Inner Classes:
—-----------------
Declaring a class inside another class is called Inner class.
Syntax:
```
class Outer{
      class Inner{
            —--------
      }
}
```

Advantages:
1. Modularity
2. Abstraction
3. Security
4. Shareability
5. Reusability

Modularity:
In Java applications, we are able to provide modularity by declaring packages, but inside a class if we want to provide modularization over the code then we have to use “Inner Classes”.
EX:
```
class Maths{
      class Alzebra{
            —-----
      }
      class Trigonometry{
            —-----
      }
      class Calculus{
            —---
      }
      class Geometry{
            —-
      }
}
```

Abstraction:
—-------------
If we declare variables and methods inside an inner class then that variables and methods are not available to outside of the inner class, because inner classes are able to hide their internal declarations., so inner classes are able to provide Abstraction.

```
EX:
class A{
      class B{
            int i = 10;
      }
      class C{
            int j = i + 10; ------> Error
      }
}
```

Security:

In JAva applications, we are unable to declare an outer class as private but we are able to declare an inner class as Private, so inner classes are able to improve Security.

```
EX:
private class A{
}
Status: Compilation Error
EX:
class A{
      private class B{
      }
}
Status: Valid.
```

Shareability:

In Java applications, it is not possible to declare an outer class as static , but it is possible to declare an inner class as static, so inner classes are able to improve Shareability.

```
EX:
static class A{
}
Status: Compilation Error

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

Reusability:
—------------
In java applications, it is possible to extend one inner class to another inner class, so inner classes are able to improve Reusability.

In Java applications, there are four types of inner classes.
1. Member Inner classes
2. Static Inner classes
3. Method Local Inner classes
4. Anonymous Inner classes

If we compile a Java file that includes inner classes then the compiler will generate a separate .class file for the outer class and a separate .class file for the inner class.
EX:
```
class Outer{
      class Inner{
      }
}
```

If we compile the above program then the compiler will generate the .,class files like below.
```
Outer.class —---> Outer class
Outer$Inner.class —--> Inner class.
```

Member Inner class:
—-----------------------
Declaring a non-static class inside a class is called a member inner class.
Syntax:
```
class Outer{
      class Inner{
            —------
      }
}
```

If we declare variables and methods inside an inner class and if we want to access them in the main class then we have to create an object for the respective inner class and we have to use the generated reference variable.

To create an object for the member inner classes then we have to use the following syntax.

```
OuterClassName.InnerClassName refVar = new OuterClass().new InnerClass();
```

In Java applications, By using outer class reference variables we are able to access only outer class members, not possible to access inner class members, by using inner class reference variables we are able to access only inner class members, not possible to access outer class members.

In Java applications, by default all outer class members are available to inner classes, but inner class members are not available to outer classes.

In general, Member inner classes are not allowed static declaration directly , but member inner classes are able to allow static variables along with the final keyword[In the case of variables only].

Note: The above rule is valid up to JAVA 15 version , but from JAVA 16 version onwards static declarations are allowed in member inner classes.

EX:

```
class A{
   int i =10;
   static int k = 30;
   void m1(){
       System.out.println("m1-A");
       //System.out.println(j); ---> Error
       //m2(); ----> Error
       //m3(); ----> Error
   }
   class B{
       int j =20;
       static final int m = 40;
```

```
        void m2(){
            System.out.println("m2-B");
            System.out.println(i);
            m1();
        }
        void m3(){
            System.out.println("m3-B");
        }
    }
}
public class Main {
    public static void main(String[] args) {
        A a = new A();
        System.out.println(a.i);
        //System.out.println(a.j); ---> Error
        a.m1();
        //a.m2(); ----> Error

        A.B ab = new A().new B();
        //System.out.println(ab.i); ----> Error
        //ab.m1(); ---->Error
        System.out.println(ab.j);
        ab.m2();
        ab.m3();

    }
}
```

Inheritance Over Inner classes:

In Java applications , we are able to define inheritance relations between inner classes in the following cases.

Case#1: We can extend one inner class to another inner class , but both the inner classes must be provided in the same outer class.

EX:

```java
class A{
   class B{
       int i = 10;
       void m1(){
           System.out.println("m1-B");
       }
   }
   class C extends B{
       int j = 20;
       void m2(){
           System.out.println("m2-C");
       }
   }
}
public class Main {
   public static void main(String[] args) {
       A.B ab = new A().new B();
       System.out.println(ab.i);
       ab.m1();

       //System.out.println(ab.j); ---> Error
       //ab.m2(); ---> Error

       A.C ac = new A().new C();
       System.out.println(ac.i);
       System.out.println(ac.j);
       ac.m1();
       ac.m2();

   }
}
```

```
EX:
class A{
     class B{
     }
}
class C{
     class D extends B{
     }
}
Status: Invalid
```

Case#2: We can inherit a class from an inner class existing in an outer class.

EX:

```
class A{
   void m1(){
       System.out.println("m1-A");
   }
}
class B{
   class C extends A{
       void m2(){
           System.out.println("m2-C");
       }
   }
}
public class Main {
   public static void main(String[] args) {
       A a = new A();
       a.m1();

       B.C bc = new B().new C();
       bc.m1();
       bc.m2();
   }
}
```

Case#3: We can inherit from an outer class to the inner class.
EX:
```
class A{
   void m1(){
       System.out.println("m1-A");
   }
   class B extends A{
       void m2(){
           System.out.println("m2-B");
       }
   }
}
public class Main {
   public static void main(String[] args) {
       A a = new A();
       a.m1();
       A.B ab = a.new B();
       ab.m1();
       ab.m2();
   }
}
```

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

Ans:

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

EX:
```
class A{
   interface I{
       void m1();
       void m2();
```

```
        void m3();
    }
    class B implements I{
        public void m1(){
            System.out.println("m1-B");
        }
        public void m2(){
            System.out.println("m2-B");
        }
        public void m3(){
            System.out.println("m3-B");
        }
    }
}
public class Main {
    public static void main(String[] args) {
        A.I ai = new A().new B();
        ai.m1();
        ai.m2();
        ai.m3();

    }
}
```

Q)Is it possible to declare an abstract class inside a class?

Ans:

Yes, It is possible to declare an abstract class inside a class but we must provide the subclass for the abstract class within the same outer class.
EX:

```
class A{
    abstract class B{
        void m1(){
            System.out.println("m1-B");
        }
        abstract void m2();
        abstract void m3();
    }
    class C extends B{
        void m2(){
            System.out.println("m2-C");
        }
        void m3(){
            System.out.println("m3-C");
        }
    }
}
public class Main {
    public static void main(String[] args) {
        A.B ab = new A().new C();
        ab.m1();
        ab.m2();
        ab.m3();
    }
}
```

Q)Is it possible to declare a class inside an abstract class?

Ans:

Yes, It is possible to declare a class inside an abstract class, in this case, to access members of the inner class we have to take a subclass for the abstract class and we have to create an object for the inner class through a subclass object.
EX:

```
abstract class A{
    class B{
        void m1(){
            System.out.println("m1-B");
        }
        void m2(){
            System.out.println("m2-B");
        }
        void m3(){
            System.out.println("m3-B");
        }
    }
}
class C extends A{
    // Here class B is an inner class to class C also.
}
public class Main {
    public static void main(String[] args) {
        A.B ab = new C().new B();
        ab.m1();
        ab.m2();
        ab.m3();
    }
}
```

Q)Is it possible to declare an abstract class inside an abstract?

Ans:

Yes, it is possible to declare an abstract class in another abstract class, but we have to declare a subclass for the inner abstract class either in the same outer abstract class or in the subclass of the outer abstract class.

```
EX:
abstract class A{
   abstract class B{
       void m1(){
           System.out.println("m1-B");
       }
       abstract void m2();
       abstract void m3();
   }
   class C extends B{
       void m2(){
           System.out.println("m2-C");
       }
       void m3(){
           System.out.println("m3-C");
       }
   }
}
class D extends A{
   // Here abstract class B and the subclass C are inner
classes in Class D.
}
public class Main {
   public static void main(String[] args) {
       A.B ab = new D().new C();
       ab.m1();
       ab.m2();
       ab.m3();

       D.B db = new D().new C();
       db.m1();
       db.m2();
       db.m3();
   }
}
```

Q)Is it possible to declare an interface inside an abstract class?

Ans:

Yes, it is possible to declare an interface inside an abstract , but the implementation class for the interface class must be provided either in the same outer abstract class or in the subclass of the outer abstract class.

EX:

```
abstract class A{
   interface I{
       void m1();
       void m2();
       void m3();
   }
   class B implements I{
       @Override
       public void m1() {
           System.out.println("m1-B");
       }

       @Override
       public void m2() {
           System.out.println("m2-B");
       }

       @Override
       public void m3() {
           System.out.println("m3-B");
       }
   }
}
class C extends A{
   // Here interface I and the implementation class B are
inner classes in class C.
}
public class Main {
   public static void main(String[] args) {
        A.I ai = new C().new B();
        ai.m1();
        ai.m2();
        ai.m3();
   }
}
```

Static Inner classes:

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

Syntax:

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

If we declare variables and methods in the static inner class and if we want to access those members in the main class then we have to create an object for the static inner class.

```
OuterClass.InnerClass refVar = new
OuterClass.InnerClass();
```

Static inner classes are allowing static declarations directly.

Static inner classes are able to allow only static members from the outer class, not allowing instance members from the outer class.

EX:
```
class A{
   int i = 10;
   static int j = 20;
   static class B{
       void m1(){
           System.out.println("m1-B");
           //System.out.println(i); ---> Error
           System.out.println(j);
       }
       void m2(){
           System.out.println("m2-B");
       }
       static void m3(){
           System.out.println("m3-B");
       }
   }
}
public class Main {
   public static void main(String[] args) {
       A.B ab = new A.B();
       ab.m1();
       ab.m2();
       ab.m3();
       A.B.m3();


   }
}
```

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

Ans:

Yes, it is possible to declare a class inside an interface, if we declare a class inside an interface then that class is a static inner class by default inside the interface and we can create an object for the inner class as like a static inner class.

EX:

```
interface I{
   class A{
       void m1(){
           System.out.println("m1-A");
       }
       void m2(){
           System.out.println("m2-A");
       }
       void m3(){
           System.out.println("m3-A");
       }
   }
}
public class Main {
   public static void main(String[] args) {
       I.A ia = new I.A();
       ia.m1();
       ia.m2();
       ia.m3();
   }
}
```

Q)Is it possible to declare an abstract class inside an interface?

Ans:

Yes, it is possible to declare an abstract class inside an interface , but we must provide a subclass for the abstract class in the same outer interface, here the provided subclass is by default a static inner class, we can create objects for the subclass as like a static inner class.

EX:
```
interface I{
   abstract class A{
       void m1(){
           System.out.println("m1-A");
       }
       abstract void m2();
       abstract void m3();
   }
   class B extends A{
       void m2(){
           System.out.println("m2-B");
       }
       void m3(){
           System.out.println("m3-B");
       }
   }
}
public class Main {
   public static void main(String[] args) {
       I.A ia = new I.B();
       ia.m1();
       ia.m2();
       ia.m3();
```

```
    }
}
```

Q)Is it possible to declare an interface inside another interface?

Ans:

Yes, it is possible to declare an interface in another interface, but we must provide the implementation class for the inner interface in the same outer interface, here the provided implementation inside the interface is by default static inner class and we can object for the implementation class as like a static inner class.

EX:

```
interface I1{
    interface I2{
        void m1();
        void m2();
        void m3();
    }
    class A implements I2{
        public void m1(){
            System.out.println("m1-A");
        }
        public void m2(){
            System.out.println("m2-A");
        }
        public void m3(){
            System.out.println("m3-A");
        }
    }
}
public class Main {
   public static void main(String[] args) {
       I1.I2 i12 = new I1.A();
       i12.m1();
       i12.m2();
       i12.m3();
   }
}
```

Method Local Inner Class:

If we declare a class inside a method then that class is Method Local Inner Class.

If we declare a class inside a method then that class will have scope up to the respective method only, that is, we must create objects for the inner class in the same method only and we must access inner class members within the same method only.

EX:

```
class A{
   void m1(){
       class B{
           void m2(){
               System.out.println("m2-B");
           }
           void m3(){
               System.out.println("m3-B");
           }
       }
       B b = new B();
       b.m2();
       b.m3();
   }
}
```

```java
public class Main {
   public static void main(String[] args) {
       A a = new A();
       a.m1();
   }
}
```

Note: Up to Java 15 version, interfaces are not allowed inside the methods, but from Java 16 version onwards methods are allowing interfaces.

EX:

```java
class A{
   void m1(){
       interface I{
           void m1();
           void m2();
           void m3();
       }
       class B implements I{
           public void m1(){
               System.out.println("m1-B");
           }
           public void m2(){
               System.out.println("m2-B");
           }
           public void m3(){
               System.out.println("m3-B");
           }
       }
       I i = new B();
       i.m1();
       i.m2();
       i.m3();

   }
}
public class Main {
   public static void main(String[] args) {
        A a = new A();
        a.m1();
   }
}
```

Anonymous Inner classes:

Anonymous inner class is a nameless inner class, it can be used for classes, abstract classes and interfaces to override methods or to provide implementation for the abstract methods.

Syntax:

```
EntityName refVar = new EntityName(){
    ----Implementation for the methods----
};
```

If we want to access members of the anonymous inner class we have to use the refVar.

The main advantage of anonymous inner class is to avoid providing the sub classes in method overriding, subclasses for the abstract classes and implementation classes for the interfaces.

EX1:

```
class A{
   void m1(){
        System.out.println("m1-A");
   }
   void m2(){
        System.out.println("m2-A");
    }
}
public class Main {
    public static void main(String[] args) {
        A a = new A(){
             void m1(){
                 System.out.println("m1-AIC");
             }
        };
        a.m1();
        a.m2();

    }
}
```

EX:

```
abstract class A{
    void m1(){
        System.out.println("m1-A");
    }
    abstract void m2();
    abstract void m3();
}
public class Main {
    public static void main(String[] args) {
        A a = new A(){
            void m2(){
                System.out.println("m2-AIC");
            }
            void m3(){
                System.out.println("m3-AIC");
            }
        };
        a.m1();
        a.m2();
        a.m3();
    }
}
```

EX:

```
interface I{
   void m1();
   void m2();
   void m3();
}

public class Main {
   public static void main(String[] args) {
       I i = new I(){
           public void m1(){
               System.out.println("m1-AIC");
           }
           public void m2(){
               System.out.println("m2-AIC");
           }
           public void m3(){
               System.out.println("m3-AIC");
           }
       };
       i.m1();
       i.m2();
       i.m3();
   }
}
```

In general, we will use anonymous inner classes when we have an abstract class reference variable or an interface reference variable as a parameter to the methods.

EX:

```
interface I{
   void m1();
}

class B{
   void m2(I i){
        System.out.println("m2-B");
        i.m1();
   }
}
public class Main {
   public static void main(String[] args) {
        I i = new I(){
            public void m1(){
                System.out.println("m1-AIC");
            }
        };
        B b = new B();
        b.m2(i);

   }
}
```

In the above program, we can pass an anonymous inner class as a parameter to the method directly instead of passing the Anonymous inner class reference variable.

EX:

```
interface I{
   void m1();
}

class B{
   void m2(I i){
        System.out.println("m2-B");
        i.m1();
   }
}
```

```java
public class Main {
   public static void main(String[] args) {
        B b = new B();
        b.m2(new I(){
            public void m1(){
                System.out.println("m1-AIC");
            }
        });

   }
}
```