Inner Classes

- Sometimes we can declare a class inside another class such type of classes are called inner classes
- Inner classes concept introduced in 1.1 version to fix GUI bugs as a part of event handling but because of powerful features and 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.
- Example 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{
          Class department{
          }
  }
  ```

  Example2: Map is group of key value pairs and each pair is called "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{
          }
  }
  ```

  Note:

  1. without existing outer class object there is no chance of existing inner class object.
  2. The relation between outer and inner class is not a IS-A relation and it is HAS-A relationship(composition or aggregation)

Types of inner classes:

Based on position of declaration and behavior all inner classes are divided 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

```
Example1:class Outer{                          javac Outer.java→Outer.class
                Class Inner{                                   →Outer$Inner.class
                }
        }
                                        Java Outer→No such method error: main()
                                        Java Outer$Inner→No such method error: main()
```

```
Example2:class Outer{                                   javac Outer.java→Outer.class
            Class Inner{                                                  →Outer$Inner.class
            }
            Psvm(String args[]){
                        Sop("Outer main");
            }
    }
                                        Java Outer→Outer main
                                        Java Outer$Inner→No such method error: main()
```

- Inside inner class we cannot declare any static members hence we cannot declare main method and we cannot run inner class directly from CMD

```
Eg: class Outer{
            Class Inner{
                        Psvm(String args[]){
                                    Sop("inner main");
                        }
            }
}
CE: Inner classes cannot have static declarations
```

<u>Case1: Accessing inner class code from static area of outer class</u>

```
Class Outer{
            Class Inner{
                        Public void m1(){
                                    Sop("inner method");
                        }
            }                                           //short cut  i=new outer().new Inner();
            Psvm(String args[]){                        // new outer().new Inner().m1();
                        Outer o=new Outer();
                        Outer.Inner i=o.new Inner();
                        I.m1();
            }
}
```

<u>Case2: Accessing inner class code from instance area of outer class</u>

```
Class Outer{
            Class Inner{
                        Public void m1(){
                                    Sop("inner m1");
                        }
            }
            Public void m2(){
                        Inner i=new Inner();
                        I.m1();
            }
            PSVM(String args[]){
                        Outer o=new Outer();
```

```
                o.m2();
        }
}
```

<u>Case3: Accessing Inner class code outside of outer class</u>

```
Class Outer{
        Class inner{
                Public void m1(){
                        Sop("inner main");
                }
        }
}
Class Test{
        Psvm(String args[]){
                Outer o=new Outer();
                Outer.Inner i=o.new Inner();
                i.m1();
        }
}
```

- From normal or regular inner class we can access both static and non-static members of outer class directly.

  ```
  Eg:class Outer{
          Int x=10;
          Static int y=20;
          Class Inner{
                  Public void m1(){
                          Sop(x);
                          Sop(y);
                  }
          }
          Psvm(String args[]){
                  new Outer().new Inner().m1();
          }
  }
  ```

- 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".

  ```
  Eg: class Outer{
          Int x=10;
          Class Inner{
                  Int x=100;
                  Public void m1(){
                          Int x=1000;
                          Sop(x);
                          Sop(this.x);|sop(Inner.this);
                          Sop(Outer.this.x);
                  }
          }
  ```

```
        Psvm(String args[]){
                new Outer().new Inner().m1();
        }
}

o/P:    1000
        100
        10
```

- The only applicable modifiers for Outer class are “public, default, final, abstract, stricFp”. But for Inner classes applicable modifiers are above 5 + “private,protected,static”.

- Nesting of Innerclasses:
  Inside inner class we can declare another inner class i.e. nesting of inner classes is possible.

```
Eg: class A{
        Class B{
                Class C{
                        Public void m1(){
                                Sop(“inner most class method”);
                        }
                }
        }
}
A a=new A();
A.B b=a.new B();
A.B.C c=b.new C();
c.m1();
```

Method Local Inner classes:

Sometimes we can declare a class inside a method, such type of inner class is called *method local inner class.*

- 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 only within a method where we declared, outside of the method we cannot access, because of its less scope method local inner classes are most rarely used type of inner classes.

```
Eg: class Test{
        Public void m1(){
                Class Inner{
                        Public void sum(int x,int y){
                                Sop(x+y);
                        }
                }
                Inner i=new Inner();
                i.sum(10,20);
                i.sum(100,200);
```

```
                    i.sum(100,200);
                    }
            }
        }
Test t=new Test();
t.m1();
```

- We can declare method local inner class inside both instance and static methods
- If we declare inner class inside instance method then from that method local inner class we can access both static and non-static members of outer class directly. If we declare inner class inside static method then we can access only static members of outer class directly, from that method local inner class.

  ```
  Eg: class Test{
          Int x=10;
          Static int y=20;
          Public void m1(){                    //if m1 is static  we get CE:non static variable x
              Class Inner{                         cannot be reference from static context
                  Pubic void m2(){
                      Sop(x);
                      Sop(y)
                  }
              }
              Inner i=new Inner();
              i.m2();
          }
          Psvm(String  args[]){
              Test t=new Test();
              t.m1();
          }
  }
  ```

- From method local inner class we cannot access local variables of the method in which we declare inner class, **if the local variable declared as final then we can access.**

  ```
  Class Test{
          Public void m1(){
              Int x=10;                                  //if x is final it is valid
              Class Inner{
                  Public void m2(){
                      Sop(x);
                  }
              }
              Inner i= new Inner();
                  i.m2();
          }
          Psvm(String args[]){
              Test t=new Test();
              t.m1();
          }
  }
  ```

CE: local variable x is accessed from within inner class; needs to be declared final.

Q1. Consider the following code

```
class Test{
        int i=10;
        static int j=20;
        public vid m1(){
                int k=30;
                final int m=40;
                class Inner{
                        public void m2(){
                                line….(1)
                        }
                }
        }
}
```

At line 1 which of the following variables we can access directly

i→valid, j→valid, m→valid
k→ invalid based on above concept.

If m1() is static

i→ invalid, j→valid, k→ invalid, m→valid

if m2() is static

CE: we cannot declare static members inside inner class

- The only applicable modifiers for method local inner classes are final, abstract, strictFP. If we try applying any other modifier then we will get compile time error.

Anonymous Inner classes:

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 instance use(one time usage).
- Based on declaration and behavior 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 argument.
- <u>Anonymous inner class that extends a class:</u>

```
Class Popcorn{
        Public void taste(){
                Sop("Salty");
        }
}
```

```
Class Test{
        Psvm(String args[]){
                Popcorn p =new Popcron()
                {
                        Public void taste(){
                        Sop(“spicy”);
                        }
                };
                p.taste();                              //spicy
                Popcorn p1=new Popocorn();
                P1.taste();                             //salty
                Popcorn p2=new Popcorn()
                {
                        Public void taste(){
                        Sop(“sweet”);
                        }
                };
                P2.taste                                //sweet
                Sop(p.getClass().getName());            Test$1
                Sop(p1.getClass().getName());           PopCorn
                Sop(p2.getClass().getName());           Test$2
        }
}
```

The generated .class file are 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(){
               Sop(“spicy”);
       }
   }
   ```
   We are declaring a class that extends popcorn without name(anonymous inner class). In that child class we are overriding taste() method. For that child class we are creating an object with parent reference.

Example Defining a thread by extending Thread class

Anonymous inner class approach:

```
Class ThreadDemo{
    Psvm(String args[]){
            Thread t=new Thread()
            {
                    Public void run(){
                            For(int i=0;i<5;i++){
                                    Sop("child thread");
                            }
                    }
            };
            t.start();
            For(int i=0;i<5;i++){
                    Sop("main thread");
            }
    }
}
```

- Anonymous inner class that implements a interface:

  Defining a Thread by implementing Runnable interface:

```
Class ThreadDemo{
        Psvm(String args[]){
                Runnable r=new Runnable()
                {
                        Public void run(){
                                For(int i=0;1<5;i++){
                                        Sop("child thread");
                                }
                        }
                };
                Thread t=new Thread(r);
                t.start();
                for(int i=0;i<5;i++){
                        sop("main thread");
                }
        }
}
```

- Anonymous inner class that defined inside argument

```
Class ThreadDemo{
        Psvm(String args[]){
                new Thread(new Runnable()
                                {
                                        Public void run(){
                                                For(int i=0;i<10;i++){
                                                        Sop("child thread");
                                        }       }
                                }).start();
                For(int i=0;i<10;i++){
```

```
                Sop( “main thread”);
            }
        }
    }
```

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 class at a time.
2. A normal Java class can implement any number of interfaces simultaneously but anonymouse inner class can implement only one interface at a time.
3. A normal java class can extend a class and can implement any number of interfaces simultaneously but, anonymous inner class can extend a class or can implement an interface but not both simultaneously.
4. In normal Java class we can write any number of constructors simultaneously, but in anonymous inner classes we cannot write any constructor explicitly (because name of the class and name of constructor must be same but anonymous inner class not having any name).

Note: if the requirement is standard and required several times then we should go for normal top level class.
If requirement is temporary and required 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.

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 chance of existing nested class object hence static nested class object is not strongly associated with outer class object.

```
Eg: class Outer{
        Static class Nested{
            Public void m1(){
                Sop(“static nested method”);
            }
        }
        Psvm(String args[]){
            Nested n=new Nested();
            n.m1();
        }

}

//create nested class object from outside of outer class : Outer.Nested n=new Outer.Nested();
```

- In Normal or regular inner classes we cannot declare any static members, but in static nested classes we can declare static members including main method, hence we can invoke static nesed class directly from CMD.
- From normal|regular inner classes we can access both static and non-static members of outer class directly but from static nested classes we can access static member of Outer class directly and we cannot access non-static members.

```
Eg: class Test{
        Int x=10;
        Static int y=20;
        Static Class Nested{
                Public void m1(){
                        Sop(x);   //CE: non-static  variable x cannot be referenced from static
                        Sop(y);                              context

                }
        }
}
```

<u>Difference between Normal|regular inner class and static nested class</u>

Normal	Static nested class
Object are strongly associated	Objects are weakly associaed
Cannot declare static members	Can declare static member
Cannot have main method	Can have main method
Can access both static and instance member of outer class .	Can access only static members of outer class

Various combinations of nested classes and interfaces:

<u>Case1: class inside class</u>

Without existing one type of object if there is no chance of exisitin another type of object then we can declare a class inside a class.

Eg: University consists multiple departments without existing university there is no chance of existing department hence we have to declare department class inside university class.

```
Class University{
        Class Department{
        }
}
```

<u>Case2: Interface inside a class</u>

Inside a class if we require multiple implementations of an interface and all these implementations are related to a particular class then , we can define interface inside a class.

```
Class  VehicleType{
        Interface vehicle
        {
                Public int getNoOfWheel();
```

```
                    }
                    Class Bus implements vehicle {
                              Public int getNoOfWheel(){
                                        Return6;
                                        }
                              }
                    }
                    Class Auto implements vehicle{
                              Public int getNoOfWheel(){
                                        Return 3;
                              }
                    }
}
```

Case3: Interface inside interface
We can declare interface inside interface.
A map is a group of key-value pairs and each key-value pair is called entry, without existing map object there is no chance of existing entry object. Hence interface entry is defined inside map interface.

Every interface present inside interface is always public and static whether we are declaring or not, hence we can implement inner interface directly without implementing outer interface.

Similarly whenever we are implementing outer interface we are not required to implement inner interface i.e. we can implement outer and inner interfaces independently.

Case4: class inside Interface
if functionality of a class is closely associated with interface, then it is highly recommended to declare class inside interface.

```
Eg: interface emailService{
          Public void sendemail(EmailDetails e);
          Class EmailDetails{
                    String toList;
                    String ccList;
                    String subject;
                    String body;
          }
}
```

In the above example EmailDetails is required only for email service and we are not using anywhere else hence EmailDetails class is recommended to declare inside email service interface.

We can also define a class inside interface to provide default implementation for that interface.

```
Eg: Interface vehicle {
          Public int getNumberOfWheels();
          Class DefaultVehicle implements vehicle{
                    Public int getNumberOfWheels{
                              Return 2;
                    }
```

```
        }
}
Class Test{
        Psvm(String args[]){
                Vehicle.DefaultVehicle d=new Vehicle.DefaultVehlicle();
                Sop(d.getNumberOfWheels());
        }
}
```

In the above example is DefaultVehicle is the default implementation of vehicle interface.

Note: class declared inside interface is always public static whether we declare or not hence we can create object of the class directly, without having outer interface type object.

Conclusions:

1. Among classes and interfaces we can declare anything inside anything.
2. Interface declared inside interface is always public static, whether we are declaring or not.
3. Class declared inside interface is always public static, whether we are declaring or not.
4. Interface declared within class is always static need not be public.