8. Inner Classes

Inner class:—

→ 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 the part of Event Handling

→ But becoz of powerful features and benefits of Inner classes slowly programmers are 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.

Ex①: University consists of several departments. Without existing University there is no chance of existing Department.

Hence Department is the part of University and we have to declare Department class inside University class.

```
class University ——→ Outer class
{
    class Department ——→ Inner class
    {
    }
}
```

Ex②: Without existing Bank object there is no chance of existing Account object.

Hence we have to declare Account class inside Bank class.

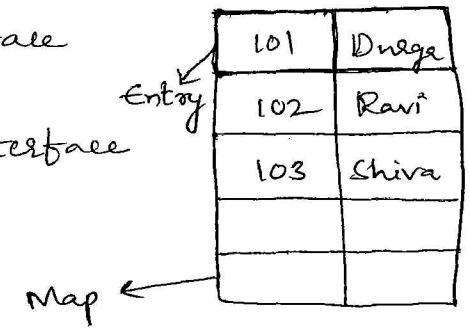
```
class Bank
{
    class Account
    {
    }
}
```

Ex③: Map is a collection of key-value pairs and each key-value pair is called an Entry.

Hence Map is considered as a collection of Entry objects.

Without existing Map object there is chance of existing Map Object. Hence interface Entry is defined inside Map interface.

```
interface Map ~~> Outer interface
{
    interface Entry ~~> Inner interface
    {
    =
    =
    }
}
```



***

Note①:- Without existing Outer class object there is no chance of existing Inner class object.

② The relationship b/w Outer class & Inner class is not parent to child relationship. It is Has-A relationship (Composition/Aggregation).

→ Based on purpose & position of declaration all Inner classes are divided into 4 types.

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

1. Normal (or) 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 (or) Regular Inner class.

Ex①:

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

Outer.java

Normal (or) Regular Inner class

java Outer.java

Outer.class Outer$Inner.class

java Outer ↵

RE: NoSuchMethodError: main

java Outer$Inner ↵

RE: NoSuchMethodError: main

Ex②:

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

java Outer.java

Outer.class Outer$Inner.class

java Outer ↵

O/P: Outer class main method

java Outer$Inner ↵

RE: NoSuchMethodError: main

***

→ Inside inner class we can't declare static members. Hence we can't declare main(-) method & we can't invoke inner class directly from command prompt.

***

→ If we are trying to declare static members inside inner classes then we will get compile time error.

Ex:

```
class Outer
{
    class Inner
    {
        P s v main(String[] args)
        {
            S.o.p("Inner class main method");
        }
    }
}
```

CE: Inner classes cannot have static declaration.

*** ① Accessing Inner class code from static area of Outer class :-

Ex:
```
class Outer
{
  class Inner
  {
    public void m1()
    {
      S.o.p("Inner class method");
    }
  }
  p s v m(-)
  {
    Outer o=new Outer();
    Outer.Inner i= o.new Inner();
    i.m1();
  }
}
```

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

o/p : Inner class method

new Outer().new Inner().m1();

Case (ii) : Accessing Inner class code from instance area of Outer class :-

Ex:
```
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 m(-)
  {
    Outer o=new Outer();
    o.m2();
  }
}
```

o/p : Inner class method.

Case(iii): Accessing Inner class code from outside of Outer class:-

Ex:
```
class Outer
{
  class Inner
  {
    public void m1()
    {
      S.o.p("Inner class method");
    }
  }
}
class Test
{
  p s v m(-)
  {
    Outer o=new Outer();
    Outer.Inner i=o.new Inner();
    i.m1();
  }
}
```

O/P : Inner class method.

Accessing Inner class code

From static area of Outer class

(or)

From outside of Outer class

```
Outer o=new Outer();
Outer.Inner i=o.new Inner();
i.m1();
```

From instance area of Outer class

```
Inner i=new Inner();
i.m1();
```

→ From Normal (or) Regular Inner class, we can access both static & non-static members of Outer class directly.

Ex:
```
class Outer
{
  int x=10;
  static int y=20;
  class Inner
  {
    public void m1()
    {
      S.o.p(x); ⇒ o/p : 10
      S.o.p(y); ⇒ o/p : 20
    }
  }
  p s v m(-)
  {
    Outer o=new Outer();
    Outer.Inner i=o.new Inner();
    i.m1();
  }
}
```

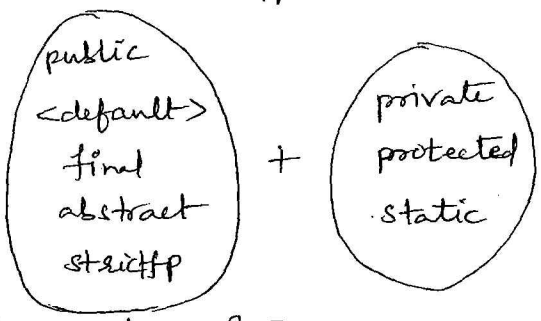
→ Within the Inner class this always refers current Inner class object. But to refer current Outer class object we have to use

Outer_class_name.this

Ex:
```
class Outer
{
  int x=10;
  class Inner
  {
    int x=100;
    public void m1();
    {
      int x=1000;
      S.o.p(x); ⇒ o/p : 1000   (or) → S.o.p(Inner.this.x);
      S.o.p(this.x); ⇒ o/p : 100
    } S.o.p(Outer.this.x); ⇒ o/p : 10
  }
}
p s v m(-)
{
  new Outer().new Inner().m1();
}
```

→ The only applicable modifiers for top level classes (Outer classes) are public, <default>, final, abstract & strictfp.

→ But for Inner classes applicable modifiers are



Nesting of Inner classes:-

→ We can declare Inner class inside Inner class i.e., nesting of Inner classes is possible.

Ex:

```
class A
{
  class B
  {
    class C
    {
      public void m1()
      {
        S.o.p("Inner most class method");
      }
    }
  }
  P s v m(-)
  {
    A a=new A();
    A.B b=a.new B();
    A.B.C c=b.new C();
    c.m1();
  }
}
```

A.B.C c=new A().new B().new C();

new A().new B().new C().m1();

O/P: Inner most class method

2. 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 classes is to define method specific repeatedly required f°.ality.

→ Method local inner classes are best suitable to handle nested method requirements.

→ We can access Method local inner classes only inside the method in which we declared it i.e., from outside the method we can't access.

→ Hence Method local inner classes are most rarely used type of Inner classes (beeoz their scope is very less).

Ex:

```
class Outer
{
  public void m1()
  {
    class Inner
    {
      public void sum(int x, int y)
      {
        S.o.p(x+y);
      }
    }
    Inner i=new Inner();
    i.sum(10,20); => o/p : 30
    i.sum(100,200); => o/p : 300
    i.sum(1000,2000); => o/p : 3000
  }
  P S V m(—)
  {
    Test t=new Test();
    t.m1();
  }
}
```

*** → We can declare Inner class inside both instance & static methods.

→ If we declare inside instance method then we can access both static & non-static members of Outer class directly.

→ If we declare inside static method then we can access only static members of Outer class from that Method local inner class.

Ex:
```
class Test
{
  int x=10;
  static int y=20;
  public void m1()
  {
    class Inner
    {
      public void m2()
      {
        S.o.p(x);  ⟹ O/P : 10   → (1)
        S.o.p(y);  ⟹ O/P : 20
      }
    }
    Inner i=new Inner();
    i.m2();
  }
  P S V m()
  {
    Test t=new Test();
    t.m1();
  }
}
```

→ If we declare m1() method as static then at line (1) we will get compile time error saying non-static variable x cannot be referenced from a static context.

*** → From Method local inner class we can't access local variables of the method in which we declared that inner class.

*** → But if that local variable declared as final then we can access.

Ex:

```
class Test
{
  int x=10;
  public void m1()
  {
    int y=20;
    class Inner
    {
      public void m2()
      {
        S.o.p(x);
        S.o.p(y);
      }
    }
    Inner i=new Inner();
    i.m2();
  }
  P S V m(-)
  {
    Test t=new Test();
    t.m1();
  }
}
```

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

→ If we declared y as final then we won't get any CE.

Q: Consider the following code.

```
class Test
{
  int i=10;
  static int j=20;
  public void m1()
  {
    int k=30;
    final int l=40;
    class Inner
    {
      public void m2()
      {
        line①
      }
    }
  }
}
```

At line① which of the following variable we can access?

1) i ✓
2) j ✓
3) k ✗
4) l ✓

Q: If we declare m1() method as static then at line① which variables we can access?

1) i ✗
2) j ✓
3) k ✗
4) l ✓

Q: If we declare m2() method as static then at line① which variables we can access?

Ans: We will get CE becoz Inner classes can't have static declarations.

*** → The only applicable modifiers for Method local inner classes are final, abstract & strictfp.

3. 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 instant use (1 time usage).

→ 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.

1. Anonymous Inner class that extends a class:—

Ex:
```
class PopCorn
{
  public void taste()
  {
    S.o.p("Salty");
  }
  100 more methods
}
```

```
class Test
{
  P S V m(-)
  {
    PopCorn p=new PopCorn()
    {
      public void taste()
      {
        S.O.p("Spicy");
      }
    };
    P.taste(); => O/P: Spicy
    PopCorn P1=new PopCorn();
    P1.taste(); => O/p: Salty
    PopCorn P2=new PopCorn()
    {
      public void taste()
      {
        S.O.p("Sweet");
      }
    };
    P2.taste(); => O/P: Sweet
  }
}
```

Analysis:—

1. PopCorn p=new PopCorn();

→ We are creating just PopCorn object.

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

→ We are creating child class for 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 creating child class for PopCorn without name (Anonymous Inner class).

→ In child class, we are overriding taste() method for that child class we are creating an object with parent reference.

Ex:
```
class Test
{
  P S V m(-)
  {
    Thread t=new Thread()
    {
      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");
    }
  }
}
```

2. Anonymous Inner class that implements an interface:-

Ex:
```
class Test
{
  P S V m(-)
  {
```

```
Runnable r=new Runnable()
{
  public void run()
  {
    for(int i=0; i<10; i++)
    {
      S.o.p("Child Thread");
    }
  }
};
Thread t=new Thread();
t.start();
for(int i=0; i<10; i++)
{
  S.o.p("main Thread");
}
}
}
```

3. Anonymous Inner class that define inside arguments :-

Ex:

```
class Test
{
  p s v m(-)
  {
    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");
    }
  }
}
```

*** Anonymous Inner Class Vs Normal Class :—

→ A Normal Java class can extend only one class at a time, but Anonymous Inner class also can extend only one class at a time.

→ A Normal Java class can implement any no. of interfaces at a time, but Anonymous Inner class can implement only one interface at a time.

→ A Normal Java class can extend a class and can implement any no. of interfaces simultaneously, but Anonymous Inner class can extend a class & implement an interface simultaneously.

*** → The main application area of Anonymous Inner classes is to implement GUI based applications for Event Handling.

Ex①:

```
import java.awt.*;
import java.awt.event.*;
public class JarDemo
{
  P S V M(-)
  {
    Frame f=new Frame();
    f.addWindowListener(new WindowAdapter()
    {
      public void windowClosing(WindowEvent e)
      {
        System.exit(0);
      }
    });
    f.add(new Label("I can create Executable Jar File!!!"));
    f.setSize(500,500);
    f.setVisible(true);
  }
}
```

Ex②:
```
class SomeGUI extends JFrame
{
    JButton b1, b2, b3, b4, b5, b6;
    b1.addActionListener(new ActionListener()
                         {
                           public void actionPerformed(ActionEvent e)
                           {
                             // perform our required operation
                           }
                         });
    b2.addActionListener(new ActionListener()
                         {
                           public void actionPerformed(ActionEvent e)
                           {
                             // perform our required operation
                           }
                         });
}
```

4. Static Nested classes :—

→ Sometimes we can declare inner class with static modifier such type of inner classes are called Static Nested classes.

→ In Normal (or) Regular Inner class, without existing Outer class object there is no chance of existing Inner class object.

→ But in case of Static Nested classes, without existing Outer class object there may be a chance of existing static Nested class object. i.e., Nested class object is not strongly associated with Outer class object.

Ex:
```
class Test
{
    static class Nested
    {
```

```
    public void m1()
    {
      S.o.p ("Static Nested class method");
    }
  }
  P s v m(-)
  {
    //Test.Nested n=new Test.Nested(); (outside of the class)
    Nested n=new Nested();
    n.m1();
  }
}
```

→ In Normal or Regular Inner classes, we can't take static declarations, but in Static Nested classes we can take static declarations including main(-) method also.

→ Hence we can invoke static Nested class directly from command prompt.

Ex:

```
class Test
{
  static class Nested
  {
    P s v m(-)
    {
      S.o.p("Static Nested class main method");
    }
  }
  P s v m(-)
  {
    S.o.p ("Outer class main method");
  }
}
```

javac Test.java

Test.class Test$Nested.class

java Test ↵

o/p: Outer class main method.

java Test$Nested ↵

o/p: Static Nested class main method.

→ From Normal (or) Regular Inner class, we can access both static & non-static members of Outer class directly.

→ But from Static Nested class, we can access only static members

of Outer class directly.

Ex:
```
class Test
{
    int x=10;
    static int y=20;
    static class Nested
    {
        public void m1()
        {
            S.o.p(x); --> CE: non-static variable x cannot be
            S.o.p(y); ✓   referenced from a static context.
        }
    }
}
```
O/P : 20

***

Differences b/w Normal Inner class & Static Nested Class :—

Normal Inner class	Static Nested Class
1. 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 (Composition).	1. Without existing Outer class object there may be a chance of existing Static Nested class object i.e., Static Nested class object is not strongly associated with Outer class object (Aggregation).
2. In Normal (or) Regular Inner classes, static declarations are not allowed.	2. In Static Nested classes, we can declare static members.
3. In Normal Inner class, we can't declare main(-) method & hence we can't invoke inner class directly from command prompt.	3. In Static Nested classes, we can declare static members & hence we can invoke Static Nested class directly from cmd prompt.
4. From Normal Inner class, we can access both static & non-static members of Outer class directly.	4. From Static Nested classes, we can access only static members of Outer class directly.

Various possible combinations of classes and interfaces :-

1. class inside a class :-

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

Ex:
```
class University
{
  class Department
  {
  }
}
```

→ Without existing University object there is no chance of existing Department object.

→ Hence we have to define Department class inside University class.

2. interface inside a class :-

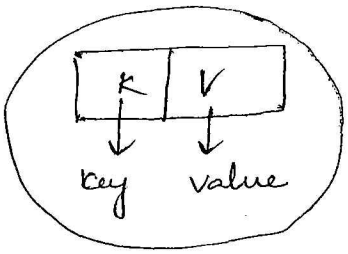
→ Inside a class, if we require multiple implementations of an interface & these implementations are relavant to a particular class then we can define an interface inside a class.

Ex:
```
class VeehicleType
{
  interface Veehicle
  {
    public int getNoOfWheels();
  }
  class Bus implements Vehicle
  {
    public int getNoOfWheels()
    {
      return 6;
    }
  }
  class Auto implements Veehicle
  {
    public int getNoOfWheels()
    {
      return 3;
    }
  }
}
```

3. interface inside interface :—

→ We can declare interface inside interface.

Ex:
```
interface Map
{
  :
  interface Entry
  {
    :
    public Object getKey();
    public Object getValue();
    public Object setValue(Object o);
  }
}
```



***
→ Every inner interface is always static.

→ Hence we can implement Inner interface directly without implementing Outer interface.

→ Whenever we are implementing Outer interface we are not required to implement Inner interface i.e., Outer & Inner interfaces we can implement independently.

Ex:
```
interface Outer
{
  public void m1();
  interface Inner
  {
    public void m2();
  }
}
class Test1 implements Outer.Inner
{
  public void m2()
  {
    S.o.p("Inner interface method");
  }
}
class Test2 implements Outer
{
  public void m1()
  {
    S.o.p("Outer interface method");
  }
}
```

4. class inside interface :—

→ If a class functionality is closely associated with the use of interface then it is highly recommended to declare that class inside interface.

Ex:
```
interface EmailService
{
    public void sendMail(EmailDetails e);
    class EmailDetails
    {
        private String to_list;
        private String subject;
        private String cc_list;
    }
}
```

→ In the above example, EmailDetails functionality is required for EmailService & we are not using anywhere else.

→ Hence it is highly recommended to declare EmailDetails class inside EmailService interface.

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

Ex:
```
interface Vechicle
{
    public int getNoOfWheels();
    class DefaultVechicle implements Vehicle
    {
        public int getNoOfWheels()
        {
            return 2;
        }
    }
    class Bus implements Vehicle
    {
        public int getNoOfWheels()
        {
            return 4;
        }
    }
```

```
class Test
{
  p s v m(-)
  {
    Vechicle.DefaultVechicle d=new Vechicle.DefaultVechicle();
    S.o.p(d.getNoOfWheels());  ⇒ O/P : 2
    Bus b=new Bus();
    S.o.p(b.getNoOfWheels());  ⇒ O/P : 7
  }
}
```

→ In the above example, DefaultVechicle is the default implementation of Vechicle interface where as Bus is customized implementation of Vechicle interface.

*** Note:- Every class which is declared inside interface is always public static. whether we are declaring or not.

Hence we can create object directly without implementing interface & without creating an instance of interface type.

Conclusions:-

→ We can declare anything inside anything.

```
class X
{
  class Y
  {
  }
}
```

```
class X
{
  interface Y
  {
  }
}
```

```
interface X
{
  class Y
  {
  }
}
```

```
interface X
{
  interface Y
  {
  }
}
```

→ Nested interfaces are always static, but Nested class need not be static always.