

2025년 상반기 K-디지털 트레이닝

중첩 선언과 익명 객체

[KB] IT's Your Life

1 중첩 클래스

• 중첩 클래스

- 클래스 내부에 선언한 클래스. 클래스의 멤버를 쉽게 사용할 수 있고 외부에는 중첩 관계 클래스를 감춤으로써 코드의 복잡성을 줄일 수 있음
- 멤버 클래스: 클래스의 멤버로서 선언되는 중첩 클래스
- 로컬 클래스: 메소드 내부에서 선언되는 중첩 클래스

선언 위치에 따른 분류		선언 위치	객체 생성 조건
멤버 클래스	인스턴스 멤버 클래스	<pre>class A { class B { ... } }</pre>	A 객체를 생성해야만 B 객체를 생성할 수 있음
	정적 멤버 클래스	<pre>class A { static class B { ... } }</pre>	A 객체를 생성하지 않아도 B 객체를 생성할 수 있음
로컬 클래스		<pre>class A { void method() { class B { ... } } }</pre>	method가 실행할 때만 B 객체를 생성할 수 있음

A \$ B .class
바깥 클래스 멤버 클래스

A \$1 B .class
바깥 클래스 로컬 클래스

2 인스턴스 멤버 클래스

• 인스턴스 멤버 클래스

- A 클래스의 멤버로 선언된 B 클래스

```
[public] class A {
    [public | private] class B {
    }
}
```

인스턴스 멤버 클래스

구분	접근 범위
public class B { }	다른 패키지에서 B 클래스를 사용할 수 있다.
class B { }	같은 패키지에서만 B 클래스를 사용할 수 있다.
private class B { }	A 클래스 내부에서만 B 클래스를 사용할 수 있다.

- 인스턴스 멤버 클래스 B는 주로 A 클래스 내부에서 사용되므로 **private** 접근 제한을 갖는 것이 일반적

- **A.java**

```
package ch09.sec02.exam01;

public class A {
    //인스턴스 멤버 클래스
    class B {}

    //인스턴스 필드 값으로 B 객체 대입
    B field = new B();

    //생성자
    A() {
        B b = new B();
    }

    //인스턴스 메소드
    void method() {
        B b = new B();
    }
}
```

- **AExample.java**

```
package ch09.sec02.exam01;

public class AExample {
    public static void main(String[] args) {
        //A 객체 생성
        A a = new A();

        //B 객체 생성
        A.B b = a.new B();
    }
}
```

● A.java

```
package ch09.sec02.exam02;

public class A {
    //인스턴스 멤버 클래스
    class B {
        //인스턴스 필드
        int field1 = 1;

        //정적 필드 (Java 17부터 허용)
        static int field2 = 2;

        //생성자
        B() {
            System.out.println("B-생성자 실행");
        }

        //인스턴스 메소드
        void method1() {
            System.out.println("B-method1 실행");
        }

        //정적 메소드 (Java 17부터 허용)
        static void method2() {
            System.out.println("B-method2 실행");
        }
    }

    //인스턴스 메소드
    void useB() {
        //B 객체 생성 및 인스턴스 필드 및 메소드 사용
        B b = new B();
        System.out.println(b.field1);
        b.method1();

        //B 클래스의 정적 필드 및 메소드 사용
        System.out.println(B.field2);
        B.method2();
    }
}
```

- **AExample.java**

```
package ch09.sec02.exam02;

public class AExample {
    public static void main(String[] args) {
        //A 객체 생성
        A a = new A();

        //A 인스턴스 메소드 호출
        a.useB();
    }
}
```

B-생성자 실행
1
B-method1 실행
2
B-method2 실행

3 정적 멤버 클래스

- 정적 멤버 클래스

- `static` 키워드와 함께 A 클래스의 멤버로 선언된 B 클래스

```
[public] class A {
    [public | private] static class B {
    }
}
```

정적 멤버 클래스

구분	접근 범위
<code>public static class B { }</code>	다른 패키지에서 B 클래스를 사용할 수 있다.
<code>static class B { }</code>	같은 패키지에서만 B 클래스를 사용할 수 있다.
<code>private static class B { }</code>	A 클래스 내부에서만 B 클래스를 사용할 수 있다.

- 정적 멤버 클래스는 주로 `default` 또는 `public` 접근 제한을 가진다.

- A.java

```
package ch09.sec03.exam01;

public class A {
    //정적 멤버 클래스
    static class B {}

    //인스턴스 필드 값으로 B 객체 대입
    B field1 = new B();

    //정적 필드 값으로 B 객체 대입
    static B field2 = new B();

    //생성자
    A() {
        B b = new B();
    }

    //인스턴스 메소드
    void method1() {
        B b = new B();
    }

    //정적 메소드
    static void method2() {
        B b = new B();
    }
}
```

- **AExample.java**

```
package ch09.sec03.exam01;

public class AExample {
    public static void main(String[] args) {
        //B 객체 생성
        A.B b = new A.B();
    }
}
```

3 정적 멤버 클래스

• A.java

```
package ch09.sec03.exam02;

public class A {
    //정적 멤버 클래스
    static class B {
        //인스턴스 필드
        int field1 = 1;

        //정적 필드 (Java 17부터 허용)
        static int field2 = 2;

        //생성자
        B() {
            System.out.println("B-생성자 실행");
        }

        //인스턴스 메소드
        void method1() {
            System.out.println("B-method1 실행");
        }

        //정적 메소드 (Java 17부터 허용)
        static void method2() {
            System.out.println("B-method2 실행");
        }
    }
}
```

- AExample.java

```
package ch09.sec03.exam02;

public class AExample {
    public static void main(String[] args) {
        //B 객체 생성 및 인스턴스 필드 및 메소드 사용
        A.B b = new A.B();
        System.out.println(b.field1);
        b.method1();

        //B 클래스의 정적 필드 및 메소드 사용
        System.out.println(A.B.field2);
        A.B.method2();
    }
}
```

B-생성자 실행

1

B-method1 실행

2

B-method2 실행

- 로컬 클래스

- 생성자 또는 메소드 내부에서 다음과 같이 선언된 클래스
- 생성자와 메소드가 실행될 동안에만 객체를 생성할 수 있음

```
[public] class A {
```

```
//생성자
```

```
public A() {
```

```
    class B { }
```

```
}
```

```
//메소드
```

```
public void method() {
```

```
    class B { }
```

```
}
```

```
}
```

로컬 클래스

- **A.java**

```
package ch09.sec04.exam01;
```

```
public class A {
```

```
    //생성자
```

```
    A() {
```

```
        //로컬 클래스 선언
```

```
        class B { }
```

```
        //로컬 객체 생성
```

```
        B b = new B();
```

```
    }
```

```
    //메소드
```

```
    void method() {
```

```
        //로컬 클래스 선언
```

```
        class B { }
```

```
        //로컬 객체 생성
```

```
        B b = new B();
```

```
    }
```

```
}
```

● A.java

```

package ch09.sec04.exam02;
public class A {
    //메소드
    void useB() {
        //로컬 클래스
        class B {
            //인스턴스 필드
            int field1 = 1;

            //정적 필드 (Java 17부터 허용)
            static int field2 = 2;

            //생성자
            B() {
                System.out.println("B-생성자 실행");
            }

            //인스턴스 메소드
            void method1() {
                System.out.println("B-method1 실행");
            }

            //정적 메소드 (Java 17부터 허용)
            static void method2() {
                System.out.println("B-method2 실행");
            }
        }

        //로컬 객체 생성
        B b = new B();

        //로컬 객체의 인스턴스 필드와 메소드 사용
        System.out.println(b.field1);
        b.method1();

        //로컬 클래스의 정적 필드와 메소드 사용
        // (Java 17부터 허용)
        System.out.println(B.field2);
        B.method2();
    }
}

```

- **AExample.java**

```
package ch09.sec04.exam02;

public class AExample {
    public static void main(String[] args) {
        //A 객체 생성
        A a = new A();

        //A 메소드 호출
        a.useB();
    }
}
```

B-생성자 실행

1

B-method1 실행

2

B-method2 실행

• A.java

```
package ch09.sec04.exam03;

public class A {
    //메소드
    public void method1(int arg) {          //final int arg
        //로컬 변수
        int var = 1;                        //final int var = 1;

        //로컬 클래스
        class B {
            //메소드
            void method2() {
                //로컬 변수 읽기
                System.out.println("arg: " + arg);    //(o)
                System.out.println("var: " + var);    //(o)

                //로컬 변수 수정
                //arg = 2;                            //(x)
                //var = 2;                            //(x)
            }
        }

        //로컬 객체 생성
        B b = new B();
        //로컬 객체 메소드 호출
        b.method2();

        //로컬 변수 수정
        //arg = 3;                                    //(x)
        //var = 3;                                    //(x)
    }
}
```

- 바깥 클래스의 멤버 접근 제한

- 정적 멤버 클래스 내부에서는 바깥 클래스의 필드와 메소드를 사용할 때 제한이 따름

구분	바깥 클래스의 사용 가능한 멤버
인스턴스 멤버 클래스	바깥 클래스의 모든 필드와 메소드
정적 멤버 클래스	바깥 클래스의 정적 필드와 정적 메소드

- 정적 멤버 클래스는 바깥 객체가 없어도 사용 가능해야 하므로 바깥 클래스의 인스턴스 필드와 인스턴스 메소드는 사용하지 못함

● A.java

```
package ch09.sec05.exam01;

public class A {
    //A의 인스턴스 필드와 메소드
    int field1;
    void method1() {}

    //A의 정적 필드와 메소드
    static int field2;
    static void method2() {}

    //인스턴스 멤버 클래스
    class B {
        void method() {
            //A의 인스턴스 필드와 메소드 사용
            field1 = 10;           //(o)
            method1();             //(o)
            //A의 정적 필드와 메소드 사용
            field2 = 10;           //(o)
            method2();             //(o)
        }
    }

    //정적 멤버 클래스
    static class C {
        void method() {
            //A의 인스턴스 필드와 메소드 사용
            //field1 = 10;           //(x)
            //method1();             //(x)
            //A의 정적 필드와 메소드 사용
            field2 = 10;           //(o)
            method2();             //(o)
        }
    }
}
```

- 바깥 클래스의 객체 접근

- 중첩 클래스 내부에서 바깥 클래스의 객체를 얻으려면 바깥 클래스 이름에 **this**를 붙임

바깥클래스이름.this → 바깥객체

● A.java

```
package ch09.sec05.exam02;

public class A {
    //A 인스턴스 필드
    String field = "A-field";

    //A 인스턴스 메소드
    void method() {
        System.out.println("A-method");
    }

    //인스턴스 멤버 클래스
    class B {
        //B 인스턴스 필드
        String field = "B-field";

        //B 인스턴스 메소드
        void method() {
            System.out.println("B-method");
        }

        //B 인스턴스 메소드
        void print() {
            //B 객체의 필드와 메소드 사용
            System.out.println(this.field);
            this.method();

            //A 객체의 필드와 메소드 사용
            System.out.println(A.this.field);
            A.this.method();
        }
    }

    //A의 인스턴스 메소드
    void useB() {
        B b = new B();
        b.print();
    }
}
```

- **AExample.java**

```
package ch09.sec05.exam02;

public class AExample {
    public static void main(String[] args) {
        //A 객체 생성
        A a = new A();

        //A 메소드 호출
        a.useB();
    }
}
```

B-field
B-method
A-field
A-method

- 중첩 인터페이스

- 해당 클래스와 긴밀한 관계를 맺는 구현 객체를 만들기 위해 클래스의 멤버로 선언된 인터페이스

```
class A {  
    [public | private] [static] interface B {  
        //상수 필드  
        //추상 메소드  
        //디폴트 메소드  
        //정적 메소드  
    }  
}
```

중첩 인터페이스

- 안드로이드와 같은 **UI** 프로그램에서 이벤트를 처리할 목적으로 많이 활용

- **Button.java**

```
package ch09.sec06.exam01;

public class Button {
    //정적 멤버 인터페이스
    public static interface ClickListener {
        //추상 메소드
        void onClick();
    }
}
```


- **Button.java**

```
package ch09.sec06.exam02;

public class Button {
    //정적 멤버 인터페이스
    public static interface ClickListener {
        //추상 메소드
        void onClick();
    }

    //필드
    private ClickListener clickListener;

    //메소드
    public void setClickListener(ClickListener clickListener) {
        this.clickListener = clickListener;
    }
}
```

- **Button.java**

```
package ch09.sec06.exam03;

public class Button {
    //정적 멤버 인터페이스
    public static interface ClickListener {
        //추상 메소드
        void onClick();
    }

    //필드
    private ClickListener clickListener;

    //메소드
    public void setClickListener(ClickListener clickListener) {
        this.clickListener = clickListener;
    }

    public void click() {
        this.clickListener.onClick();
    }
}
```

- Button.java

```
package ch09.sec06.exam03;

public class ButtonExample {
    public static void main(String[] args) {
        //Ok 버튼 객체 생성
        Button btnOk = new Button();

        //Ok 버튼 클릭 이벤트를 처리할 ClickListener 구현 클래스(로컬 클래스)
        class OkListener implements Button.ClickListener {
            @Override
            public void onClick() {
                System.out.println("Ok 버튼을 클릭했습니다.");
            }
        }

        //Ok 버튼 객체에 ClickListener 구현 객체 주입
        btnOk.setClickListener(new OkListener());

        //Ok 버튼 클릭하기
        btnOk.click();
    }
}
```


- Button.java

```
//Cancel 버튼 객체 생성
Button btnCancel = new Button();

//Cancel 버튼 클릭 이벤트를 처리할 ClickListener 구현 클래스(로컬 클래스)
class CancelListener implements Button.ClickListener {
    @Override
    public void onClick() {
        System.out.println("Cancel 버튼을 클릭했습니다.");
    }
}

//Cancel 버튼 객체에 ClickListener 구현 객체 주입
btnCancel.setClickListener(new CancelListener());

//Cancel 버튼 클릭하기
btnCancel.click();
}
```



7 익명 객체

- 익명 객체

- 이름이 없는 객체. 명시적으로 클래스를 선언하지 않기 때문에 쉽게 객체를 생성할 수 있음
- 필드값, 로컬 변수값, 매개변수값으로 주로 사용

- 익명 자식 객체

- 부모 클래스를 상속받아 생성되는 객체
- 부모 타입의 필드, 로컬 변수, 매개변수의 값으로 대입할 수 있음

```
new 부모생성자(매개값, ...) {  
    //필드  
    //메소드  
}
```

- **Tire.java**

```
package ch09.sec07.exam01;

public class Tire {
    public void roll() {
        System.out.println("일반 타이어가 굴러갑니다.");
    }
}
```

7 익명 객체

• Car.java

```
package ch09.sec07.exam01;

public class Car {
    //필드에 Tire 객체 대입
    private Tire tire1 = new Tire();

    //필드에 익명 자식 객체 대입
    private Tire tire2 = new Tire() {
        @Override
        public void roll() {
            System.out.println("익명 자식 Tire 객체 1이 굴러갑니다.");
        }
    };

    //메소드(필드 이용)
    public void run1() {
        tire1.roll();
        tire2.roll();
    }
}
```

- Car.java

```
//메소드(로컬 변수 이용)
public void run2() {
    //로컬 변수에 익명 자식 객체 대입
    Tire tire = new Tire() {
        @Override
        public void roll() {
            System.out.println("익명 자식 Tire 객체 2가 굴러갑니다.");
        }
    };
    tire.roll();
}

//메소드(매개변수 이용)
public void run3(Tire tire) {
    tire.roll();
}
}
```


- **CarExample.java**

```
package ch09.sec07.exam01;

public class CarExample {
    public static void main(String[] args) {
        //Car 객체 생성
        Car car = new Car();

        //익명 자식 객체가 대입된 필드 사용
        car.run1();

        //익명 자식 객체가 대입된 로컬변수 사용
        car.run2();

        //익명 자식 객체가 대입된 매개변수 사용
        car.run3(new Tire() {
            @Override
            public void roll() {
                System.out.println("익명 자식 Tire 객체 30이 굴러갑니다.");
            }
        });
    }
}
```

7 익명 객체

- 익명 구현 객체

- 인터페이스를 구현해서 생성되는 객체
- 인터페이스 타입의 필드, 로컬변수, 매개변수의 값으로 대입할 수 있음
- 안드로이드와 같은 **UI** 프로그램에서 이벤트를 처리하는 객체로 많이 사용

```
new 인터페이스() {  
    //필드  
    //메소드  
}
```

- **RemoteControl.java**

```
package ch09.sec07.exam02;

public interface RemoteControl {
    //추상 메소드
    void turnOn();
    void turnOff();
}
```

- Home.java

```
package ch09.sec07.exam02;

public class Home {
    //필드에 익명 구현 객체 대입
    private RemoteControl rc = new RemoteControl() {
        @Override
        public void turnOn() {
            System.out.println("TV를 켭니다.");
        }
        @Override
        public void turnOff() {
            System.out.println("TV를 끕니다.");
        }
    };

    //메소드(필드 이용)
    public void use1() {
        rc.turnOn();
        rc.turnOff();
    }
}
```

- Home.java

//메소드(로컬 변수 이용)

```
public void use2() {  
    //로컬 변수에 익명 구현 객체 대입  
    RemoteControl rc = new RemoteControl() {  
        @Override  
        public void turnOn() {  
            System.out.println("에어컨을 켭니다.");  
        }  
        @Override  
        public void turnOff() {  
            System.out.println("에어컨을 끕니다.");  
        }  
    };  
    rc.turnOn();  
    rc.turnOff();  
}
```

//메소드(매개변수 이용)

```
public void use3(RemoteControl rc) {  
    rc.turnOn();  
}
```

• Home.java

```
package ch09.sec07.exam02;

public class HomeExample {
    public static void main(String[] args) {
        //Home 객체 생성
        Home home = new Home();

        //익명 구현 객체가 대입된 필드 사용
        home.use1();

        //익명 구현 객체가 대입된 로컬 변수 사용
        home.use2();

        //익명 구현 객체가 대입된 매개변수 사용
        home.use3(new RemoteControl() {
            @Override
            public void turnOn() {
                System.out.println("난방을 켭니다.");
            }
            @Override
            public void turnOff() {
                System.out.println("난방을 끕니다.");
            }
        });
    }
}
```

```
TV를 켭니다.
TV를 끕니다.
에어컨을 켭니다.
에어컨을 끕니다.
난방을 켭니다.
난방을 끕니다.
```

- **Button.java**

```
package ch09.sec07.exam03;

public class Button {
    //정적 멤버 인터페이스
    public static interface ClickListener {
        //추상 메소드
        void onClick();
    }

    //필드
    private ClickListener clickListener;

    //메소드
    public void setClickListener(ClickListener clickListener) {
        this.clickListener = clickListener;
    }

    public void click() {
        this.clickListener.onClick();
    }
}
```

- **Button.java**

```
package ch09.sec07.exam03;

public class ButtonExample {
    public static void main(String[] args) {
        //Ok 버튼 객체 생성
        Button btnOk = new Button();

        //Ok 버튼 객체에 ClickListener 구현 객체 주입
        btnOk.setClickListener(new Button.ClickListener() {
            @Override
            public void onClick() {
                System.out.println("Ok 버튼을 클릭했습니다.");
            }
});

        //Ok 버튼 클릭하기
        btnOk.click();
    }
}
```


- Button.java

```
//Cancel 버튼 객체 생성
Button btnCancel = new Button();

//Cancel 버튼 객체에 ClickListener 구현 객체 주입
btnCancel.setOnClickListener(new Button.ClickListener() {
    @Override
    public void onClick() {
        System.out.println("Cancel 버튼을 클릭했습니다.");
    }
});

//Cancel 버튼 클릭하기
btnCancel.click();
}
```

Ok 버튼을 클릭했습니다.
Cancel 버튼을 클릭했습니다.