Java means DURGA SOFT..

CORE JAVA

Material

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

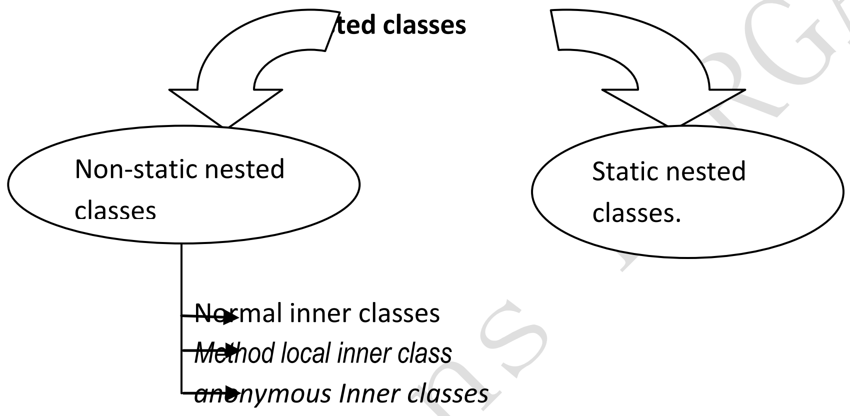
Nested classes

- Declaring the class inside another class is called nested classes. This concept is introduced in the 1.1 version.
- Declaring the methods inside another method is called inner methods java not supporting inner methods concept.

The nested classes are divided into two categories

1. **Static nested classes(nested class declared with static modifier)**
2. **Non static nested classes(these are called inner classes**)
   a. **Normal inner classes**
   b. **Method local inner classes**
   c. **Anonymous inner classes**

Static nested classes:- The nested classes declare as a static modifier is called static nested classes.



<u>syntax of nested classes :-</u>

```
classOuterclasses
{           //static nested class
            static class staticnestedclass
            {           };
            //non-static nested class
            classInnerclass
            {           };
};
```

<u>Uses of nested classes:-</u>

1. **It is the way logically grouping classes that are only used in the one place.**
   If a class is useful to other class only one time then it is logically embedded it into that classes make the two classes together.

<u>A is only one time usage in the B class (without using inner classes)</u>

```
class A
{           };
class B
{           A a=new A();
};
```

<u>by using inner classes</u>

```
class B
{           class A
            {           };
};
```

2. **It increase the encapsulation**
   If we are taking two top level classes A and B the B class need the members of A that members even we are declaring private modifier the B class can access the private numbers moreover the B is not visible for outside the world.
3. **It lead the more readability and maintainability of the code**
   Nesting the classes within the top level classes at that situation placing the code is very closer to the top level class.

For the outer classes the compiler will provide the .class and for the inner classes also the compiler will provide the .class file.

The .class file name for the inner classes is **OuterclassName$innerclasssname.class**

Outer class object creation	:-	**Outer o=new Outer();**
Inner class object creation	:-	**Outer.Inner i=o.new Inner();**
Outer class name	:-	**Outer.class**
Inner class name	:-	**Outer$Inner.class**

Member inner classes:-

1. If we are declaring any data in outer class then it is automatically available to inner classes.
2. If we are declaring any data in inner class then that data is should not have the scope of the outer class.

Syntax:-

```
class Outer
{    class Inner
     {
     };
};
```

Object creation syntax:-

Syntax 1:-

```
OuterClassName o=new OuterClassName();
OuterClassName.InnerClassNameoi=OuterObjectreference.newInnterClassName();
```

Syntax 2:-

```
OuterclassName.InnerClassNameoi=new OuterClass().new InnerClass();
```

Note:- by using outer class name it is possible to call only outer class peroperties and methods and by using inner class object we are able to call only inner classes properties and methods.

Example :-

```
class Outer
{	privateint a=100;
	class Inner
	{	void data()
		{	System.out.println("the value is :"+a);		}
	}
}
class Test
{	public static void main(String[] args)
	{	Outer o=new Outer();
		Outer.Inner i=o.newInner();
		i.data();
	}
};
```

Example :-

```
class Outer
{	int i=100;
	void m1()
	{	//j=j+10;// compilation error
		//System.out.println(j);//compilation error
		System.out.println("m1 method");
	}
	class Inner
	{	int j=200;
		void  m2()
		{	i=i+10;
			System.out.println(i);
		}
	};
};
```

```
class Test
{          public static void main(String[] args)
           {          A a=new A();
                      System.out.println(a.i);
                      a.m1();
                      A.B b=a.newB();
                      System.out.println(b.j);
                      b.m2();
                      //b.m1();   compilation error
           }
};
```

Example :-

```
class Outer
{          privateint a=10;private int b=20;
           void m1()
           {          //m2(); not possible
                      System.out.println("outer class m1()");
           }
           class Inner
           {          int i=100; int j=200;
                      void m2()
                      {          System.out.println("inner class m1()");
                                 System.out.println(a+b);
                                 System.out.println(i+j);
                                 m1();
                      }
           };
};
class Test
{          public static void main(String... ratan)
           {          Outer o = new Outer();                          o.m1();
                      Outer.Inner i = o.newInner();           i.m2();
           }
};
```

Application required this & super keywords:-

```
class Outer
{           privateint a=10;private int b=20;
            class Inner
            {           int a=100; int b=200;
                        void m1(inta,int b)
                        {           System.out.println(a+b);//local variables
                                    System.out.println(this.a+this.b);//Inner class variables
                                    System.out.println(Outer.this.a+Outer.this.b);//outer class variables
                        }
            };
};
class Test
{           public static void main(String... ratan)
            {           Outer.Inner i = new Outer().new Inner();
                        i.m1(1000,2000);
            }
};
```

```
class Outer
{           void m1(){          System.out.println("outer class m1()");   }
            class Inner
            {           void m1()
                        {   Outer.this.m1();
                                    System.out.println("inner class m1()");
                        }
            };
};
class Test
{           public static void main(String... ratan)
            {           Outer.Inner i = new Outer().new Inner();
                        i.m1();
            }
};
```

Method local inner classes:-

1. Declaring the class inside the method is called method local inner classes.
2. In the case of the method local inner classes the class has the scope up to the respective method.
3. Method local inner classes do not have the scope of the outside of the respective method.
4. whenever the method is completed
5. we are able to perform any operations of method local inner class only inside the respective method.

Syntax:-

```
class Outer
{	void m1()
	{	class inner
		{			};
	}
};
```

Example:-

```
class Outer
{	privateint a=100;
	void m1()
	{	class Inner
		{
		voidinnerMethod()
		{	System.out.println("inner class method");
			System.out.println(a);
		}
	};
	Inner i=new Inner();
	i.innerMethod();
	}
};
class Test
{	public static void main(String[] args)
	{	Outer o=new Outer();
		o.m1();
	}
};
class Outer
{	void m1()
	{	class Inner
		{	void m1(){System.out.println("inner class m1()");}
		};
		Inner i =  new Inner();
		i.m1();
	}
	public static void main(String[] args)
	{	Outer o = new Outer();
		o.m1();
```

```
        }
};
class Outer
{       privateint a=100;
        void m1()
        {       finalint b=200;//local variables must be final variables
                class Inner
                {       void m1()
                        {       System.out.println("inner class m1()");
                                System.out.println(a);
                                System.out.println(b);
                        }
                };
                Inner i =  new Inner();
                i.m1();
        }
        public static void main(String[] args)
        {       Outer o = new Outer();
                o.m1();
        }
};
class Outer
{       int a=10;//instance varaible
        staticint b=20; //static variable
        class Inner   //inner class able to access both instance and static variables
        {       void m1()
                {       System.out.println(a);
                        System.out.println(b);
                }
        };
};
class Outer
{       staticint a=10;//static variable
        int b=20;               //instance variable
        static class Inner //this inner class able to access only static memebers of outer class
        {               void m1(){
                        System.out.println(a);
                        System.out.println(b);//compilation error
                        }
        };
};
```

Ex 2:-in method local inner classes it is not possible to call the non-final variables inside the inner classes hence we must declare that local variables must be final then only it is possible to access that members.

```
class Outer
{       privateint a=100;
        void m1()
```

```
          {finalint b=1000;
          class Inner
          {          voidinnerMethod()
                     {          System.out.println("inner class method");
                                System.out.println(a);
                                System.out.println(b);
                     }
          };
          Inner i=new Inner();
          i.innerMethod();
          }
};
class Test
{         public static void main(String[] args)
          {          Outer o=new Outer();
                     o.m1();
          }
};
```

Static inner classes:-
In general in java classes it is not possible to declare any class as aabstract class but is possible to declare inner class as a static modifier.
Declaring the static class inside the another class is called static inner class.
Static inner classes can access only static variables and static methods it does not access the instace variables and instance methods.

Syntax:-
```
          class Outer
          {          static class Inner
                     {
                     };
          };
```

```
class Outer
{         staticint a=10;
```

```
        staticint b=20;
        static class Inner
        {       int c=30;
                void  m1()
                {       System.out.println(a);
                        System.out.println(b);
                        System.out.println(c);
                }
        };
        public static void main(String[] args)
        {       Outer o=new Outer();
                Outer.Inner i=new Outer.Inner();
                i.m1();
        }
};
class Outer
{       staticint a=10;//static variable
        staticint b=20;//static variable
        static class Inner //this inner class able to access only static memebers of outer class
        {               void m1(){
                        System.out.println(a);
                        System.out.println(b);
                        }
        };
        public static void main(String[] args)
        {       Outer.Inner i = new Outer.Inner();//it creates object of static inner class
                i.m1();
        }
};
class Outer
{       staticint a=10;//static variable
        staticint b=20;//static variable
        static class Inner //this inner class able to access only static memebers of outer class
        {               void m1(){
                        System.out.println(a);
                        System.out.println(b);
                        }
        };
        public static void main(String[] args)
        {       Outer.Inner i = new Outer.Inner();//it creates object of static inner class
                i.m1();
        }
};
```

Anonymous inner class:-

1. The name less inner class is called anonymous inner class.
2. it can be used to provide the implementation of normal class   or abstract class or interface

Anonymous inner classes for abstract classes:-

it is possible to provide abstract method implementations by taking inner classes.

Ex:- we are able to declare anonymousinner class inside the class.

```
abstract class Animal
{       abstract void eat();
};
class Test
{               //anonymous inner class
                Animal a=new Animal()
                {       void eat()      {       System.out.println("animals eating gross");     }
                };
        public static void main(String[] args)
        {       Test t=new Test();
                t.a.eat();
        }
}
```

Ex:- we are able to declare anonymous inner class inside the main method.

```
abstract class Animal
{       abstract void eat();
};
class Test
{       public static void main(String[] args)
        {
                Animal a=new Animal()
                {       void eat()
                        {               System.out.println("animals eating gross");     }
                };
                a.eat();
        }
}
```

Note :- In above example we are taking animal class having eat() method and we are overriding method but this thing can done by creating subclasses of existing class by using extends keyword then what is the need of anonymous inner classes.

The answer is creating anonymous inner class simple. And whenever we are inherit few properties (only method) of superclass instead of extending class use anonymous inner class.

```
 //interface (contains abstract methods)
interface it
{
            void m1();
            void m2();
            ;;;;;;;;;;;;
            void m100();
}
//adaptor class(contains empty implementation of interface methods)
class X implements it
{
            void m1(){}
            void m2(){}
            ;;;;;;;;
            void m100(){}
};
//userdefined class extending adaptor class
class Test extends X
{
            //all methods are visible here
};
//useing anonymous inner class (override required method)
class Test
{
            //anonymous inner class
            X x = new X()
            {
            //override required methods(requied methods are loaded)
            void m1(){System.out.println("anonymous inner class");}
            };//semicolan mandatory
};
interface It1 //interface
{
            void m1();  //by default interface methods are puliv abstract
            void m2();
            ;;;;;;;;;;;
            void m100();
}
class X implements It1//adaptor class
{ //it is adaptor class contains empty implementation of all interface methods
            public void m1(){} //implementation method must be public
            public void m2(){}
            ;;;;;;;;;;;
            public void m100(){}
```

```
};
abstract class Test implements It1
{
	//must provide the implementation of 100 methods
};
//approach-1 it is possible to extends the class and override required method
class Test1 extends X
{
	//override the required methods
	public void m1(){System.out.println("m1 method");}
};
//approach-2 without extending class it is possible to create the object directly and override required
method
classRatan
{
	X x= new X()//this is anonymous inner class
	{
		public void m1(){System.out.println("anonymous inner class");}  }; //semicoln
mandatory
	public static void main(String[] args)
	{		Ratan r = new Ratan();
		r.x.m1();
	}
};
//predefined class contains 2-methods
class A
{
	void m1(){System.out.println("A m1 method ");}
	void m2(){System.out.println("A m2 method ");}
};
//approach-1 extends the then override required methods
class Test extends A
{
	void m1(){System.out.println("Test extends A --> m1 method ");}
	public static void main(String[] args)
	{
		Test t = new Test();  t.m1();
	}
};
//approach-2 don't extends the class declare anonymous inner class then override the required methods
classRatan
{
	A a = new A()
	{
	void m1(){System.out.println("Anonumous inner class m1 method ");}
	};//semicolan mandatory
```

```
        public static void main(String[] args)
        {
                Ratan r = new Ratan();
                r.a.m1();
        }
};

//predefined class contains 2-methods
class A
{
        void m1(){System.out.println("A m1 method ");}
        void m2(){System.out.println("A m2 method ");}
};
//approach-1 extends the then override required methods
class Test extends A
{
        void m1(){System.out.println("Test extends A --> m1 method ");}
        public static void main(String[] args)
        {
                Test t = new Test();  t.m1();
        }
};
//approach-2 don't extends the class declare anonymous inner class then override the required methods
classRatan
{
        A a = new A()
        {
        void m1(){System.out.println("Anonumous inner class m1 method ");}
        };//semicolan mandatory
        public static void main(String[] args)
        {
                Ratan r = new Ratan();
                r.a.m1();
        }
};
```

LEARN FROM EXPERTS ...

COMPLETE JAVA

CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...

COMPLETE .NET

C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF

TESTING TOOLS

MANUAL + SELENIUM

ORACLE **D2K**

MSBI **SHARE POINT**

HADOOP **ANDROID**

C, C++, DS, UNIX

CRT & APTITUDE TRAINING

AN ISO 9001:2008 CERTIFIED

DURGA

Software Solutions®

202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696

www.durgasoft.com