

내부클래스

1. 내부 클래스 (Inner Class)

1-1. 내부 클래스란?

- 클래스 내부에 선언된 클래스
- 외부 클래스와 밀접한 관계를 맺으며 외부 클래스의 멤버에 쉽게 접근할 수 있는 클래스.(외부 클래스 접근 제어자 관계 없음)

1-2. 내부 클래스의 종류

종류	특징	선언 위치	사용 시기
인스턴스 내부 클래스	- 외부 클래스 인스턴스 필요 - 외부 클래스의 모든 멤버 접근 가능 - private 포함 모든 멤버 접근 가능	외부 클래스의 멤버 변수 선언부	외부 클래스와 밀접한 관계를 맺을 때
정적 내부 클래스	- 외부 클래스 인스턴스 불필요 - 외부 클래스의 static 멤버만 접근 가능 - 독립적인 클래스 구성 가능	외부 클래스의 멤버 변수 선언부	외부 클래스와 독립적인 기능 구현 시
지역 내부 클래스	- 메소드 내에서만 사용 가능 - 메소드 내의 지역변수 접근 가능 - 메소드 실행 시에만 사용	메소드 내부	특정 메소드에서만 사용되는 기능 구현 시
익명 내부 클래스	- 이름이 없는 일회성 클래스 - 인터페이스나 추상클래스 구현에 주로 사용 - 즉시 생성하고 사용	메소드 내부 또는 필드 선언 시	일회성 구현이 필요한 경우


2. 인스턴스 내부 클래스(Instance Inner Class)

2-1. 인스턴스 내부 클래스란?

- 외부 클래스의 인스턴스 멤버처럼 다뤄지는 클래스
- 외부 클래스의 인스턴스가 생성되어야만 사용 가능

2-2. 인스턴스 내부 클래스 사용 이유

- 외부 클래스의 private을 포함한 모든 멤버에 자유롭게 접근할 수 있어 데이터 은닉성을 유지하면서도 긴밀한 작업 가능
- 외부 클래스와 내부 클래스가 서로 연관성이 높은 경우, 코드의 가독성과 유지보수성 향상
- 특정 클래스 내에서만 사용되는 기능을 구현할 때 캡슐화를 강화 가능
- UI 이벤트 처리나 데이터 처리와 같은 콜백 기능을 구현할 때 유용

 내부 클래스는 외부 클래스의 인스턴스가 필요하므로, 메모리 사용량이 증가할 수 있다는 점을 고려해야 한다

2-3. 인스턴스 내부 클래스 사용 예시

2-3-1. 예시 1

```
public class OuterClass {
    private int num = 10;

    class InstanceInnerClass {
        void display() {
```

```

        System.out.println("외부 클래스의 num: " + num);
    }
}
}

```

실행 클래스

```

public class Main {
    public static void main(String[] args) {
        // 외부 클래스의 인스턴스 생성
        OuterClass outer = new OuterClass();

        // 인스턴스 내부 클래스의 인스턴스 생성
        OuterClass.InstanceInnerClass inner = outer.new InstanceInnerClass();

        // 내부 클래스의 메소드 호출
        inner.display();
    }
}

```

실행 결과

외부 클래스의 num: 10

2-3-2. 예시 2

```

public class OuterClass {
    private String message = "Hello";

    class InnerClass {
        private String innerMessage = "World";

        void combineMessages() {
            // 외부 클래스의 private 멤버와 내부 클래스의 private 멤버를 함께 사용
            System.out.println(message + " " + innerMessage);
        }
    }

    public void displayMessage() {
        InnerClass inner = new InnerClass();
        inner.combineMessages();
    }
}

```

실행 클래스

```

public class Main {
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        outer.displayMessage();

        // 직접 내부 클래스 인스턴스 생성
        OuterClass.InnerClass inner = outer.new InnerClass();
        inner.combineMessages();
    }
}

```

실행 결과

```
Hello World
Hello World
```

3. 정적 내부 클래스(Static Inner Class)

3-1. 정적 내부 클래스란?

- static으로 선언된 내부 클래스
- 외부 클래스의 인스턴스 생성 없이도 사용 가능

3-2. 정적 내부 클래스 사용 이유

- 외부 클래스의 static 멤버만 접근이 필요한 경우
- 외부 클래스의 인스턴스 생성 없이도 내부 클래스를 사용할 수 있어 편리함
- 유틸리티성 기능을 구현할 때 적합하며, 독립적인 기능을 캡슐화할 수 있음
- 외부 클래스와 논리적으로 묶여있지만, 인스턴스 멤버와는 관계없는 작업을 수행할 때 유용함



단, 정적 내부 클래스는 외부 클래스의 인스턴스 멤버에는 직접 접근할 수 없음!

3-3. 정적 내부 클래스 사용 예시

```
public class OuterClass {
    private int outerInstanceNum = 100; // 인스턴스 변수
    private static int staticNum = 20; // 클래스 변수

    /* 정적 내부 클래스 정의*/
    static class StaticInnerClass {
        int innerInstanceNum = 300; // 정적 클래스 내부 변수

        void display(){
            // 오류 발생 - 외부 클래스의 인스턴스 변수 접근 불가
            // System.out.println("outerInstanceNum = " + outerInstanceNum);

            System.out.println("innerInstanceNum = " + innerInstanceNum);
            System.out.println("staticNum = " + staticNum);
        }

        static void staticDisplay(){

            // 오류 발생 - 같은 내부 클래스의 변수라도 static 메서드는 static 필드만 접근 가능
            // System.out.println("innerInstanceNum = " + innerInstanceNum);
            System.out.println("staticDisplay().staticNum = " + staticNum);
        }
    }
}
```

실행 클래스

```

public class Main {
    public static void main(String[] args) {
        // 정적 내부 클래스는 외부 클래스의 인스턴스 없이 직접 생성 가능
        OuterClass2.StaticInnerClass inner = new OuterClass2.StaticInnerClass();
        inner.display();

        OuterClass2.StaticInnerClass.staticDisplay();
    }
}

```

실행 결과

```

innerInstanceNum = 300
staticNum = 20
staticDisplay().staticNum = 20

```

3. 지역 내부 클래스(Local Inner Class)

3-1. 지역 내부 클래스란?

- 메소드 내부에서 선언되는 클래스
- 메소드 내부에서만 사용 가능한 클래스

3-2. 지역 내부 클래스 사용 이유

- 메소드 내에서만 한정적으로 사용되는 기능을 구현할 때 유용
- 특정 메소드에서만 사용되는 로직을 캡슐화할 수 있음
- 메소드의 지역변수와 매개변수에 쉽게 접근 가능

3-3-1. 예시 1

```

public class OuterClass {
    private String message = "Local Class Example";

    public void method() {
        final String localVar = "Hello from";

        class LocalInnerClass {
            void display() {
                System.out.println(localVar + " " + message);
            }
        }

        LocalInnerClass local = new LocalInnerClass();
        local.display();
    }
}

```

실행 클래스

```

public class Main {
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        outer.method();
    }
}

```

실행 결과

Hello from Local Class Example

4. 익명 내부 클래스(Anonymous Inner Class)

4-1. 익명 내부 클래스란?

- 이름이 없는 클래스로, 주로 인터페이스나 추상 클래스를 구현할 때 사용
- 일회성으로 사용되는 클래스를 간단히 구현할 때 유용함

4-2. 익명 내부 클래스 사용 이유

- 클래스 정의와 객체 생성을 동시에 할 수 있어 코드가 간결해짐
- 이벤트 처리나 콜백 함수 구현 시 많이 사용됨

4-3. 익명 내부 클래스 사용 예시

```
public interface Animal {  
  
    public abstract void move();  
  
    public abstract void eat();  
}
```

실행 클래스

```
public class Main4 {  
    public static void main(String[] args) {  
  
        // // 익명 내부 클래스 사용  
        Animal cat = new Animal() {  
            @Override  
            public void move() {  
                System.out.println("cat move");  
            }  
  
            @Override  
            public void eat() {  
                System.out.println("cat eat");  
            }  
        }; // 익명 클래스 정의 후 마지막에 꼭 세미콜론 추가!  
  
        cat.move();  
        cat.eat();  
    }  
}
```

실행 결과

```
cat move  
cat eat
```

5. 내부 클래스 사용 시 주의사항

- 내부 클래스는 외부 클래스의 모든 멤버에 접근할 수 있으므로, 캡슐화를 해칠 수 있습니다.
- 내부 클래스의 남용은 코드의 복잡성을 증가시킬 수 있습니다.

- 메모리 누수에 주의해야 합니다 (특히 익명 내부 클래스 사용 시).