Inner Classes

Inner Classes

Declaring a class inside a class is called as Inner class.

Syntax:

```
class Outer_Class
{
   class Inner_Class
   {
   }
}
```

In Java applications, Inner classes are able to provide the following advantages.

1) Modularity
2) Abstraction
3) Security
4) Sharability
5) Reusability

1) Modularity:

- In Java application, we are able to improve we modularity by using packages.
- If we want to divide our implementation in a class in the form of modules then we have to use Inner classes.

EX:

```
class Account {
        class SavingsAccount{
              ----
        }
        class CurrentAccount {
              ----
        }
    }
```

2) Abstraction:

If we declare any variable or method in an inner class then that variable and method are having scope up to that inner class only, it is not available to other inner classes, so that, Inner classes are able to improve Abstraction.

EX:

```
class A {
```

```
class B {
     int i = 10;
}
class C {
    void m1() {
         i = i+10; --> Error
    }
}
}
```

3) Security:

It is not possible to declare a class as private, but, it is possible to declare an inner class as private, so that, inner classes are able to improve Security.

EX:
```
private class A {
}
```
Status: Invalid

EX:
```
class A {
    private class B {
        ----
    }
}
```
Status: Valid

4) Sharability:

It is not possible to declare a class as Static, but, it is possible to declare an inner class as static, so that, inner classes are able to improve Sharability.

EX:
```
static class A {
}
```
Status: Invalid

EX:
```
class A {
     static class B {
     }
}
```
Status: Valid

5) Reusability:

- In Java, inheritance feature will improve Code Reusability.
- In java applications, we can extend one inner class to another class, so that, inner classes are able to improve Code Reusability.

EX:
```
class A {
        class B {
            ---
        }
        class C extends B {
                ---
        }
    }
```
Status: Valid

There are four types of inner classes in java.

- Member Inner class
- Static Inner class
- Method Local Inner class
- Anonymous Inner class

Note: If we compile java file contains inner classes then Compiler will generate a separate .class for outer class and a separate .class file will be generated for inner class.

Outer_Class.class ---> for Outer class
Outer_Class$Inner_Class.class---> For Inner class

Member Inner Class:

Declaring a normal class [Non static class] inside a class is called as member inner class.

EX:
```
class Outer {
        class Inner {
                ----
        }
    }
```

If we want to access the members of inner class then we have to create object for the inner class, for this, we have to use the following syntax.

```
Outer.Inner ref_Var = new Outer().new Inner();
```

- By using outer class reference variable we are able to access only outer class members, we are unable to access inner class members.
- By using Inner class reference variable we are Able to access only inner class members we are unable to access outer class members.

- By default, all outer class members are available to inner class, so we can access outer class members inside the inner class, but, Inner class members are not available to outer classes, we are unable to access inner class members in outer classes.
- Member inner classes are not allowing static declarations directly, but, static keyword is allowed along with final keyword.

EX:

```
1)  class A
2)  {
3)     void m1()
4)     {
5)        System.out.println("m1-A");
6)     }
7)     class B
8)     {
9)        //static int i=10;--> Error
10)       static final int j=20;
11)       void m2()
12)       {
13)          System.out.println("m2-B");
14)          System.out.println(j);
15)       }
16)       void m3()
17)       {
18)          System.out.println("m3-B");
19)       }
20)    }
21) }
22) class Test {
23)    public static void main(String[] args)
24)    {
25)       A a=new A();
26)       a.m1();
27)       //a.m2();--> Error
28)       A.B ab=new A().new B();
29)       ab.m2();
30)       ab.m3();
31)       //ab.m1();--> Error
32)    }
33) }
```

In Member inner classes, we can extend one inner class to another inner class but both the inner classes must be provided with in the same outer class.

EX:

```
1)  class A
2)  {
3)     class B
4)     {
5)        void m1()
6)        {
7)           System.out.println("m1-B");
8)        }
9)        void m2()
10)       {
11)          System.out.println("m2-B");
12)       }
13)    }
14)    class C extends B
15)    {
16)       void m3()
17)       {
18)          System.out.println("m3-C");
19)       }
20)       void m4()
21)       {
22)          System.out.println("m4-C");
23)       }
24)    }
25) }
26) class Test
27) {
28)    public static void main(String[] args)
29)    {
30)       A.B ab=new A().new C();
31)       ab.m1();
32)       ab.m2();
33)       //ab.m3();--> Error
34)       A.C ac=new A().new C();
35)       ac.m1();
36)       ac.m2();
37)       ac.m3();
38)       ac.m4();
39)    }
40) }
```

We cannot extend one inner class to another inner class which are available in two different outer classes.

EX:

```
class A {
    class B {
       ---
    }
}
class C {
    class D extends A.B {
              ---
    }
}
```

Status: Invalid.

We can extend an inner class from an outer class.

EX:

```
class A {
    ---
}
class B {
     class C extends A {
         ----
      }
}
```

Status: Valid

We can extend the immediate outer class to its inner class.

EX:

```
class A {
     class B extends A {
          ---
     }
}
```

Q) Is it possible to write an interface inside a class?

Yes, it is possible to provide an interface inside a class, but, the respective implementation class must be provided with in the same outer class.

EX:

```
1)  class A
2)  {
3)     interface I
4)     {
5)        void m1();
6)        void m2();
7)        void m3();
8)     }
9)     class B implements I
10)    {
11)       public void m1()
12)       {
13)          System.out.println("m1-B");
14)       }
15)       public void m2()
16)       {
17)          System.out.println("m2-B");
18)       }
19)       public void m3()
20)       {
21)          System.out.println("m3-B");
22)       }
23)    }
24) }
25) class Test
26) {
27)    public static void main(String[] args)
28)    {
29)       A.I ai=new A().new B();
30)       ai.m1();
31)       ai.m2();
32)       ai.m3();
33)    }
34) }
```

EX:

```
1)  class A
2)  {
3)     abstract class B
4)     {
```

```
5)      void m1()
6)      {
7)         System.out.println("m1-B");
8)      }
9)      abstract void m2();
10)     abstract void m3();
11)  }
12)  class C extends B
13)  {
14)     void m2()
15)     {
16)        System.out.println("m2-C");
17)     }
18)     void m3()
19)     {
20)        System.out.println("m3-C");
21)     }
22)  }
23) }
24) class Test
25) {
26)   public static void main(String[] args)
27)   {
28)     A.B ab=new A().new C();
29)     ab.m1();
30)     ab.m2();
31)     ab.m3();
32)   }
33) }
```

EX:

```
1)  package com.durgasoft.core;
2)
3)  abstract class A{
4)     class B{
5)        void m1(){
6)           System.out.println("m1-B");
7)        }
8)        void m2(){
9)           System.out.println("m2-B");
10)       }
11)       void m3(){
12)          System.out.println("m3-B");
13)       }
14)   }
```

```
15) }
16) class C extends A{
17)     // class B{   }
18) }
19) class Test{
20)     public static void main(String[] args){
21)         A.B ab = new C().new B();
22)         ab.m1();
23)         ab.m2();
24)         ab.m3();
25)     }
26) }
```

EX:

```
1)   package com.durgasoft.core;
2)
3)   abstract class A{
4)       abstract class B{
5)           void m1(){
6)               System.out.println("m1-B");
7)           }
8)           abstract void m2();
9)           abstract void m3();
10)      }
11)      class C extends  B{
12)          @Override
13)          void m2() {
14)              System.out.println("m2-C");
15)          }
16)
17)          @Override
18)          void m3() {
19)              System.out.println("m3-C");
20)          }
21)      }
22) }
23) class D extends A{
24)
25) }
26) class Test{
27)     public static void main(String[] args){
28)         A.B ab = new D().new C();
29)         ab.m1();
30)         ab.m2();
31)         ab.m3();
```

```
32)    }
33) }
```

Ex:

```
1)  package com.durgasoft.core;
2)
3)  abstract class A{
4)     interface I{
5)        void m1();
6)        void m2();
7)        void m3();
8)     }
9)     class B implements I{
10)       @Override
11)       public void m1() {
12)          System.out.println("m1-B");
13)       }
14)
15)       @Override
16)       public void m2() {
17)          System.out.println("m2-B");
18)       }
19)
20)       @Override
21)       public void m3() {
22)          System.out.println("m3-B");
23)       }
24)    }
25) }
26) class C extends A{
27)
28) }
29) class Test{
30)    public static void main(String[] args){
31)       A.I ai = new C().new B();
32)       ai.m1();
33)       ai.m2();
34)       ai.m3();
35)    }
36) }
```

Static Inner Class:

Declaring a static class inside a class is called as Static inner class.

Syntax:

```
class Outer
{
   static class Inner
   {
     ----
   }
}
```

If we want to access the members of static inner class we have to create object for static inner class, for this we have to use the following syntax.

```
Outer.Inner ref_Var = new Outer.Inner();
```

- Static inner classes are able to allow only static members of the outer class, it will not allow non static members of the outer class.
- In general, inner classes are not allowing static keyword directly, but, static inner class is able to allow static declarations directly.

EX:

```
1)  class A
2)  {
3)     static class B
4)     {
5)        void m1()
6)        {
7)           System.out.println("m1-B");
8)        }
9)        void m2()
10)       {
11)          System.out.println("m2-B");
12)       }
13)       static void m3()
14)       {
15)          System.out.println("m3-B");
16)       }
17)    }
18) }
19) class Test
```

```
20) {
21)    public static void main(String[] args)
22)    {
23)       A.B ab=new A.B();
24)       ab.m1();
25)       ab.m2();
26)       A.B.m3();
27)    }
28) }
```

Q) Is it possible to write a class inside an interface?

Yes, it is possible to write a class inside an interface, If we declare a class inside an interface then that class is converted as a static inner class, we can access members of this inner class like static inner class members.

EX:

```
1)  interface I
2)  {
3)     class A// static class A
4)     {
5)        void m1()
6)        {
7)           System.out.println("m1-A");
8)        }
9)        void m2()
10)       {
11)          System.out.println("m2-A");
12)       }
13)    }
14) }
15) class Test
16) {
17)    public static void main(String[] args)
18)    {
19)       I.A ia=new I.A();
20)       ia.m1();
21)       ia.m2();
22)    }
23) }
```

Note: We can write abstract class inside an interface, but, the respective sub class must be declared in the same interface, here the declared sub class is converted as static inner class.

EX:

```
1)  interface I
2)  {
3)     abstract class A
4)     {
5)        void m1()
6)        {
7)           System.out.println("m1-A");
8)        }
9)        abstract void m2();
10)       abstract void m3();
11)    }
12)    class B extends A
13)    {
14)       void m2()
15)       {
16)          System.out.println("m2-B");
17)       }
18)       void m3()
19)       {
20)          System.out.println("m3-B");
21)       }
22)    }
23) }
24) class Test
25) {
26)    public static void main(String[] args)
27)    {
28)       I.A ia=new I.B();
29)       ia.m1();
30)       ia.m2();
31)       ia.m3();
32)    }
33) }
```

NOTE: We can write an interface inside an interface, but, the respective implementation class must be provided with in the same outer interface.

EX:

```
1)  interface I1
2)  {
3)     interface I2
4)     {
```

```
5)      void m1();
6)      void m2();
7)      void m3();
8)   }
9)   class A implements I2
10)  {
11)     public void m1()
12)     {
13)        System.out.println("m1-A");
14)     }
15)     public void m2()
16)     {
17)        System.out.println("m2-A");
18)     }
19)     public void m3()
20)     {
21)        System.out.println("m3-A");
22)     }
23)  }
24) }
25) class Test
26) {
27)   public static void main(String[] args)
28)   {
29)     I1.I2 i12=new I1.A();
30)     i12.m1();
31)     i12.m2();
32)     i12.m3();
33)   }
34) }
```

□ Method Local Inner Class:

Declaring class inside a method is called as Method Inner class.
If we declare a class inside a method then the scope of that is available up to the respective method only, we have to create object for5 that inner class in the respective method only, we have to access the members of that inner class inside the respective method.

EX:

```
1)  class A
2)  {
3)     void m1()
4)     {
5)        class B
```

```
6)      {
7)          void m2()
8)          {
9)              System.out.println("m2-B");
10)         }
11)         void m3()
12)         {
13)             System.out.println("m3-B");
14)         }
15)     }//B
16)     B b=new B();
17)     b.m2();
18)     b.m3();
19)   }//m1()
20)}//A
21)class Test
22){
23)   public static void main(String[] args)
24)   {
25)     A a=new A();
26)     a.m1();
27)   }
28)}
```

Anonymous Inner Classes:

Anonymous inner classes are nameless inner classes, these are used to provide implementation for abstract classes and interfaces.

Note: For abstract class we may take sub classes and for interfaces we may take implementation classes to provide implementations , but, here sub classes of the abstract class and implementation class of the interface are able to allow their own members and these sub classes and implementation classes objects are having their identity, not interface identity and abstract class identity.

In java applications, if we want to provide implementations for only abstract class members or interface members and if we want to create objects with interface identity and with abstract class identity then we have to use Anonymous inner classes.

Syntax:

```
abstract class Name / interface Name
{
  ----
}
```

```
class Outer {
      Name ref_Var = new Name()
      {
            ---implementation----
      };
}
```

EX:

```
1)  abstract class A
2)  {
3)     void m1()
4)     {
5)        System.out.println("m1-A");
6)     }
7)     abstract void m2();
8)     abstract void m3();
9)  }
10) class Outer
11) {
12)    A a=new A()
13)    {
14)       void m2()
15)       {
16)          System.out.println("m2-AIC");
17)       }
18)       void m3()
19)       {
20)          System.out.println("m3-AIC");
21)       }
22)       void m4()
23)       {
24)          System.out.println("m4-AIC");
25)       }
26)    };
27) }
28) class Test
29) {
30)    public static void main(String[] args)
31)    {
32)       Outer o=new Outer();
33)       o.a.m1();
34)       o.a.m2();
35)       o.a.m3();
36)       //o.a.m4();--> Error
37)    }
```

```
38) }
```

EX:

```
1)  interface I
2)  {
3)     void m1();
4)     void m2();
5)     void m3();
6)  }
7)  class Outer
8)  {
9)     I i=new I()
10)    {
11)       public void m1()
12)       {
13)          System.out.println("m1-AIC");
14)       }
15)       public void m2()
16)       {
17)          System.out.println("m2-AIC");
18)       }
19)       public void m3()
20)       {
21)          System.out.println("m3-AIC");
22)       }
23)       public void m4()
24)       {
25)          System.out.println("m4-AIC");
26)       }
27)    };
28) }
29) class Test
30) {
31)    public static void main(String[] args)
32)    {
33)       Outer o=new Outer();
34)       o.i.m1();
35)       o.i.m2();
36)       o.i.m3();
37)       //o.i.m4();--> Error
38)    }
39) }
```

In general, we will use Anonymous Inner classes when we have requirement to pass any interface or abstract class references as parameters to the methods.

EX:

```
1)  interface I
2)  {
3)     void m1();
4)     void m2();
5)     void m3();
6)  }
7)  class A
8)  {
9)     void meth(I i)
10)    {
11)       i.m1();
12)       i.m2();
13)       i.m3();
14)    }
15) }
16) class Test
17) {
18)    public static void main(String[] args)
19)    {
20)       A a=new A();
21)       a.meth(new I()
22)       {
23)          public void m1()
24)          {
25)             System.out.println("m1-AIC");
26)          }
27)          public void m2()
28)          {
29)             System.out.println("m2-AIC");
30)          }
31)          public void m3()
32)          {
33)             System.out.println("m3-AIC");
34)          }
35)          public void m4()
36)          {
37)             System.out.println("m4-AIC");
38)          }
39)       });
40)    }
41) }
```