
네스티드 클래스

네스티드 클래스

- 네스티드 클래스의 구분
  - 클래스 내에 또 다른 클래스를 정의할 수 있는데, 이를 네스티드 클래스라 하며 네스티드 클래스를 감사는 클래스를 카리켜 외부 클래스라 함
  - 네스티드 클래스는 static의 선언 여부를 기준으로 static 네스티드 클래스, non-static 네스티드 클래스(이너클래스)로 나뉨

```
public class OuterClass {
    static class StaticNestedClass {

    }
}
```

```
public class OuterClass {
    class InnerClass {

    }
}
```

  - 이너 클래스는 정의되는 위치나 특성에 따라 아래와 같이 세 종류로 구분
    - 멤버 이너 클래스(Member Inner Class) 또는 멤버 클래스
    - 로컬 이너 클래스(Local Inner Class) 또는 로컬 클래스
    - 익명 이너 클래스(Anonymous Inner Class) 또는 익명 클래스

네스티드 클래스

- Static 네스티드 클래스(Static Nested Class)
  - static 선언이 갖는 특성이 반영된 클래스
  - 자신을 감싸는 외부 클래스의 인스턴스와 상관없이 Static 네스티드 클래스의 인스턴스 생성 가능

```
public class Outer {
    private static int num = 0;
    static class Nested1 {
        void add(int n) { num += n; }
    }
    static class Nested2 {
        int get() { return num; }
    }
}
```

```
public class StaticNested {
    public static void main(String[] args) {
        Outer.Nested1 nst1 = new Outer.Nested1();
        nst1.add(5);
        Outer.Nested2 nst2 = new Outer.Nested2();
        System.out.println(nst2.get());
    }
}
```

  - Outer 클래스 내에 두 개의 Static 네스티드 클래스 정의
  - Nested1, Nested2 클래스 내에서는 Outer의 static 멤버 변수 num에 접근 가능하며 변수 num은 Nested1과 Nested2의 인스턴스가 공유하는데, 이는 Static 네스티드 클래스가 갖는 주요 특징
  - Static 네스티드 클래스의 인스턴스는 외부 클래스의 인스턴스를 생성하지 않고 생성 가능
  - Static 네스티드 클래스 내에서 외부 클래스의 인스턴스 변수와 메소드에 접근 불가능
  - 단, 외부 클래스에 static으로 선언된 변수와 메소드는 접근 가능

열거형, 가변 인자, 어노테이션

- 이너(Inner) 클래스의 구분
  - 이너 클래스의 구분
    - 멤버 클래스 (Member Class) : 인스턴스 변수, 인스턴스 메소드와 동일한 위치에 정의
    - 로컬 클래스 (Local Class) : 중괄호 내에, 특히 메소드 내에 정의

```
public class Outer {
    class MemberInner {...} // 멤버 클래스
    void method() { class LocalInner {...} }    // 로컬 클래스
}
```

    - 익명 (Anonymous Class) : 클래스의 이름이 존재하지 않는 클래스

네스티드 클래스

- 멤버 클래스 (Member Class)

```
public class Outer {
    private int num = 0;
    class Member {   // 멤버 클래스 정의
        void add(int n) {
            num += n;
        }
        int get() {
            return num;
        }
    }
}
```

```
public class MemberInner {
    public static void main(String[] args) {
        Outer o1 = new Outer();
        Outer o2 = new Outer();
        // o1 기반으로 두 인스턴스 생성
        Outer.Member o1m1 = o1.new Member();
        Outer.Member o1m2 = o1.new Member();
        // o1 기반으로 두 인스턴스 생성
        Outer.Member o2m1 = o2.new Member();
        Outer.Member o2m2 = o2.new Member();
        // o1 기반으로 생성된 두 인스턴스 메소드 호출
        o1m1.add(5);
        System.out.println(o1m2.get());
        // o2 기반으로 생성된 두 인스턴스 메소드 호출
        o2m1.add(7);
        System.out.println(o2m2.get());
    }
}
```

- Member 클래스 내에서 Outer 클래스 인스턴스 변수에 접근 가능
- 멤버 클래스의 인스턴스는 외부 클래스의 인스턴스에 종속적
- 즉, Outer 클래스의 인스턴스(o1, o2)로부터 생성한 멤버 클래스의 인스턴스는 Outer 클래스의 인스턴스(o1, o2)의 멤버 공유

네스티드 클래스

- "멤버 클래스"를 언제 사용하는가?
  - 클래스의 정의를 감추어야 할 때 유용

```
public interface Printable {
    void print();
}
```

```
public class UseMemberInner {
    public static void main(String[] args) {
        // Printer 클래스의 인스턴스 생성
        Papers p = new Papers("서류 내용 : 행복합니다.");
        Printable prn = p.getPrinter();
        prn.print();
    }
}
```

```
public class Papers {
    private String con;
    public Papers(String con) {
        this.con = con;
    }
    // 멤버 클래스의 정의
    private class Printer implements Printable {
        public void print() {
            System.out.println(con);
        }
    }
    public Printable getPrinter() {
        // 멤버 클래스 인스턴스 생성 및 반환
        return new Printer();
    }
}
```

  - Printer 클래스를 private로 선언하면 이 클래스 정의를 감사는 클래스 내에서만 인스턴스 생성 가능
  - 따라서 Printer 클래스의 인스턴스는 위 코드와 같은 방법(getPrinter 메소드 사용)으로만 참조 가능
  - Papers 클래스의 외부에서는 getPrinter 메소드가 어떠한 인스턴스의 참조 값을 반환하는지 알 수 없음

네스티드 클래스

- 다만 반환되는 참조 값의 인스턴스가 Printable을 구현하고 있어 Printable의 참조 변수로 참조할 수 있다는 사실만 알 수 있으며, 이러한 상황을 "클래스의 정의가 감추어진 상황"이라 함
- 클래스의 정의를 감추면, getPrinter 메소드가 반환하는 인스턴스가 다른 클래스의 인스턴스로 변경되어도 Paper 클래스 외부의 코드는 수정할 필요가 없어지므로 코드에 유연성 부여
- 따라서 Printer 클래스의 인스턴스는 위 코드와 같은 방법(getPrinter 메소드 사용)으로만 참조 가능
- Papers 클래스의 외부에서는 getPrinter 메소드가 어떠한 인스턴스의 참조 값을 반환하는지 알 수 없음

네스티드 클래스

- 로컬 클래스(Local Class)
  - 로컬 클래스는 멤버 클래스와 상당 부분 유사하지만 if문, while문, 메소드 몸체와 같은 블록 안에 클래스를 정의

```
public interface Printable {
    void print();
}
```

```
public class UseLocalInner {
    public static void main(String[] args) {
        Papers p = new Papers("서류 내용 : 행복합니다.");
        Printable prn = p.getPrinter();
        prn.print();
    }
}
```

```
public class Papers {
    private String con;
    public Papers(String con) {
        this.con = con;
    }
    public Printable getPrinter() {
        // 로컬 클래스의 정의
        class Printer implements Printable {
            public void print() {
                System.out.println(con);
            }
        }
        // 로컬 클래스 인스턴스 생성 및 반환
        return new Printer();
    }
}
```

  - 메소드 내에 클래스를 정의하면 해당 메소드 내에서만 인스턴스 생성이 가능
  - 즉, private 선언은 의미가 없으며 멤버 클래스보다 클래스를 더 깊이 감추는 효과 발생

네스티드 클래스

- 익명 클래스 (Anonymous Class)
  - 익명 클래스는 다름 챕터의 "람다"와 관련이 있음
  - 이전에 작성했던 코드를 확인하면 Printer 클래스의 정의와 Printer 인스턴스의 생성이 분리되어 있음
  - 이를 익명 클래스 형태로 정의하면 정의와 인스턴스 생성을 동시에 실행

```
public interface Printable {
    void print();
}
```

```
public class UseAnonymousInner {
    public static void main(String[] args) {
        Papers p = new Papers("서류 내용 : 행복합니다.");
        Printable prn = p.getPrinter();
        prn.print();
    }
}
```

```
public class Papers {
    private String con;
    public Papers(String con) {
        this.con = con;
    }
    public Printable getPrinter() {
        // 익명 클래스 정의와 인스턴스 생성 후 반환
        return new Printable() {
            @Override
            public void print() {
                System.out.println(con);
            }
        };
    }
}
```

  - "new Printable()" – 인터페이스 Printable을 대상으로 인스턴스 생성
  - "new Printable()" 뒤에 정의 부분을 붙여 인스턴스 생성
  - "new Printable()" 뒤에 정의 부분은 이름이 없는 클래스의 정의이고, 이를 "익명 클래스"라고 함

네스티드 클래스

- 컬렉션 프레임워크와 관련한 익명 클래스 정의

```
import java.util.Comparator;

public class StrComp implements Comparator<String> {
    @Override
    public int compare(String s1, String s2) {
        return s1.length() - s2.length();
    }
}
```

```
public class SortComparator {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Robot");
        list.add("Box");
        StrComp cmp = new StrComp();
        Collections.sort(list, cmp);
        System.out.println(list);
    }
}
```

- 위 코드를 익명 클래스 기반으로 수정 (람다 등장 이전 코드 스타일)

```
public class AnonymousComparator {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Robot");
        list.add("Box");
        Comparator<String> cmp = new Comparator<String>()
{
            @Override
            public int compare(String s1, String s2) {
                return s1.length() - s2.length();
            }
        };
        Collections.sort(list, cmp);
        System.out.println(list);
    }
}
```