

■ 내부 클래스(inner class)

- 클래스 안에 선언된 클래스
- 특정 클래스 내에서만 주로 사용되는 클래스를 내부 클래스로 선언한다.

```
class OuterClass
{
    . . . .
    class InnerClass
    {
        . . . .
    }
}
```

왼쪽에서와 같이 클래스의 정의가 다른 클래스의 내부에 삽입될 수 있다. 이 때 외부의 클래스를 가리켜 **Outer 클래스**라 하고, 내부의 클래스를 가리켜 **Inner 클래스**라 한다.

■ 내부 클래스 분류

선언 위치에 따른 분류		선언 위치	설명
멤버 클래스	인스턴스 멤버 클래스	<pre>class A { class B { ... } }</pre>	A 객체를 생성해야만 사용할 수 있는 B 중첩 클래스
	정적 멤버 클래스	<pre>class A { static class B { ... } }</pre>	A 클래스로 바로 접근할 수 있는 B 중첩 클래스
로컬 클래스		<pre>class A { void method() { class B { ... } } }</pre>	method()가 실행할 때만 사용할 수 있는 B 중첩 클래스

■ 내부 클래스 사용 (1 / 2)

```
public class InnerClass {  
    public class NormalClass {  
        public void run() {  
            System.out.println("Normal 실행");  
        }  
    }  
  
    public static class StaticClass {  
        public void run() {  
            System.out.println("Static 실행");  
        }  
    }  
}
```

```
public void method() {  
    class LocalClass {  
        void run() {  
            System.out.println("Local 실행");  
        }  
    }  
    new LocalClass().run();  
}
```

■ 내부 클래스 사용 (2 / 2)

```
public class InnerClassMain {  
    public static void main(String[] args) {  
        InnerClass ic = new InnerClass();  
  
        InnerClass.NormalClass nc = ic.new NormalClass();  
        nc.run();  
  
        InnerClass.StaticClass sc = new InnerClass.StaticClass();  
        sc.run();  
  
        ic.method();  
    }  
}
```

■ 인스턴스 멤버 클래스

● static 키워드 사용 불가

```
class A {  
    /**인스턴스 멤버 클래스**/  
    class B {  
        B() {}                -----생성자  
        int field1;           -----인스턴스 필드  
        //static int field2;   -----정적 필드 (x)  
        void method1() {}     -----인스턴스 메소드  
        //static void method2() {} -----정적 메소드 (x)  
    }  
}
```

```
A  a = new A();  
A.B b = a.new B();  
b.field1 = 3;  
b.method1();
```

■ 정적 멤버 클래스

- static 키워드로 선언된 클래스, 모든 종류의 필드, 메소드 선언 가능

```
class A {  
    /**정적 멤버 클래스**/  
    static class C {  
        C() {}                -----생성자  
        int field1;           -----인스턴스 필드  
        static int field2;    -----정적 필드  
        void method1() {}    -----인스턴스 메소드  
        static void method2() {} -----정적 메소드  
    }  
}
```

```
A.C c = new A.C();  
c.field1 = 3;    //인스턴스 필드 사용  
c.method1();    //인스턴스 메소드 호출  
A.C.field2 = 3; //정적 필드 사용  
A.C.method2();  //정적 메소드 호출
```

■ 로컬 클래스

● 메소드 내에서만 사용가능

```
void method() {  
    /**로컬 클래스**/  
    class D {  
        D() {}                -----생성자  
        int field1;           -----인스턴스 필드  
        //static int field2;   -----정적 필드(x)  
        void method1() {}     -----인스턴스 메소드  
        //static void method2() {} -----정적 메소드(x)  
    }  
    D d = new D();  
    d.field1 = 3;  
    d.method1();  
}
```

```
void method() {  
    class DownloadThread extends Thread { ... }  
    DownloadThread thread = new DownloadThread();  
    thread.start();  
}
```

■ 내부 클래스 사용 - 외부 클래스 필드와 메소드에서 사용

```
public class A {  
    //인스턴스 멤버 클래스  
    class B {}  
  
    //정적 멤버 클래스  
    static class C {}  
}
```

● 인스턴스 멤버 사용

```
public class A {  
    //인스턴스 필드  
    B field1 = new B();          ----- (o)  
    C field2 = new C();          ----- (o)  
  
    //인스턴스 메소드  
    void method1() {  
        B var1 = new B();        ----- (o)  
        C var2 = new C();        ----- (o)  
    }  
}
```

● static 멤버 사용

```
//정적 필드 초기화  
//static B field3 = new B();      ----- (x)  
static C field4 = new C();        ----- (o)  
  
//정적 메소드  
static void method2() {  
    //B var1 = new B();            ----- (x)  
    C var2 = new C();              ----- (o)  
}
```


■ 내부 클래스 사용 - 내부 클래스 내에서 사용

```
class A {  
    int field1;  
    void method1() { ...}  
  
    static int field2;  
    static void method2() {...}  
  
    class B {  
        void method() {  
            field1 = 10;  
            method1();  
  
            field2 = 10;  
            method2();  
        }  
    }  
}
```

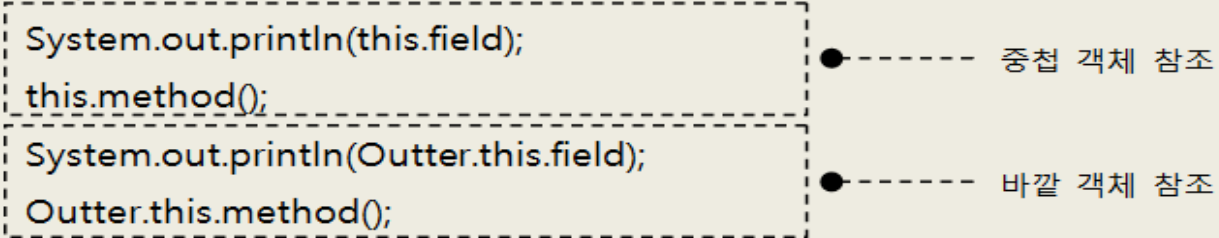
The diagram illustrates the valid use of an internal class. Class B, defined inside class A, has four lines of code. Each line has a circle at the end of the line, from which an arrow points to the corresponding member of class A. The arrows show that B can access A's instance field `field1`, A's instance method `method1()`, A's static field `field2`, and A's static method `method2()`.

```
class A {  
    int field1;  
    void method1() { ...}  
  
    static int field2;  
    static void method2() {...}  
  
    static class C {  
        void method() {  
            field1 = 10;  
            methodA1();  
  
            fieldA2 = 10;  
            methodA2();  
        }  
    }  
}
```

The diagram illustrates an invalid use of an internal class. Class C is a static class defined inside class A. It has four lines of code. The first two lines, `field1 = 10;` and `methodA1();`, have an 'X' at the end of the line, indicating an error. The last two lines, `fieldA2 = 10;` and `methodA2();`, have circles at the end of the line, from which arrows point to `field2` and `method2()` in class A. This shows that static class C cannot access A's instance members (`field1` and `methodA1()`) but can access its static members.

■ 내부 클래스 사용 - 내부 클래스 내에서 외부 클래스 참조 사용

```
public class Outer {  
    String field = "Outer-field";  
    void method() {  
        System.out.println("Outer-method");  
    }  
  
    class Nested {  
        String field = "Nested-field";  
        void method() {  
            System.out.println("Nested-method");  
        }  
        void print() {  
            System.out.println(this.field);  
            this.method();  
            System.out.println(Outer.this.field);  
            Outer.this.method();  
        }  
    }  
}
```



중첩 객체 참조

바깥 객체 참조

```

class OuterClass
{
    private String myName;
    private int num;
    OuterClass(String name)
    {
        myName=name;
        num=0;
    }
    public void whoAreYou()
    {
        num++;
        System.out.println(myName+ " OuterClass "+num);
    }
}

```

```

class InnerClass
{
    InnerClass()
    {
        whoAreYou();
    }
}

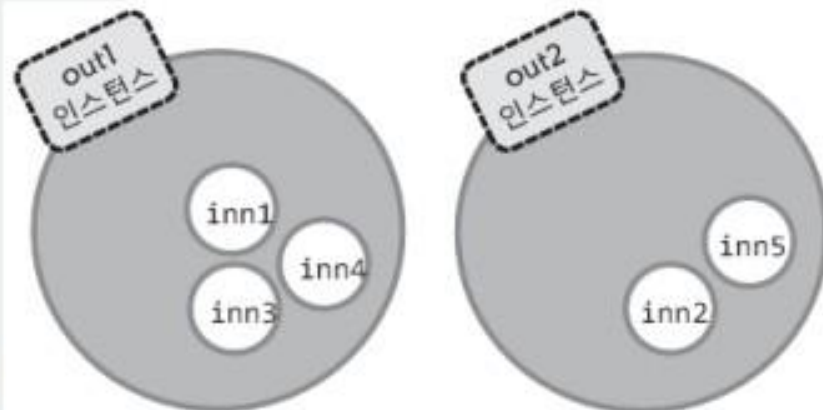
```

**Inner 클래스의 인스턴스는
Outer 클래스의 인스턴스에 종속적이다!**

```

public static void main(String[] args)
{
    OuterClass out1=new OuterClass("First");
    OuterClass out2=new OuterClass("Second");
    out1.whoAreYou();
    out2.whoAreYou();
    OuterClass.InnerClass inn1=out1.new InnerClass();
    OuterClass.InnerClass inn2=out2.new InnerClass();
    OuterClass.InnerClass inn3=out1.new InnerClass();
    OuterClass.InnerClass inn4=out1.new InnerClass();
    OuterClass.InnerClass inn5=out2.new InnerClass();
}

```



- Outer 클래스의 인스턴스 생성 후에야 Inner 클래스의 인스턴스 생성이 가능하다.
- Inner 클래스 내에서는 Outer 클래스의 멤버에 직접 접근이 가능하다.
- Inner 클래스의 인스턴스는 자신이 속할 Outer 클래스의 인스턴스를 기반으로 생성된다.

**Inner 클래스의
성격**

■ 익명 클래스 (Anonymous Class)

- 이름이 없는 클래스 (변수명이 없는 것이 아니라 구현 클래스명이 없음)
- 익명 객체는 단독 생성 불가
 - 클래스 상속하거나 인터페이스 구현해야만 생성 가능
- 사용 위치
 - 필드의 초기값, 로컬 변수의 초기값, 매개변수의 매개값으로 주로 대입
 - UI 이벤트 처리 객체나, 스레드 객체를 간편하게 생성할 목적으로 주로 활용
- 익명 클래스 생성

```
class A {  
    Parent field = new Parent() {  
        int childField;  
        void childMethod() { }  
        @Override  
        void parentMethod() { }  
    };  
}
```

A 클래스의 필드 선언

Parent 의 메소드를 오버라이딩

■ 익명 클래스 내부 요소 사용

- 새롭게 정의된 변수와 메소드는 익명 클래스 내부에서만 사용 가능
- 외부에서는 익명 클래스의 재정의 메소드만 사용가능
 - 익명 클래스는 부모 타입으로 선언되므로
부모 타입에 선언되어 있는 요소만 사용 가능

```
class A {  
    Parent field = new Parent() {  
        int childField;  ←-----  
        void childMethod() { } ←-----  
        @Override  
        void parentMethod() { ←-----  
            childField = 3;  
            childMethod();  
        }  
    };  
  
    void method() {  
        field.childField = 3; -----  
        field.childMethod(); -----  
        field.parentMethod(); -----  
    }  
}
```

■ 익명 클래스 – 버튼 이벤트 (1 / 2)

```
public interface OnClickListener {  
    void onClick();  
}
```

```
public class Button {  
    public void setOnClickListener(OnClickListener onClickListener) {  
        onClickListener.onClick();  
    }  
}
```

```
public class MyClickListener implements OnClickListener {  
    @Override  
    public void onClick() {  
        System.out.println("버튼 클릭 - 전화걸기");  
    }  
}
```

■ 익명 클래스 – 버튼 이벤트 (2 / 2)

```
public class Main {  
    public static void main(String[] args) {  
        Button btn = new Button();  
        MyClickListener mcl = new MyClickListener();  
        btn.setOnClickListener(mcl);  
  
        btn.setOnClickListener(new OnClickListener() {  
            @Override  
            public void onClick() {  
                System.out.println("버튼 클릭 - 인터넷 연결");  
            }  
        });  
    }  
}
```