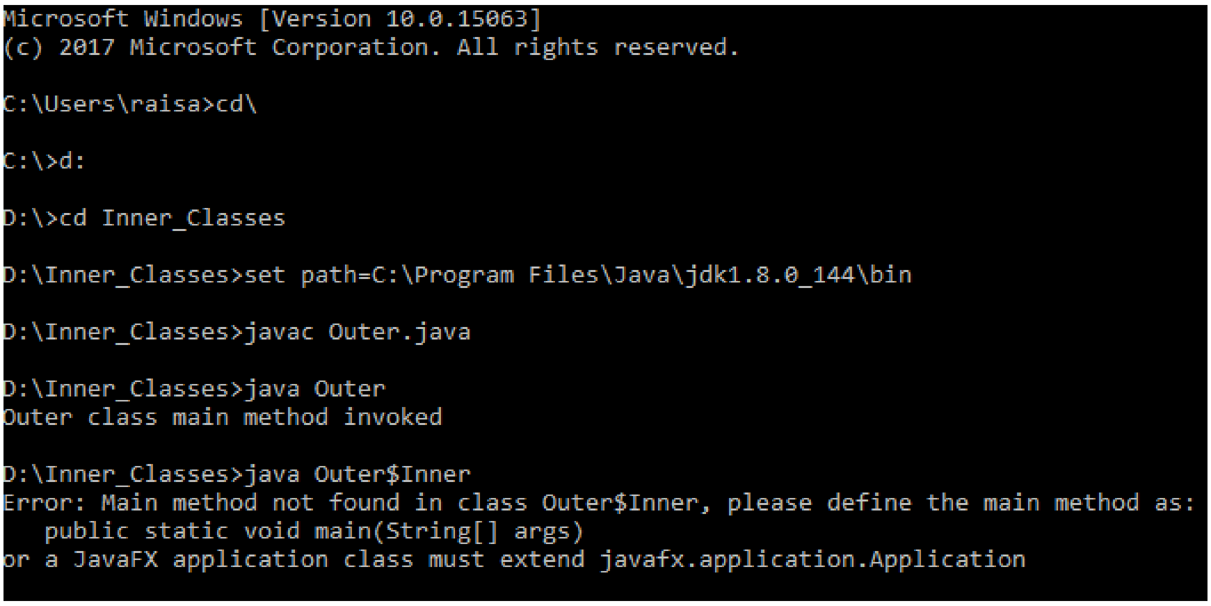
Save this class as Outer.java

Here on compiling two .class file will be generated.

.class for outer class – Outer.class

.class for inner class – Outer$Inner.class ($ symbol for inner class)

Running using command prompt-



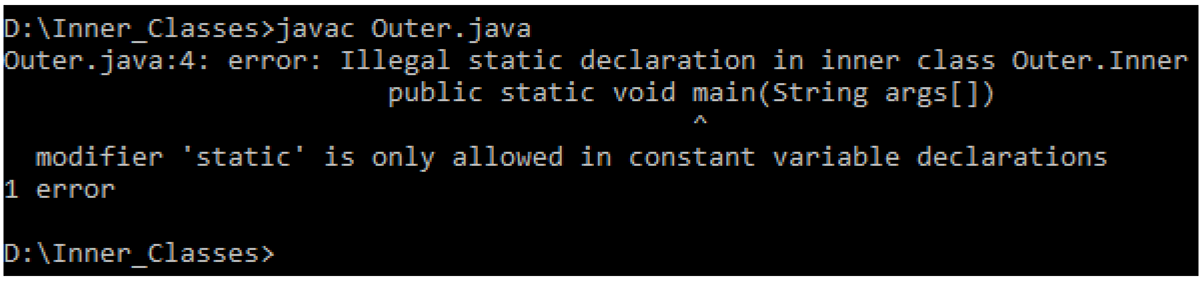
```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\raisa>cd\

C:\>d:

D:\>cd Inner_Classes

D:\Inner_Classes>set path=C:\Program Files\Java\jdk1.8.0_144\bin

D:\Inner_Classes>javac Outer.java

D:\Inner_Classes>java Outer
Outer class main method invoked

D:\Inner_Classes>java Outer$Inner
Error: Main method not found in class Outer$Inner, please define the main method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

Note : Inside inner class we cannot declare any static members hence main method cannot be declared and we cannot run inner class directly from command prompt. Inner class always talks about instance terminologies.

Now define the same main method in inner class and try to compile

```java
public class Outer {
	class Inner{
		public static void main(String[] args) {
			System.out.println("Inner Class main method invoked");
			
	}
	}
}
```

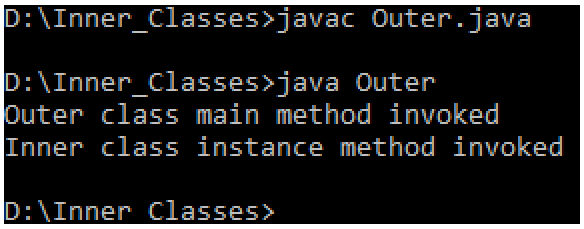
The code will not compile-



```
D:\Inner_Classes>javac Outer.java
Outer.java:4: error: Illegal static declaration in inner class Outer.Inner
                        public static void main(String args[])
                                           ^
  modifier 'static' is only allowed in constant variable declarations
1 error

D:\Inner_Classes>
```

Case-1 : Accessing Inner class code from static area of outer class

```
class Outer{
	class Inner
	{
		public void innerClassMethod()
		{
			System.out.println("Inner class instance method invoked");
		}
	}
	public static void main(String args[])
	{
		System.out.println("Outer class main method invoked");
		Outer outer=new Outer();//create outer object
		Outer.Inner oi=outer.new Inner();//Inner cannot exist without Outer
//so inner object is created using outer class object.
		oi.innerClassMethod();

	}
}
```



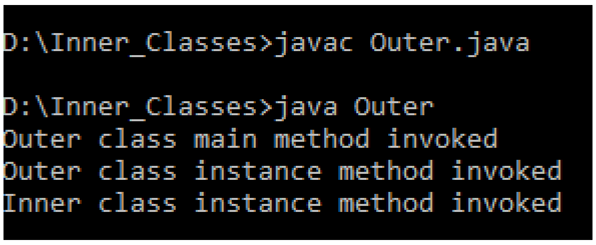
```
D:\Inner_Classes>javac Outer.java

D:\Inner_Classes>java Outer
Outer class main method invoked
Inner class instance method invoked

D:\Inner_Classes>
```

Case-02 : Accessing Inner class code from instance area of Outer class

```
class Outer{
	class Inner
	{
		public void innerClassMethod()
		{
			System.out.println("Inner class instance method invoked");
		}
	}
	public static void main(String args[])
	{
		System.out.println("Outer class main method invoked");
		Outer outer=new Outer();
		outer.outerClassInstanceMethod();

	}
```

```
        public void outerClassInstanceMethod()
        {
                System.out.println("Outer class instance method invoked");
                Inner inner=new Inner();
                inner.innerClassMethod();
        }
}
```

```
Here one question may arise inner class object is created without outer
class.Since outer object is already created in main method from where
outerClassInstanceMethod is called outer object is not required.
```

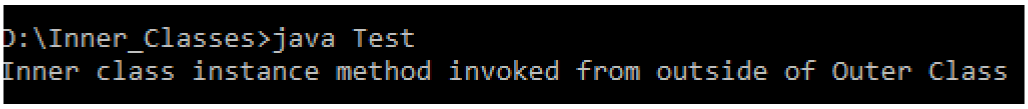


```
D:\Inner_Classes>javac Outer.java

D:\Inner_Classes>java Outer
Outer class main method invoked
Outer class instance method invoked
Inner class instance method invoked
```

Case -03 : Accessing Inner class code from outside of outer class

It is same like case – 01

```
class Outer{

        class Inner
        {
                public void innerClassMethod()
                {
                        System.out.println("Inner class instance method invoked from
outside of Outer Class");
                }
        }

}

class Test {
        public static void main(String args[])
        {
                Outer outer=new Outer();
                Outer.Inner i=outer.new Inner();
                i.innerClassMethod();


        }
}
```



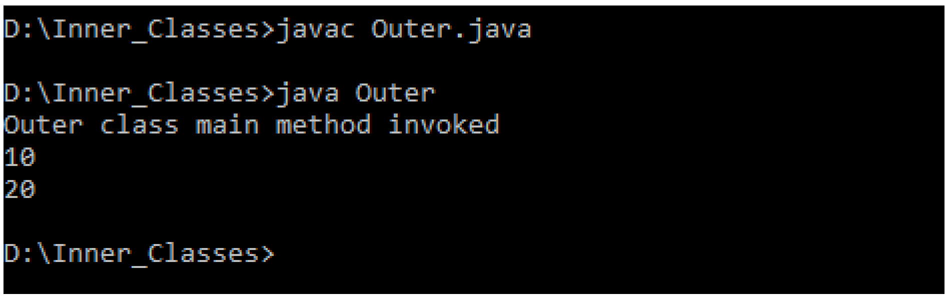
```
D:\Inner_Classes>java Test
Inner class instance method invoked from outside of Outer Class
```

From normal or Regular inner class we can access both static or non -static members directly.

Please note we can access all members of outer class in normal inner classes irrespective of static or access modifiers.

Accessing static members

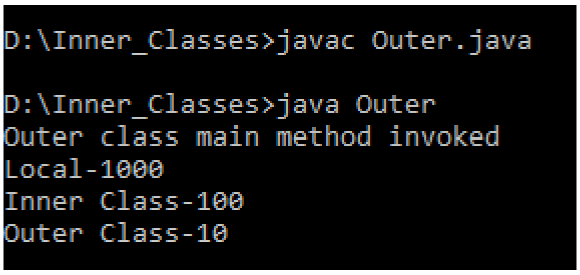
```
class Outer{
        int x=10;
        static int y=20;
        class Inner
        {
                public void innerClassMethod()
                {
                        System.out.println(x);
                        System.out.println(y);
                }
        }
        public static void main(String args[])
        {
                System.out.println("Outer class main method invoked");
                new Outer().new Inner().innerClassMethod();


        }


}
```



```
D:\Inner_Classes>javac Outer.java

D:\Inner_Classes>java Outer
Outer class main method invoked
10
20

D:\Inner_Classes>
```

Within the inner class this represents current inner class object. If we want current outer class object then we have to use outer class name.this

```
class Outer{
        int x=10;

        class Inner
        {
                int x=100;
                public void innerClassMethod()
                {
                        int x=1000;
                        System.out.println("Local-"+x);//prints current value 1000
                        System.out.println("Inner Class-"+this.x);//current instance
100
                        // we can use Inner.this.x also
                        System.out.println("Outer Class-"+Outer.this.x);//prints outer
instance 10

                }
        }
        public static void main(String args[])
        {
                System.out.println("Outer class main method invoked");
                new Outer().new Inner().innerClassMethod();


        }
}
```



```
D:\Inner_Classes>javac Outer.java

D:\Inner_Classes>java Outer
Outer class main method invoked
Local-1000
Inner Class-100
Outer Class-10
```

Applicable modifiers for Outer and Inner classes

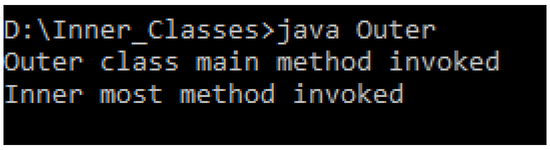
Outer – Public, Default, Final, Abstract, Strictfp

Inner – Public, Default, Final, Abstract, Strictfp, Private, Protected and Static

Nesting of Inner Classes

Inside inner class we can declare another inner class that is nesting of inner classes.

```
class Outer{
	int x=10;

	class Inner
	{

		class InnerMost
		{
			public void innerMostMethod()
			{
				System.out.println("Inner most method invoked");
			}
		}
	}
	public static void main(String args[])
	{
		System.out.println("Outer class main method invoked");
		new Outer().new Inner().new InnerMost().innerMostMethod();


	}


}
```



```
D:\Inner_Classes>java Outer
Outer class main method invoked
Inner most method invoked
```

Method Local Inner Class

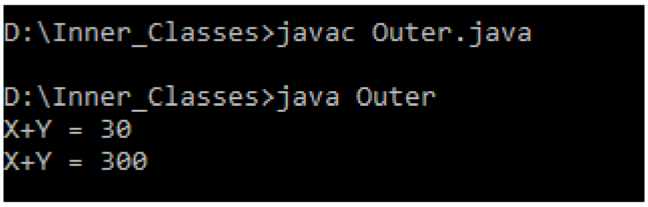
Sometime we declare a class inside a method such type of inner classes are called method local inner class.

Why we go for it?

To define method specific repeated functionality. The main purpose of it is to define method specific repeatedly required functionality. Method local inner classes are best suitable to meet nested method requirement.

We can access method local inner classes only within the method where we declare. Outside of the method we cannot access because of its less scope, it is most rarely used inner classes.

```
class Outer{

                    public void m1()
                    {
                            class MethodLocal
                            {
                                    public void sum(int x, int y)
                                    {
                                            System.out.println("X+Y = "+(x+y));
                                    }
                            }
                            MethodLocal ml=new MethodLocal();
                            ml.sum(10, 20);
                            ml.sum(100, 200);

                    }
                    public static void main(String args[])
                    {
                            Outer o=new Outer();
                            o.m1();
                    }


        }
```



```
D:\Inner_Classes>javac Outer.java

D:\Inner_Classes>java Outer
X+Y = 30
X+Y = 300
```

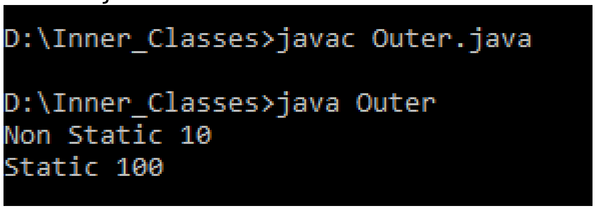
We can declare method local inner class inside both static and instance method. If it is declare inside a static method then there is a small difference.

If we declare inner class inside instance method then from that method local inner class we can static and non-static members of outer class directly.

Example-

```
class Outer{
        int x=10;
        static int y=100;

                    public void m1()
                    {
                            class MethodLocal
                            {
                                    public void m2()
                                    {
                                            System.out.println("Non Static "+x);
```

```
                        System.out.println("Static "+y);
                    }
                }
                MethodLocal ml=new MethodLocal();
                ml.m2();


            }
            public static void main(String args[])
            {
                Outer o=new Outer();
                o.m1();
            }


    }
```

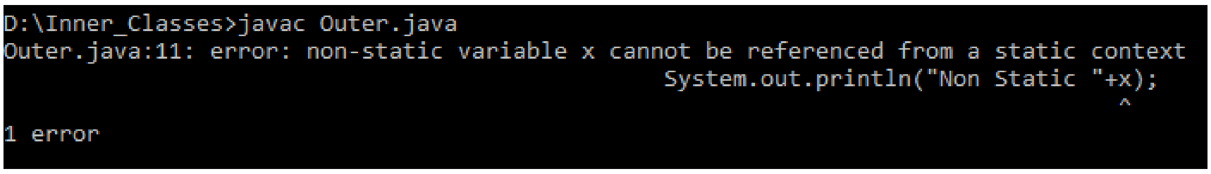


```
D:\Inner_Classes>javac Outer.java

D:\Inner_Classes>java Outer
Non Static 10
Static 100
```

If we declare inner class inside a static method then we can access only static members of outer class from that method local inner class.

Example-

```
int x=10;
    static int y=100;

            public static void m1()// complete m1() is static area now so
x cannot be accessed
            {
                class MethodLocal
                {
                    public void m2()
                    {
                        System.out.println("Non Static "+x);
                        System.out.println("Static "+y);
                    }
                }
                MethodLocal ml=new MethodLocal();
                ml.m2();


            }
            public static void main(String args[])
            {
                Outer o=new Outer();
                o.m1();
            }


    }
```



```
D:\Inner_Classes>javac Outer.java
Outer.java:11: error: non-static variable x cannot be referenced from a static context
                                                System.out.println("Non Static "+x);
                                                                                 ^
1 error
```

Note : From method local inner class we cannot access local variables of the method in which we declared inner class. If the local is variable is declared as final then we can access.

Anonymous Inner Class

Sometimes we can declare inner class without name such type of inner classes are called Anonymous Inner Classes.

The main purpose of anonymous inner classes is just for instant use (One time usage). There are two anonymous characters in Java – Class and Array.

Types of Anonymous Inner Class – Based on declaration and behaviour there are three types

- That extends a Class
- That implements an Interface
- That is defined inside arguments

❖ Anonymous Inner Class that extends a class

Consider a scenario having a super class containing 100 of methods out of which some methods are required only one time (instant use) then we can use concept of anonymous class. If it is required many times then we should go for top level class (extend to super class).

Example-

```
class Insurance {
	public void getPremium()
	{
		System.out.println("Rate is 230");
	}

}
public class AnonymousDemo {

	public static void main(String[] args) {
		Insurance i1=new Insurance();
		Insurance ins=new Insurance()
				{
				public void getPremium()
				{
					System.out.println("Premium is 349");

				}
				};
				ins.getPremium();

				Insurance ins1=new Insurance()
				{
				public void getPremium()
				{
					System.out.println("Premium is 378");

				}
```

```
                                        };
                                        ins1.getPremium();
                                        System.out.println(i1.getClass().getName());
                                        System.out.println(ins.getClass().getName());
                                        System.out.println(ins1.getClass().getName());

        }

}
```

Output-

```
Premium is 349
Premium is 378
Insurance
AnonymousDemo$1
AnonymousDemo$2
```

In above example Insurance ins=new Insurance()

{ };

We are creating subclass of Insurance without name. Here is new Insurance() is object of sub class type. Super class reference ins holding object of sub class.

While compiling three .class file will be generated. Insurance.class , AnonymousDemo.class. Since anonymous class is created within AnonymousDemo class. So its name will be AnonymousDemo$1.class. Here 1 means first anonymous class withing AnonymousDemo class.

Defining a thread by extending thread class

```
public class ThreadDemo {

        public static void main(String[] args) {
                Thread t1=new Thread()
                                {
                        @Override
                        public void run()
                        {
                                System.out.println("Running");
                        }

                                };
                                t1.start();

        }

}
```

Defining a thread by implementing Runnable interface

```
Runnable r=new Runnable() {

                        //Here we creating a class which implements Runnable interface
                };
```

❖ Anonymous Inner Class that implements an Interface

```
public class ThreadDemo {

	public static void main(String[] args) {
		Runnable r=new Runnable() {

			@Override
			public void run() {
				System.out.println("hello");


			}
		};

		Thread t=new Thread(r);
		t.start();
		System.out.println("Anonymous Class Name implementing runnable 
interface : "+r.getClass().getName());
	}}
```

Output

```
Anonymous Class Name implementing runnable interface : ThreadDemo$1 (Inner class 
naming )
```

❖ Anonymous Inner Class that defined in arguments
If we observe in above example- Thread t=new Thread(r); Here r is Runnable interface implementing class object passed as argument.

```
public class ThreadDemo {

	public static void main(String[] args) {

		 new Thread(new Runnable() {

			@Override
			public void run() {
				System.out.println("Anonymous Inner Class that defined 
in arguments");

			}
		}).start();
	}}
```

xcvb