● 내부 클래스(inner class): 클래스 내에 선언된 클래스로, 내부 클래스(inner class)와 내부 클래스를 감싸고 있는 외부 클래스(otuer class)가 서로 긴밀한 관계에 있다. 두 클래스의 멤버 간에 서로 쉽게 접근할 수 있다는 장점과 외부에는 불필요한 클래스를 감춤으로써 코드의 복잡성을 줄일 수 있다(캡슐화)는 장점을 얻을 수 있다. 이때 내부 클래스는 외부 클래스를 제외하고는 다든 클래스에서 잘 사용되지 않는 것이어야 한다.

▶ 내부 클래스의 종듀와 특징: 선언 위치에 따라 구분되어진다. 유효범위와 성질이 변수와 유사하다.

내부 클래스	특징
인스턴스 클래스 (instance class)	외부 클래스의 멤버변수 선언위치에 선언한다. 외부 클래스의 인스턴스멤버처럼 다두어진다. 주로 외부 클래스의 인스턴스멤버들과 관련된 작업에 사용될 목적으로 선언된다.
스태틱 클래스 (static class)	외부 클래스의 멤버변수 선언위치에 선언한다. 외부 클래스의 static멤버처럼 다두어진다. 주로 외부 클래스의 static멤버, 특히 static메서드에서 사용될 목적으로 선언된다.
지역 클래스 (local class)	외부 클래스의 메서드나 초기화블럭 안에 선언하며, 선언된 영역 내부에서만 사용될 수 있다.
익명 클래스 (anonymous class)	클래스의 선언과 객체의 생성을 동시에 하는 이름없는 클래스(일회용)

▶ 내부 클래스의 선언, 제어자와 접근성: 각 내부 클레스의 선언 위치에 따라 같은 선언위치의 변수와 동일한 유효범위(scope)와 접근성(accessibility)를 갖는다. 내부 클래스도 클래스이기 때문에 abstract이나 final와 같은 제어자를 사용할 수 있고, private, protected과 접근제어자 사용이 가능하다. 인스턴스 클래스에서는 static 변수를 선언할 수 없고, static클래스에서만 static멤버를 정의할 수 있다. 다만 final static이 동시에 붙은 변수는 상수(constant)이므로 모든 내부 클래스에서 정의가 가능하다. 또한 인스턴스클래스는 외부 클래스의 인스턴스멤버를 객체생성 없이 바로 사용할 수 있지만, 스태틱 클래스는 외부클래스의 인스턴스 멤버를 객체생성 없이 사용할 수 없다. 마찬가지로 인스턴스클래스는 스태틱 클래스의 멤버들을 객체생성 없이 사용할 수 있지만, 스태틱 클래스에서는 인스턴스클래스 멤버들을 객체생성 없이 사용할 수 없다.

```
class InnerEx2 {
    class InstanceInner {}
    static class StaticInner {}

    // 인스턴스멤버 간에는 서로 직접 접근이 가능하다.
    InstanceInner iv = new InstanceInner();
    // static 멤버 간에는 서로 직접 접근이 가능하다.
    static StaticInner cv = new StaticInner();

    static void staticMethod() {
        // static멤버는 인스턴스멤버에 직접 접근할 수 없다.
//      InstanceInner obj1 = new InstanceInner();
        StaticInner obj2 = new StaticInner();

        //  굳이 접근하려면 아래와 같이 객체를 생성해야 한다.
        //  인스턴스클래스는 외부 클래스를 먼저 생성해야만 생성할 수 있다.
        InnerEx2 outer = new InnerEx2();
        InstanceInner obj1 = outer.new InstanceInner();
    }

    void instanceMethod() {
        // 인스턴스메서드에서는 인스턴스멤버와 static멤버 모두 접근 가능하다.
        InstanceInner obj1 = new InstanceInner();
        StaticInner obj2 = new StaticInner();
        // 메서드 내에 지역적으로 선언된 내부 클래스는 외부에서 접근할 수 없다.
//      LocalInner lv = new LocalInner();
    }

    void myMethod() {
        class LocalInner {}
        LocalInner lv = new LocalInner();
    }
}
```

지역 클래스는 외부 클래스의 인스턴스멤버와 스태틱멤버 모두를 사용할 수 있으며, 지역 클래스가 포함된 메서드에 정의된 지역변수도 사용할 수 있다. 단, final이 붙은 지역 변수만 접근 가능한데 그 이유는 메서드가 수행을 마쳐서 지역변수가 소멸된 시점에도 지역 클래스의 인스턴스가 소멸된 지역변수를 참조하려는 경우가 발생할 수 있기 때문이다 (JDK1.8부터 지역 클래스에서 접근하는 지역변수 앞에 final을 생략할 수 있기 바뀌었지만 대신 컴파일러가 자동으로 붙어준다. 하지만 해당 변수의 값이 바뀌는 문장이 있으면 여전히 컴파일 에러가 발생한다).

```
class InnerEx3 {
    private int outerIv = 0;
    static int outerCv = 0;

    class InstanceInner {
        int iiv  = outerIv;  // 외부 클래스의 private멤버도 접근가능하다.
        int iiv2 = outerCv;
    }

    static class StaticInner {
// 스태틱 클래스는 외부 클래스의 인스턴스멤버에 접근할 수 없다.
//      int siv = outerIv;
        static int scv = outerCv;
    }

    void myMethod() {
        int lv = 0;
        final int LV = 0;  // JDK1.8부터 final 생략 가능

        class LocalInner {
            int liv = outerIv;
            int liv2 = outerCv;
//  외부 클래스의 지역변수는 final이 붙은 변수(상수)만 접근가능하다.
//          int liv3 = lv;  // 에러!!!(JDK1.8부터 에러 아님)
            int liv4 = LV;  // OK
        }
    }
}
```

인스턴스 내부 클래스의 인스턴스를 생성하려면 먼저 외부 클래스의 인스턴스를 생성해야 하지만, 스태틱 내부 클래스의 인스턴스는 외부 클래스 생성 없이 생성 가능하다. 하지만 이건 외부 클래스가 아닌 다른 클래스에서 내부 클래스를 생성하고 내부 클래스의 멤버에 접근하는 것이고, 이것은 내부 클래스로 선언해서는 안 되는 클래스를 내부 클래스로 선언했다는 의미이니 가볍게 보고 넘어가면 된다.

```java
class Outer {
    class InstanceInner {
        int iv=100;
    }
    static class StaticInner {
        int iv=200;
        static int cv=300;
    }

    void myMethod() {
        class LocalInner {
            int iv=400;
        }
    }
}

class InnerEx4 {
    public static void main(String[] args) {
        // 인스턴스클래스의 인스턴스를 생성하려면
        // 외부 클래스의 인스턴스를 먼저 생성해야 한다.
        Outer oc = new Outer();
        Outer.InstanceInner ii = oc.new InstanceInner();

        System.out.println("ii.iv : "+ ii.iv);
        System.out.println("Outer.StaticInner.cv : " + Outer.StaticInner.cv);

         // 스태틱 내부 클래스의 인스턴스는 외부 클래스를 먼저 생성하지 않아도 된다.
        Outer.StaticInner si = new Outer.StaticInner();
        System.out.println("si.iv : "+ si.iv);
    }
}
```

```
InnerEx4.class
Outer.class
Outer$InstanceInner.class
Outer$StaticInner.class
Outer$1LocalInner.class
```

컴파일 시 생성되는 클래스 파일은 다음과 같다. 컴파일 했을 때 생성되는 파일 명은 '외부 클래스명$내부 클래스명.class'형식으로 되어 있다. 다만 지역내부 클래스는 다든 메서드에 같은 이름의 내부클래스가 존재할 수 있기 때문에, 내부 클래스명 앞에 숫자가 붙는다.

```java
class Otuer {
    void myMethod() {
        class LocalInner {}
    }

    void myMethod2() {
        class LocalInner{}
    }
}
```

```
Outer.class
Outer$1LocalInner.Class
Outer$2LocalInner.class
```

내부 클래스와 외부 클래스에 선언된 변수의 이름이 같을 때, 변수 앞에 'this' 또는 '외부 클래스명.this'를 붙여서 서로 구별할 수 있다.

```
class Outer {
    int value=10;   // Outer.this.value

    class Inner {
        int value=20;   // this.value

        void method1() {
            int value=30;
            System.out.println("           value :" + value);    // 30 지역변수
            System.out.println("      this.value :" + this.value);  // 20 내부클래스 변수
            System.out.println("Outer.this.value :" + Outer.this.value); // 10 외부클래스
        }
    } // Inner클래스의 끝
} // Outer클래스의 끝

class InnerEx5 {
    public static void main(String args[]) {
        Outer outer = new Outer();
        Outer.Inner inner = outer.new Inner();
        inner.method1();
    }
}
```

▶ 익명 클래스(anonymous class): 다른 내부 클래스들과는 달리 이름이 없고, 클래스의 선언과 객체의 생성을 동시에 하기 때문에 단 한번만 사용될 수 있고, 오직 하나의 객체만을 생성할 수 있는 일회용 클래스이다.

```
new 조상클래스이름() {
    // 멤버 선언
}
```

```
new 구현인터페이스이름() {
    // 멤버 선언
}
```

이름이 없기 때문에 생성자도 가질 수 없으며, 조상클래스의 이름이나 구현하고자 하는 인터페이스의 이름을 사요해서 정의하기 때문에, 하나의 클래스로 상속받는 동시에 인터페이스를 구현하구나 둘 이상의 인터페이스를 구현할 수 없다. 오로지 단 하나의 클래스를 상속받거나, 단 하나의 인터페이스만을 구현할 수 있다.

```
class InnerEx6 {
    Object iv = new Object(){ void method(){} };        // 익명클래스
    static Object cv = new Object(){ void method(){} }; // 익명클래스

    void myMethod() {
        Object lv = new Object(){ void method(){} };    // 익명클래스
    }
}
```

익명 클래스는 이름이 없기 때문에 컴파일 시 '외부 클래스명$숫자.class'의 형식으로 클래스파일명이 결정된다.

```
InnerEx6.class
InnerEx6$1.class
InnerEx6$2.class
InnerEx6$3.class
```

```
import java.awt.*;
import java.awt.event.*;

class InnerEx7 {
    public static void main(String[] args) {
        Button b = new Button("Start");
        b.addActionListener(new EventHandler());
    }
}

class EventHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("ActionEvent occurred!!!");
    }
}
```

위의 예제를 익명 클래스를 이용해서 변경. 이처럼 먼저 두 개의 독립된 클래스를 작성한 다음에, 다시 익명클래스를 이용해서 변경하면 보다 쉽게 코드를 작성할 수 있다.

```
import java.awt.*;
import java.awt.event.*;

class  InnerEx8 {
    public static void main(String[] args) {
        Button b = new Button("Start");
        b.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    System.out.println("ActionEvent occurred!!!");
                }
            } // 익명 클래스의 끝
        );
    } // main메서드의 끝
} // InnerEx8클래스의 끝
```

Resource: 자바의 정석