


## 09장 중첩 클래스와 중첩 인터페이스

### 9.1 중첩 클래스와 중첩 인터페이스란?

- 중첩 클래스(Nested Class): 클래스 멤버로 선언된 클래스
- 중첩 인터페이스: 클래스 멤버로 선언된 인터페이스 -> UI 컴포넌트 내부 이벤트 처리에 많이 활용 

```
class ClassName {
    class NestedClassName {}
}

class ClassName {
    interface NestedInterfaceName {}
}

// View 클래스의 클릭 이벤트를 처리하는 구현 클래스의 예
public class View {
    public interface OnClickListener {
        public void onClick(View v);
    }
}
```

### 9.2 중첩 클래스

- 클래스의 멤버로서 선언되는 중첩 클래스를 멤버 클래스라고 하고, 메소드 내부에서 선언되는 중첩 클래스를 로컬 클래스라고 한다.

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

#### 9.2.1. 인스턴스 멤버 클래스

- 인스턴스 멤버 클래스는 인스턴스 필드와 메소드만 선언이 가능하고 정적 필드와 메소드는 선언할 수 없다.

```
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();
```

### 9.2.2. 정적 멤버 클래스

- 정적 멤버 클래스는 모든 종류의 필드와 메소드를 선언할 수 있다.

```
class A {
    //정적 멤버 클래스
    static class C {
        C() {} //생성자
        int field1; //인스턴스 필드
        static int field2; //정적 필드 (0)
        void method1() {} //인스턴스 메소드
        static void method2() {} //정적 메소드 (0)
    }
}

A.C c = new A.C();
c.field1 = 3;
c.method1();
A.C.field2 = 3;
A.C.method2();
```

### 9.2.3. 로컬 클래스

- 로컬 클래스는 접근 제한자 및 static을 붙일 수 없다. 로컬 클래스는 메소드 내부에서만 사용되므로 접근을 제한할 필요가 없기 때문이다.

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

```
d.method1();
}
```

## 9.3 중첩 클래스의 접근 제한

### 9.3.1 바깥 필드와 메서드에서 사용 제한

- 정적 필드(field3)의 초기값이나 정적 메서드(method2())에서는 객체를 생성할 수 없다.

[A.java] 바깥 필드와 메서드에서 사용 제한

```
package sec03.exam01_member_class_access;

public class A {
    // 인스턴스 필드
    B field1 = new B();
    C field2 = new C();

    // 인스턴스 메서드
    void method1() {
        B var1 = new B();
        C var2 = new C();
    }

    // 정적 필드 초기화
    // static B field3 = new B(); // (x)
    static C field4 = new C();

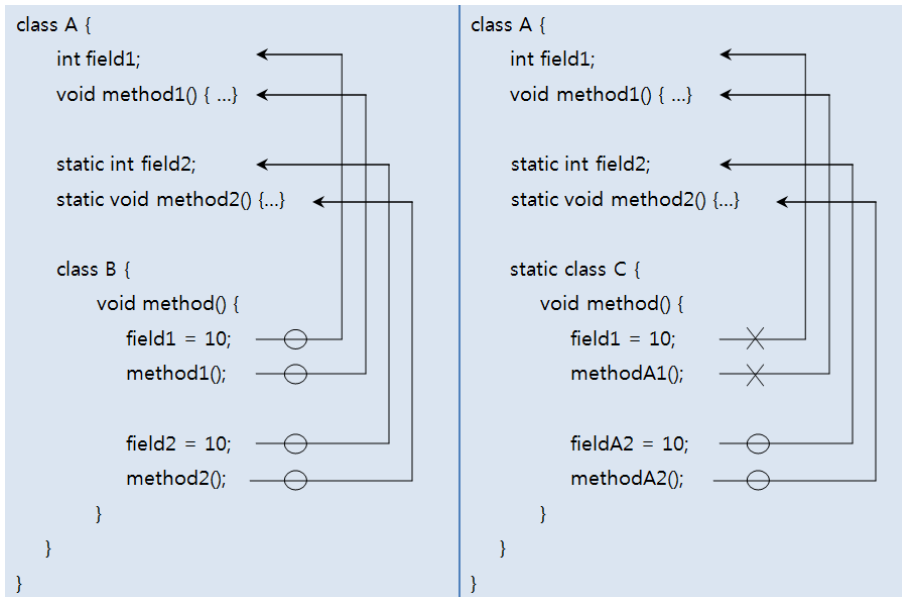
    // 정적 메서드
    static void method2() {
        // B var1 = new B(); // (x)
        C var2 = new C();
    }

    // 인스턴스 멤버 클래스
    class B {
    }

    // 정적 멤버 클래스
    static class C {
    }
}
```

### 9.3.2 멤버 클래스에서 사용 제한

- 인스턴스 멤버 클래스(B) 안에서는 바깥 클래스의 모든 필드와 모든 메서드에 접근할 수 있지만, 정적 멤버 클래스(C) 안에서는 바깥 클래스의 정적 필드(field2)와 메서드(method2())에만 접근할 수 있고 인스턴스 필드(field1)와 메서드(method1())는 접근할 수 없다.



### 9.3.3 로컬 클래스에서 사용 제한

- 로컬 클래스에서 사용된 매개 변수와 로컬 변수는 모두 `final` 특성을 갖는다. 자바7까지는 반드시 `final` 키워드를 붙여야 되지만, 자바8부터는 붙이지 않아도 `final` 특성을 가지고 있음을 주목해야 한다.

```

public class Outer {
    //자바7 이전
    public void method1(final int arg) {
        final int localVariable = 1;
        //arg = 100; (x)
        //localVariable = 100; (x)
        class Inner {
            public void method() {
                int result = arg + localVariable;
            }
        }
    }
}
    
```

final 매개변수와 로컬 변수는  
로컬 클래스의 메소드의 로컬 변수로 복사  
(final 붙이지 않으면 컴파일 오류 발생)

```

//자바8 이후
public void method2(int arg) {
    int localVariable = 1;
    //arg = 100; (x)
    //localVariable = 100; (x)
    class Inner {
        public void method() {
            int result = arg + localVariable;
        }
    }
}
    
```

매개변수와 로컬 변수는 final 특성을 가지며,  
로컬 클래스의 필드로

[Outer.java]

```

01 package sec03.exam03_localclass_access;
02
03 public class Outer {
04     //자바7 이전
05     public void method1(final int arg) {
06         final int localVariable = 1;
07         //arg = 100; (x)
08         //localVariable = 100; (x)
09         class Inner {
10             public void method() {
    
```

```

11             int result = arg + localVarible;
12         }
13     }
14 }
15
16 //자바8 이후
17 public void method2(int arg) {
18     int localVarible = 1;
19     //arg = 100; (x) //final 변수로 변경이 불가하다.
20     //localVariable = 100; (x) // final 변수로 변경이 불가하다.
21     class Inner {
22         public void method() {
23             int result = arg + localVarible;
24         }
25     }
26 }
27 }

```

### 9.3.4 중첩 클래스에서 바깥 클래스 참조 얻기

- 중첩 클래스 내부에서 this.필드, this.메소드()로 호출하면 중첩 클래스의 필드와 메소드가 사용된다. 중첩 클래스 내부에서 바깥 클래스의 객체 참조를 얻으려면 바깥 클래스의 이름을 this 앞에 붙여주면 된다.

```

바깥클래스.this.필드
바깥클래스.this.메소드();

```

```

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();
        }
    }
}

```

## 9.4 중첩 인터페이스

- 중첩 인터페이스는 클래스의 멤버로 선언된 인터페이스를 말한다. 인터페이스를 클래스 내부에 선언하는 이유는 해당 클래스와 긴밀한 관계를 맺는 구현 클래스를 만들기 위해서이다. 특히 UI 프로그래밍에서 이벤트를 처리할 목적으로 많이 활용된다.

```
[Button.java]

package sec04.exam01_nestedinterface;

public class Button {
    OnClickListener listener; // 인터페이스 타입 필드

    void setOnClickListener(OnClickListener listener) { // 매개 변수의 다형성
        this.listener = listener;
    }

    void touch() {
        listener.onClick(); // 구현 객체의 onClick() 메소드 호출
    }

    interface OnClickListener { // 중첩 인터페이스
        void onClick();
    }
}
```

## 9.5 익명 객체

- 익명(anonymous) 객체는 이름이 없는 객체를 말한다.
  - 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

### 9.5.1 익명 자식 객체 생성

(1) 부모 타입으로 필드나 변수를 선언하고 자식 객체를 초기값으로 대입할 경우

- 자식 클래스가 재사용되지 않고, 오로지 해당 필드와 변수의 초기값으로만 사용할 경우라면 익명 자식 객체를 생성해서 초기값으로 대입하는 것이 좋은 방법이다.

■ 주의할 점은 하나의 실행문이므로 끝에는 세미콜론(;)을 반드시 붙여야 한다.

■ 일반 클래스와의 차이점은 생성자를 선언할 수 없다는 것이다.

```
// 기본형
부모클래스 [필드|변수] = new 부모클래스(매개값,...) {
    // 필드
    // 메소드
};

// 기본적인 방법
class Child extends Parent {} // 자식 클래스 선언
class A {
```

```

Parent field = new Child(); // 필드에 자식 객체를 대입, 다형성(객체의 부품화)을 위함.
void method() {
    Parent localVar = new Child(); // 로컬 변수에 자식 객체를 대입
}
}

// 익명 자식 객체 생성해서 초기값으로 대입하는 방법
class A {
    Parent field = new Parent() { // A 클래스의 필드 초기값으로 익명 자식 객체를 생성해서 대입한다.
        int childField;
        void childMethod() {}
        @Override
        void parentMethod() {}
    };
}

```

## (2) 익명 객체에 새롭게 정의된 필드와 메소드

- 익명 객체 내부에서만 사용
- 외부에서는 익명 객체의 필드와 메소드에 접근할 수 없음 --> 이유: 익명 객체는 부모 타입 변수에 대입되므로 부모 타입에 선언된 것만 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();
    }
}

```

### 9.5.2 익명 구현 객체 생성

- 구현 클래스가 재사용되지 않고, 오로지 해당 필드와 변수의 초기값으로만 사용하는 경우라

면 익명 구현 객체를 초기값으로 대입하는 것이 좋다.

```
// 기본형
인터페이스 [필드|변수] = new 인터페이스() {
    // 인터페이스에 선언된 추상 메소드의 실제 메소드 선언
    // 필드
    // 메소드
};

// 일반적인 구현 객체를 초기값으로 대입하는 경우
class TV implements RemoteControl { }
class A {
    RemoteControl field = new TV(); // 필드에 구현 객체를 대입
    void method() {
        RemoteControl localVar = new TV(); // 로컬 변수에 구현 객체를 대입
    }
}

// 익명 구현 객체를 초기값으로 대입하는 경우
class A {
    RemoteControl field = new RemoteControl() { // 필드 선언
        @Override
        void turnOn() {}
    };
    void method() {
        RemoteControl localVar = new RemoteControl() { // 로컬 변수 선언
            @Override
            void turnOn() {}
        };
    }
    void method1(RemoteControl rc) {}
    void method2() {
        method1(
            new RemoteControl() { // method1()의 매개값으로 익명 구현 객체를 대입
                @Override
                void turnOn() {}
            }
        );
    }
}
```

### 9.5.3 익명 객체의 로컬 변수 사용

- 익명 객체 내부에서 메소드의 매개 변수나 로컬 변수를 사용할 경우, 이 변수들은 final 특성을 가져야 한다.

[Anonymous.java] 익명 객체의 로컬 변수 사용

```
package sec05.exam04_local_variable_access;

public class Anonymous {
    private int field;

    public void method(final int arg1, int arg2) {
        final int var1 = 0;
        int var2 = 0;

        field = 10;
    }
}
```



```
// arg1 = 20; (x)
// arg2 = 20; (x)

// var1 = 30; (x)
// var2 = 30; (x)

Calculatable calc = new Calculatable() {
    @Override
    public int sum() {
        int result = field + arg1 + arg2 + var1 + var2;
        return result;
    }
};

System.out.println(calc.sum());
}
```

## [과제] 확인문제

1. 중첩 멤버 클래스에 대한 설명으로 틀린 것은 무엇입니까?

- (1) 인스턴스 멤버 클래스는 바깥 클래스의 객체가 있어야 사용될 수 있다.
- (2) 정적 멤버 클래스는 바깥 클래스의 객체가 없어도 사용될 수 있다.
- (3) 인스턴스 멤버 클래스 내부에는 바깥 클래스의 모든 필드와 메소드를 사용할 수 있다.
- (4) 정적 멤버 클래스 내부에는 바깥 클래스의 인스턴스 필드를 사용할 수 있다.

2. 로컬 클래스에 대한 설명으로 틀린 것은 무엇입니까?

- (1) 로컬 클래스는 메소드 내부에 선언된 클래스를 말한다.
- (2) 로컬 클래스는 바깥 클래스의 모든 필드와 메소드를 사용할 수 있다.
- (3) 로컬 클래스는 static 키워드를 이용해서 정적 클래스로 만들 수 있다.
- (4) final 특성을 가진 매개 변수나 로컬 변수만 로컬 클래스 내부에서 사용할 수 있다.

3. 익명 객체에 대한 설명으로 틀린 것은 무엇입니까?

- (1) 익명 객체는 클래스를 상속하거나 인터페이스를 구현해야만 생성될 수 있다.
- (2) 익명 객체는 필드, 매개 변수, 로컬 변수의 초기값으로 주로 사용된다.
- (3) 익명 객체에는 생성자를 선언할 수 있다.
- (4) 부모 클래스나 인터페이스에 선언된 필드와 메소드 이외에 다른 필드와 메소드를 선언할 수 있지만, 익명 객체 내부에서만 사용이 가능하다.

4. 다음과 같이 Car 클래스 내부에 Tire와 Engine이 멤버 클래스로 선언되어 있습니다. NestedClassExample에서 멤버 클래스의 객체를 생성하는 코드를 괄호안에 작성해보세요.

```
[Car.java]

01 package verify.exam04;
02
03 public class Car {
04     class Tire {}
```

```
05         static class Engine {}  
06     }
```

[NestedClassExample.java]

```
01     package verify.exam04;  
02  
03     public class NestedClassExample {  
04         public static void main(String[] args) {  
05             Car myCar = new Car();  
06  
07             Car.Tire tire = (        #1        )  
08  
09             Car.Engine engine = (        #2        )  
10         }  
11     }
```