

네스티드 클래스와 람다의 소개

26-1. 네스티드 클래스와 이너 클래스

네스티드 클래스의 구분

```
class Outer { // 외부 클래스
    class Nested {...} // 네스티드 클래스
}
```

```
class OuterClass {
    static class StaticNestedClass {...} // Static 네스티드 클래스
}
```

```
class OuterClass {
    class InnerClass {...} // Non-static 네스티드 클래스, 이너 클래스
}
```

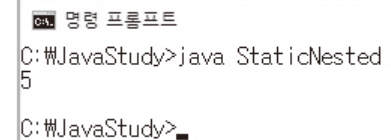
- 멤버 (이너) 클래스 (Member Inner Class)
- 로컬 (이너) 클래스 (Local Inner Class)
- 익명 (이너) 클래스 (Anonymous Inner Class)

Static 네스티드 클래스

```
class Outer {  
    private static int num = 0;  
    static class Nested1 {    // Static 네스티드 클래스  
        void add(int n) { num += n; }    Outer 클래스의 static 변수 공유!  
    }  
    static class Nested2 {    // Static 네스티드 클래스  
        int get() { return num; }  
    }  
}
```

Static 네스티드 클래스는 static 선언이 갖는 특성이 반영된 클래스이다. 따라서 자신을 감싸는 외부 클래스의 인스턴스와 상관없이 Static 네스티드 클래스의 인스턴스 생성이 가능하다.

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



```
명령 프롬프트  
C:\WJavaStudy>java StaticNested  
5  
C:\WJavaStudy>
```

이너 클래스의 구분

- 멤버 클래스 (Member Class)

→ 인스턴스 변수, 인스턴스 메소드와 동일한 위치에 정의

- 로컬 클래스 (Local Class)

→ 중괄호 내에, 특히 메소드 내에 정의

- 익명 클래스 (Anonymous Class)

→ 클래스인데 이름이 없어! ^^

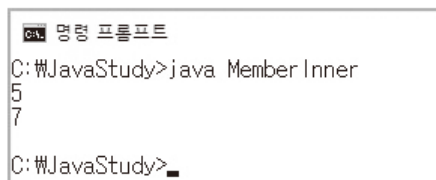
```
class Outer {  
    class MemberInner {...}    // 멤버 클래스  
  
    void method() {  
        class LocalInner {...}    // 로컬 클래스  
    }  
}
```

멤버 클래스

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

“멤버 클래스의 인스턴스는

외부 클래스의 인스턴스에 종속적이다.”



```
명령 프롬프트  
C:\JavaStudy>java MemberInner  
5  
7  
C:\JavaStudy>
```

```
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();  
  
        // o2 기반으로 두 인스턴스 생성  
        Outer.Member o2m1 = o2.new Member();  
        Outer.Member o2m2 = o2.new Member();  
  
        // o1 기반으로 생성된 두 인스턴스의 메소드 호출  
        o1m1.add(5);  
        System.out.println(o1m2.get());  
  
        // o2 기반으로 생성된 두 인스턴스의 메소드 호출  
        o2m1.add(7);  
        System.out.println(o2m2.get());  
    }  
}
```

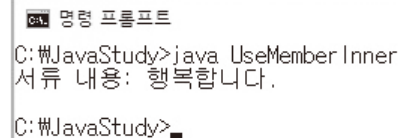
'멤버 클래스'를 언제 사용하는가?

멤버 클래스는 클래스의 정의를 감추어야 할 때 유용하게 사용이 된다.

```
interface Printable {  
    void print();  
}  
  
class Papers {  
    private String con;  
    public Papers(String s) { con = s; }  
  
    public Printable getPrinter() {  
        return new Printer();  
    }  
  
    private class Printer implements Printable {  
        public void print() {  
            System.out.println(con);  
        }  
    }  
}
```

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

클래스 사용자 입장에서 `Printable` 인터페이스는 알지만
`Printer` 클래스는 모른다! 알 필요도 없다!



```
C:\JavaStudy>java UseMember Inner  
서류 내용: 행복합니다.  
C:\JavaStudy>
```

반복자가 멤버 클래스라는 사실!

```
public static void main(String[] args) {  
    List<String> list = new ArrayList<>();  
    ....  
    Iterator<String> itr = list.iterator();    // 반복자 획득!  
    ....  
}
```

```
public class ArrayList<E> implements List<E> {  
    ....  
    public Iterator<E> iterator() {  
        return new Itr();    // 멤버 클래스의 인스턴스 생성 및 반환  
    }  
  
    private class Itr implements Iterator<E> {    // 멤버 클래스의 정의  
        ....  
    }    반복자 클래스를 감췄다! 그 이름까지도!  
}
```


로컬 클래스 (Local Class)

‘로컬 클래스’는 바로 위에서 소개한 ‘멤버 클래스’와 상당 부분 유사하다.

지역 내에 정의된다는 점에서만 차이를 보임

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

```
class Papers {  
    private String con;  
    public Papers(String s) { con = s; }  
  
    public Printable getPrinter() {  
        class Printer implements Printable {  
            public void print() {  
                System.out.println(con);  
            }  
        }  
        return new Printer();  
    }  
}
```

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

명령 프롬프트

```
C:\JavaStudy>java UseLocalInner  
서류 내용: 행복합니다.
```

```
C:\JavaStudy>
```

익명 클래스 (Anonymous Class)

```
public Printable getPrinter() {  
    class Printer implements Printable { // 로컬 클래스 Printer의 정의  
        public void print() {  
            System.out.println(con);  
        }  
    }  
  
    return new Printer(); // Printer 인스턴스의 생성  
}
```

interface **Printable** { void print(); }



```
public Printable getPrinter() {  
    return new Printable() { // 익명 클래스의 정의와 인스턴스 생성  
        public void print() {  
            System.out.println(con);  
        }  
    };  
}
```

익명 클래스 사용의 예

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

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



```
class AnonymousComparator {  
    public static void main(String[] args) {  
        List<String> list = new ArrayList<>();  
        list.add("ROBOT");  
        list.add("APPLE");  
        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);  
    }  
}
```

26-2. 람다의 소개

람다의 이해1

```
interface Printable {  
    void print(String s);  
}
```

```
class Printer implements Printable {  
    public void print(String s) {  
        System.out.println(s);  
    }  
}
```

```
class Lambda1 {  
    public static void main(String[] args) {  
        Printable prn = new Printer();  
        prn.print("What is Lambda?");  
    }  
}
```



```
interface Printable {  
    void print(String s);  
}
```

```
class Lambda2 {  
    public static void main(String[] args) {  
        Printable prn = new Printable() { //익명 클래스  
            public void print(String s) {  
                System.out.println(s);  
            }  
        };  
  
        prn.print("What is Lambda?");  
    }  
}
```

아직 람다 등장 안 했음!

람다의 이해2

```
interface Printable {  
    void print(String s);  
}
```

```
class Lambda2 {  
    public static void main(String[] args) {  
        Printable prn = new Printer() {  
            public void print(String s) {  
                System.out.println(s);  
            }  
        };  
  
        prn.print("What is Lambda?");  
    }  
}
```



드디어 람다 등장

```
interface Printable { // 추상 메소드가 하나인 인터페이스  
    void print(String s);  
}  
  
class Lambda3 {  
    public static void main(String[] args) {  
        Printable prn = (s) -> { System.out.println(s); };  
        prn.print("What is Lambda?");  
    }  
}
```

람다의 이해3 : 생략 가능한 것을 지워보자.

```
Printable prn = new Printable() {  
    public void print(String s) {  
        System.out.println(s);  
    }  
};
```

```
interface Printable {  
    void print(String s);  
}
```

prn이 Printable형 참조변수이니 = 의 왼편에는 new가 당연히 올 것이고,
메소드 정의가 온 것을 보니, 익명 클래스를 기반으로 보건대 이는 인스턴스 생성이야!



```
Printable prn = new Printable() {  
    public void print(String s) {  
        System.out.println(s);  
    }  
};
```

람다의 이해4

```
Printable prn =  
    public void print(String s) {  
        System.out.println(s);  
    } ;
```

```
interface Printable {  
    void print(String s);  
}
```

Printable 인터페이스에 있는 메소드 그거 public void print(String s)니 뻔하지 뭐!



```
Printable prn =  
    public void print(String s) {  
        System.out.println(s);  
    } ;
```


람다의 이해5

```
Printable prn = { System.out.println(s); };
```

```
interface Printable {  
    void print(String s);  
}
```

컴파일러가 저 `s`가 매개변수라고 판단해 주길 바라는 것은 무리이니까!

```
Printable prn = (String s) -> { System.out.println(s); };
```

 완성된 람다식!

`s`가 `String`형 임은 `Printable` 인터페이스 보면 알 수 있지 않아?

```
Printable prn = (s) -> { System.out.println(s); };
```

 조금 더 줄이면!

람다식의 인자 전달

```
interface Printable {  
    void print(String s);  
}
```

```
Printable prn = (s) -> { System.out.println(s); };
```

```
method((s) -> System.out.println(s));    // void method(Printable prn) {...}
```