Inner Classes

Date
Page

→ Sometimes we can declare a class inside class, such type of classes are called inner class.

→ Inner classes concept introduced in 1.1 version to fix GUI bugs as a part of Event handling but because of powerful features & benefits of inner classes slowly programmers started using in regular coding also.

→ Without existing one type of object if there is no chance of existing another type of object then we should go for inner classes.

→ Ex-1

- University consists of several departments without existing university there is no chance of existing department, hence we have to declare department class inside university class

```
class University          → outer class
{
    class Department      → inner class
    {
    }
}
```

Ex-2

→ without existing car object there is no chance of existing engine object, hence we have to declare engine class inside car class

```
class Car
{
    class Engine
    {
    }
}
```

→ Ex-3

Map is a group of key, value pairs & each key, value pair is called an Entry, without existing Map object there is no chance of existing Entry object hence Interface Entry is defined inside Map interface

```
Map                       → outer interface
interface Map
{
    interface Entry       → inner interface
    {
    }
}
```

→ Note :- Without existing outer class object there is no chance of existing inner class object.

- The relation between an outer class and inner class is not IS-A Rel^n. and it is Has-A Rel^n. (Composition / Aggregation)

→ Based on position of declaration & behavior all inner classes are devided into 4 types :-

1) Normal / Regular Inner classes
2) Method Local Inner classes.
3) Anonymous Inner classes.
4) Static Nested classes.

⇒ Normal / Regular Inner Classes

→ If we are declaring any named class directly inside a class without static modifier, such type of inner class is called Normal / Regular Inner class.

→ Ex.1 (Outer.java)

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

javac Outer.java

Outer.class

Outer$Inner.class

java Outer ↵

RE: NoSuchMethodError: main

java Outer$Inner ↵

RE: NoSuchMethodError: main

→ Ex.2 (Outer.java)

```
class Outer
{
    class Inner
    {
    }
    public static void main(String[] args)
    {
        S.O.P("outer class main method");
    }
}
```

java Outer ↵

outer class main method

java outer$Inner ↵

R.E. NoSuchMethodError: main

⇝ Inside Inner class we can't declare any static member including main method & we can't run inner class directly from command prompt

Ex-3

```
class Outer
{
    class Inner
    {
        p s v m (str)
        {
            s.o.p ("Inner class
                    main method");
        }
    }
}
```

javac Outer.java

C.E: Inner class cannot have static declarations.

⇝ case-1 :- [Accessing Inner class code from static Area of outer class]

```
class Outer
{
    class Inner
    {
        public void m1()
        {
            s.o.p ("Inner class method");
        }
    }
    p s v main (str)
    {
        Outer o = new Outer();
        Outer.Inner i = o.new Inner();
        i.m1();
    }
}
```

© Outer.Inner i = new Outer().new Inner();

O/P :-

Inner class method

→ case-2 : [Accessing Inner class code from Instance Area of outer class]

```
class Outer
{
    class Inner
    {
        public void m1()
        {
            s.o.p("Inner class method");
        }
    }
    public void m2()
    {
        Inner i = new Inner();
        i.m1();
    }
    p s v main(strn)
    {
        Outer o = new Outer();
        o.m2();
    }
}
```

o/p :

Inner class method

(this.Inner i = this.new Inner();)

→ case-3 : [Accessing Inner class code from outside of outer class]

```
class Outer
{
    class Inner
    {
        public void m1()
        {
            s.o.p("Inner class
                    method");
        }
    }
}
class Test
{
    p s v main(strn)
    {
        Outer o = new Outer();
        Outer.Inner i = o.new Inner();
        i.m1();
    }
}
```

o/p :

Inner class method.

→ From Normal/Regular Inner class we can access both static & non-static members of outerclass directly.

Ex.

```
class Outer
{
    int x=10;
    static int y=20;

    class Inner
    {
        public void m1()
        {
            s.o.p(x);
            s.o.p(y);
        }
    }

    p s v main(strn)
    {
        new Outer().new Inner().m1();
    }
}
```

O/P :- 10
20

→ Within the inner class "this" always refers current Inner class object, if we want to refer current outer class object, we have to use (OuterClassName.this)

Ex.

```
class Outer
{
    int x=10;
    class Inner
    {
        int x=100;
        public void m1()
        {
            int x=1000;
            s.o.p(x);
            s.o.p(this.x); //Inner this.x
            s.o.p(Outer.this.x);
        }
    }

    p s v main(strn)
    {
        new Outer().new Inner().
                    m1();
    }
}
```

O/P :- 1000
100
10

→ The only applicable modifiers for Outer classes are :-

1) public
2) <default>
3) final
4) abstract
5) strictfp

→ But for Inner class possible applicable modifiers are :-

1) public
2) <default>
3) final
4) abstract
5) strictfp
6) private
7) protected
8) static

→ Nesting of Inner class

- Inside Inner class we can ~~also~~ declare another Inner class i.e nesting of Inner class is possible.

Ex:

```
class A
{
    class B
    {
        class C
        {
            public void m()
            {
                S.o.p("Innermost method");
            }
        }
    }
class Test
{
    p s v main(sa-)
    {
        A a = new A();
        A.B b = a.new B();
        A.B.C c = b.new C();
        c.m();
    }
}
```

Method Local Inner classes

→ Sometimes we can declare a class inside a method. Such type of Inner classes are called Method local Inner classes.

→ The main purpose of method local Inner class is to define method specific repeatedly required functionality.

→ Method local Inner classes are best suitable to meet nested method requirements.

→ We can access method local Inner classes within a method only where we declare, we can't access it outside that method, because of less scope, this is the least used Inner class.

→ Ex:-

```
class Test
{
    public void m1()
    {
        class Inner
        {
            public void sum(int x, int y)
            {
                S.o.p("The sum: "+(x+y));
            }
        }
        Inner i = new Inner();
        i.sum(10,20);
        :
        i.sum(100,200);
    }
    public static void main(Str...)
    {
        Test t = new Test();
        t.m1();
    }
```

O/P :- The sum: 30
The sum: 300

→ We can declare Method local Inner class inside both Instance & static methods.

→ If we declare Inner class inside ~~directly~~ Instance method then from that method local Inner class we can access both static & Non-static members of outer class directly.

→ If we declare ~~inside~~ Inner class inside static method then we can access only static members of outer class directly from that method local inner class.

Ex-2

```
class Test
{
    int x=10;
    static int y=20;

    public void m1()
    {
        class Inner()
        {
            public void m2()
            {
                s.o.p(x);  ——— line (1)
                s.o.p(y);
            }
        }
        Inner i = new Inner();
        i.m2();
    }
    p s v main(sm)
    {
        Test t = new Test();
        t.m1();
    }
}
```

O/p : 10
20

⇒ If we declare m1() method as static then at line (1) we will get

[C.E. Non-static variable x can not be referenced from a static context.]

★ ⇒ From Method local Inner class, we can't access local variables of the method in which we declare Inner class.

• If the local variable declared as "final" then we can access.

Ex.

```
class Test
{
    public void m1()
    {
        int x=10;
        class Inner
        {
            public void m2()
            {
                s.o.p(x);
            }
        }
        Inner i = new Inner();
        i.m2();
    }
```

```
p s v main(String[] args)
{
    Test t = new Test();
    t.m1();
}
}
```

O/P :- C.E.

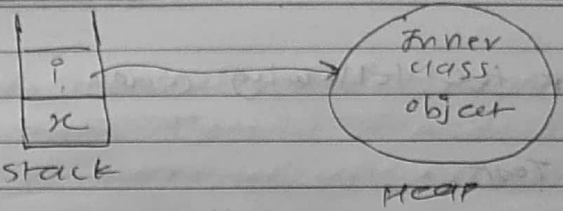
Local variable x is accessed from within inner class : needs to be declared final.

→ If we declare x as final we won't get any compile time error.

Reason :-

- Local variables get stored in stack
- Instances (objects) get stored in Heap.
- Local variable get destroyed once method execution completes
- Every final variable gets replaced by value at compile time only.

- In above program, when t.m1() gets executed, m1() method starts executing & Local variable x is created inside stack.



- And Instance (object) of Inner class get created in Heap & on which we can call m2() method of Inner class in which we are accessing Local variable x. (No problem till Now)

- But once method execution gets over stack will be destroyed so x is destroyed but Object of Inner class in Heap might not get destroyed, so when on that object m2() gets called, m2() will not be able to find variable x.

- So, to resolve this, this constraint is added that variable should be final if we want to access that variable inside Method Local Inner class.

- So, when it is declared final at compile time only variable will be replaced by its value, so no chance of getting error.

Que. Consider the following code.

```
class Test
{
    int i=10;
    static int j=20;

    public void m1()
    {
        int k=30;
        final int m=40;

        class Inner
        {
            public void m2()
            {
                line ①
            }
        }
    }
}
```

- At line ① following variables we can access directly?

i ✓
j ✓
k ✗
m ✓

- If we declare m1() method as static then at line-① which variables we can access directly

i ✗
j ✓
k ✗
m ✓

- If we declare m2() method as static then at line-① which variables we can access directly

i
j
k
m

we will get C.E bcz we can't declare static members inside Inner class.

- The only applicable modifiers for Method local Inner class are final / abstract / strictfp.

If we try to apply any other modifiers then we will get C.E.

⇒ 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.

⇝ Based on declaration & behaviour there are 3 types of Anonymous Inner classes

1) Anonymous Inner class that extends a class
2) Anonymous Inner class that implements an interface
3) Anonymous Inner class that defined inside arguments.

⇝ **Anonymous Inner class that extends a class**

Ex.
```
class Popcorn
{
    public void taste()
    {
        S.O.P("salty");
    }
}
class Test
{
    p s v main(str)
    {
        // For one time use only
        Popcorn p1 = new Popcorn()
        {
            public void taste()
            {
                S.O.P("spicy");
            }
        };
        p1.taste();

        Popcorn p2 = new Popcorn();
        p2.taste();

        // For one time use only
        Popcorn p2 = new Popcorn()
        {
            public void taste()
            {
                S.O.P("sweet");
            }
        };
        p2.taste();

        S.O.P(p1.getClass.getName());
        S.O.P(p2.getClass.getName());
        S.O.P(p3.getClass.getName());
    }
}
```

Not Popcorn object, but child class object

over riding

parent reference child object syntax

over riding

Spicy
O/P & Salty
Sweet
Test$1
PopCorn
Test$2

When above program compiles, 4 .class will be generated.
PopCorn.class
Test.class
Test$1.class
Test$2.class

→ Analysis :-

(1)
```
PopCorn p = new PopCorn();
```

- Just we are creating PopCorn object

(2)
```
PopCorn p = new PopCorn()
{
};
```

- we are declaring a class that extends popcorn without name (Anonymous Inner class)
- For that child class we are creating an object with Parent reference.

(3)
```
PopCorn p = new PopCorn()
{
    public void taste()
    {
        S.O.P("spicy");
    }
};
```

- We are declaring a class that extends PopCorn without name (Anonymous Inner class)
- In that child class we are overriding the taste method
- For that child class we are creating an object with parent reference

→ Defining a Thread by extending Thread class :-

(1) Normal class Approach

defining a Thread
```
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0; i<10; i++)
        {
            S.O.P("child Thread");
        }
    }
}
```
Job of Thread

```
class ThreadDemo
{
    p s v main(Str )
    {
        MyThread t = new
                     MyThread();
        t.start();

        for(int i=0; i<10; i++)
        {
            s.o.p("Main Thread");
        }
    }
}
```

O/p : Mixed O/P

(2) Anonymous Inner class Approach

```
class ThreadDemo
{
    p s v main(Str )
    {
        Thread t = new Thread()
        {
            public void run()
            {
                for(int i=0; i<10; i++)
                {
                    s.o.p("child Thread");
                }
            }
        };
        t.start();
        for(int i=0; i<10; i++)
        {
            s.o.p("main Thread");
        }
    }
}
```

O/p : Mixed O/P

Defining a Thread by implementing Runnable Interface

(1) Normal class Approach

```
class MyRunnable implements Runnable
{
    public void run()
    {
        for(int i=0; i<10; i++)
        {
            s.o.p("child Thread");
        }
    }
}
class ThreadDemo
{
    p s v main(Str )
    {
```

```
MyRunnable r = new MyRunnable
Thread t = new Thread(r);
t.start();
for (int i=0; i<10; i++)
{
    S.O.P("Main Thread");
}
}
}
```

O/P : Mixed O/P

(2) Anonymous Inner class Approach

```
class ThreadDemo
{
    p s v main(Str)
    {
        Runnable r = new Runnable()
        {
            public void run()
            {
                for (int i=0; i<10; i++)
                {
                    S.O.P("child Thread");
                }
            }
        };
        Thread t = new Thread(r);
        t.start();

        for (int i=0; i<10; i++)
        {
            S.O.P("Main Thread");
        }
    }
}
```

O/P : Mixed O/P

→ Another way

```
class ThreadDemo
{
    p s v main(Str)
    {
        new Thread(new Runnable()
        {
            public void run()
            {
                for (int i=0; i<10; i++)
                {
                    S.O.P("child Thread");
                }
            }
        }).start();

        for (int i=0; i<10; i++)
        {
            S.O.P("Main Thread");
        }
    }
}
```

o/p : Mixed o/p.

⇒ Normal Java class vs Anonymous Inner class

1) A Normal java class can extend only one class at a time, ofcourse Anonymous Inner class can extend only one class at a time

2) A Normal java class can implement any no. of interfaces simultaneously but Anonymous Inner class can implement only one interface at a time.

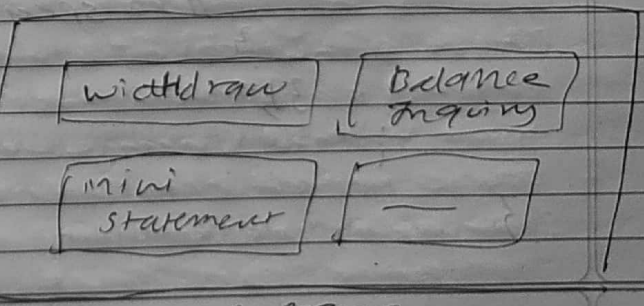
3) A Normal Java class can extend a class & can implement any no. of interfaces simultaneously, But Anonymous Inner class can extend a class or can implement an interface at a time but not both simultaneously.

4) In Normal Java classes, we can write any no. of constructors, But In Anonymous Inner classes, we can't write any constructor explicitly (Bcz, the name of the class & name of the constructor must be same, but Anonymous Inner class not have any name)

> **Note:** If the Requirement is standard & requires several times, then we should go for Normal top level class.
>
> If the Requirement is Temporary & requires only once (Instant use) then we should go for Anonymous Inner class

⇒ Where Anonymous Inner classes are best suitable?

⇒ we can use Anonymous Inner classes frequently in GUI Based Applications to implement Event handling.



```
class MyGUIFrame extends JFrame
{
    JButton b1, b2, b3;
    ;
    b1.addActionListener(
        new ActionListener(){

            public void actionPerformed
                        (ActionEvent e)
            {
                // b1 specific functionality
            }
    });

    b2.addActionListener(
        new ActionListener(){

            public void actionPerformed
                        (ActionEvent e)
            {
                // b2 specific functionality
            }
    });
    ;
    ;
}
```

Static Nested Classes

→ Sometimes we can declare Inner class with "static" modifier, such type of Inner classes are called "static Nested classes".

→ In the case of, Normal / Regular Inner class, without existing outer class object, there is no chance of existing Inner class object i.e Inner class object is strongly associated with outer class object.

→ But, In the case of static Nested classes, without existing outer class object, there may be a chance of existing Nested class object hence, static Nested class object is not strongly associated with outer class object.

→ Ex.

```
class Outer
{
    static class Nested
    {
        public void m1()
        {
```

```
        S.O.P("static Nested class Method");
    }
}
p s v main (Str-)
{
    Nested n = new Nested();
    n.m1();
}
}
```

O/P : static Nested class Method

↳ If you want to create Nested class object from outside of Outer class then we can creat as follows:

```
Outer.Nested n = new
                 Outer.Nested();
```

↳ In Normal / Regular Inner classes we can't declare any static members.

But In static Nested classes we can declare static members including main method, hence we can invoke static Nested class directly from command prompt.

```
class Test
{
    static class Nested
    {
        public static void main(Str-)
        {
            S.O.P("static Nested
                   class main method");
        }
    }
    public static void main (Str-)
    {
        S.O.P("Outer class main
               method");
    }
}
```

java Test ↵
Outer class main method

java Test$Nested ↵
static Nested class main method

↳ From Normal / Regular Inner classes, we can access both static & Non-static members of outer class directly but from static Nested classes, we can access, static members of outer class directly & we can't access non-static members.

Ex-

```
class Test
{
    int x = 10;
    static int y = 20;

    static class Nested
    {
        public void m1()
        {
            s.o.p(x);  ---- Line ①
            s.o.p(y);
        }
    }
}
```

O/P & C.E at line ①

Non-static variable x cannot be referenced from a static context

If we comment line ① then o/p ⇒ 20 y

Differences b/w Normal/Regular class & static Nested classes

Normal/Regular Inner classes	static Nested classes
- without existing outer class obj there is no chance of existing inner class object, i.e Inner class object is strongly associated with outer class object	without existing outer class object there maybe a chance of existing static Nested class Object, i.e static Nested class object is not strongly associated with outer class object
- In Normal Inner classes we can't declare static members, means no to main method hence we can't invoke it from cmd.	In static Nested classes we can declare static members including main method, so we can invoke it from cmd.
- From Normal Inner class, we can access both static/Non-static members of outer class directly	From static Nested Inner class, we can only access static members of outer class.

Various combinations of Nested classes & Interfaces

Case-1 (class inside a class)

- without existing one type of object is there is no chance of existing another type of object then we can declare class inside a class.
- Ex. University consists of several Departments, without existing University there is no chance of existing Departments, Hence we have to declare Department class inside University class.

```
class University
{
    class Department
    {
    }
}
```

Case-2 (Interface inside a class)

- Inside a class if we require multiple implementation of an interface & all these implementation are related to a particular class then we can define inside interface inside a class.

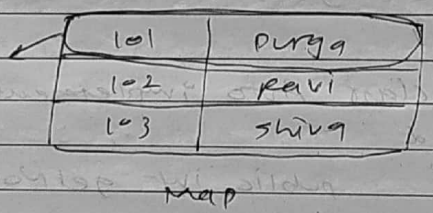
Ex.
```
class VehicleTypes
{
    interface Vehicle
    {
        public int getNoOfWheels();
    }
    class Bus implements Vehicle
    {
        public int getNoOfWheels()
        {
            return 6;
        }
    }
    class Auto implements Vehicle
    {
        public int getNoOfWheels()
        {
            return 3;
        }
    }
    .
    .
    .
}
```

Case-3 (Interface inside Interface)

- we can declare interface inside interface

- Ex. Map is a group of key, value pair & each key, value pair is called an Entry, without existing Map object, there is no chance of existing Entry object. Hence interface Entry is defined inside Map interface.

```
interface Map
{
    interface Entry
    {
    }
}
```



- Every interface present inside interface is always "public & static" whether we are declaring or not. Hence we can implement Inner interface directly w.o implementing outer interface.

- similarly whenever we are implementing outer interface we are not required to implement inner interface.

- So, we can implement outer & Inner interfaces independently.

- Ex.

```
interface Outer
{
    public void m1();
    interface Inner
    {
        public void m2();
    }
}
class Test1 implements Outer
{
    public void m1()
    {
        S.O.P("Outer interface
              method interface");
    }
}
class Test2 implements Inner
{
    public void m2()
    {
        S.O.P("Inner interface
              method interface");
    }
}
```

```
class Test
{
    p s v main (Strm)
    {
        Test1 t1 = new Test1();
        t1.m1();

        Test2 t2 = new Test2();
        t2.m2();
    }
}
```

→ case-4 (class inside an Interface)

- If functionality of a class is closely associated with interface then it is highly recommended to declare class inside interface.

- EX.

```
interface EmailService
{
    public void sendEmail(EmailDetails e);

    class EmailDetails
    {
        String to_list;
        String cc_list;
    }
}
```

- In the above example, EmailDetails is required only for EmailService & we are not using anywhere else, hence EmailDetails class is recommanded to declare inside EmailService interface.

- we can also implement/define a class inside interface to provide default implementation for that interface.

- EX. interface Vehicle

```
{
    public int getNoOfWheels();

    class DefaultImpl implements Vehicle
    {
        public int getNoOfWheels()
        {
            return 2;
        }
    }
}
class Bus implements Vehicle
{
    public int getNoOfWheels()
    {
        return 6;
    }
}
```

```
class Test
{
    p s v main(Str--)
    {
        Vehicle.DefaultImpl d =
            new Vehicle.DefaultImpl();

        S.o.p(d.getNoOfWheels());

        Bus b = new Bus();

        S.o.p(b.getNoOfWheels());
    }
}
```

Op :- 2
6

Note :- The class which is declared inside interface is always "public & static" whether we declare it or not. Hence we can create class object directly w.o having outer interface type object.

⇒ **Conclusions :-**

(1) Among class & interface, we can declare anything inside anything.

class A	class A	interface A	interface A
{	{	{	{
class B	interface B	class B	interface B
{	{	{	{
}	}	}	}
}	}	}	}

(2) The class / interface which we declare inside interface it by default "public & static", whether we declare it or not.

(3) The interface which is declared inside a class is always static but need not be public.