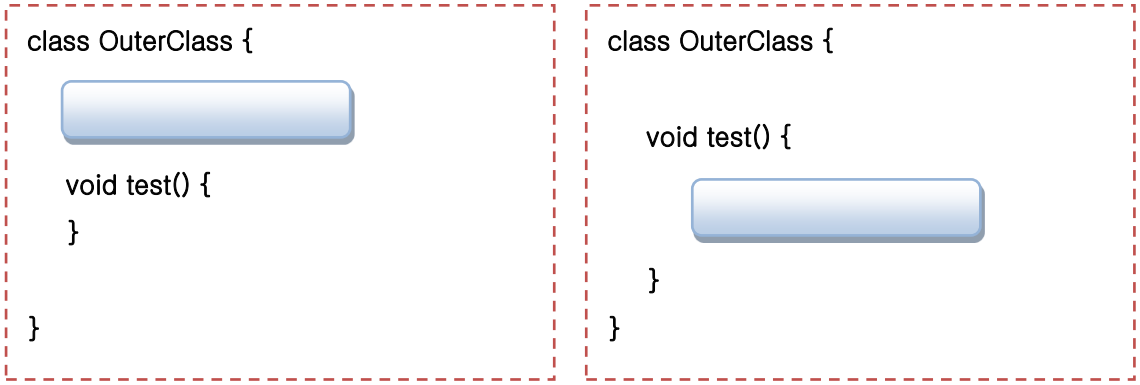
내부클래스

내부 클래스란 클래스의 내부에 정의되는 클래스로서 특정 클래스의 내에서만 주로 사용되는 클래스를 내부 클래스로 정의한다. 내부 클래스에서는 외부 클래스의 멤버들을 접근할 수 있으며 캡슐화를 통해 코드의 복잡성을 줄일 수 있다.

■ **내부 클래스의 종류**

내부 클래스는 정의되는 위치에 따라 멤버 클래스와 로컬 클래스로 나뉜다. 멤버 변수와 지역 변수로 나뉘는 변수와 같이 내부 클래스도 클래스의 멤버로 정의되는 멤버 클래스와 메서드 내에 정의되는 로컬 클래스로 나뉘며 각각 변수와 비슷한 유효범위와 성격을 지원한다.



[멤버 클래스] [로컬 클래스]

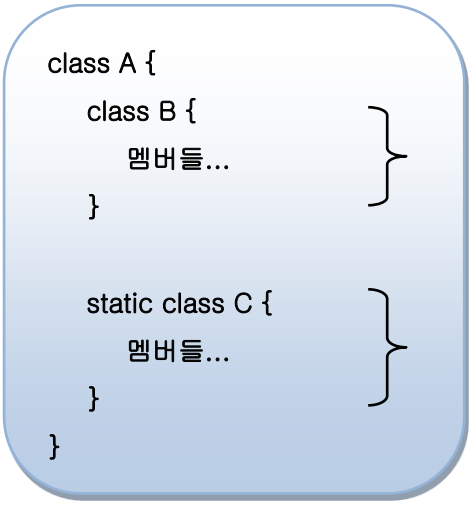
멤버 클래스는 인스턴스 클래스와 스태틱 클래스로 구성되며 로컬 클래스는 이름이 있는 로컬 클래스와 이름이 없는 로컬 클래스(anonymous 클래스)로 나뉜다.
내부 클래스는 다른 클래스내에 포함되어 정의되지만 클래스 파일은 독립적으로 만들어 다음과 같은 규칙으로 파일명이 정해진다.

내부 클래스의 종류	생성되는 클래스명의 규칙
인스턴스 클래스	**외부클래스$내부클래스.class**
스태틱 클래스	**외부클래스$내부클래스.class**
이름있는 로컬 클래스	**외부클래스N내부클래스.class**
이름없는 로컬 클래스	**외부클래스$N.class**

로컬 클래스의 경우에는 Java 소스에 정의되는 순서에 따라서 클래스명 중간 또는 뒤에 1부터 시작하는 숫자가 자동으로 부여된다.

■ **멤버 클래스**

동일 클래스에서는 물론이고 다른 클래스에서도 이 클래스들을 사용할 수 있다. 멤버 변수와 비슷한 성격을 갖는다. **인스턴스 클래스**와 **스태틱 클래스**로 나뉜다.



인스턴스 클래스이며 A$B.class 명의 클래스 파일이 만들어진다.

스태틱 클래스이며 A$C.class 명의 클래스 파일이 만들어진다.

인스턴스 클래스는 외부 클래스의 인스턴스 멤버처럼 다루어 지며 주로 외부 클래스의 인스턴스 멤버들과 관련된 작업에 사용될 목적으로 정의된다. 인스턴스 클래스 내에서는 static 멤버를 정의할 수 없다.

스태틱 클래스는 외부 클래스의 클래스 멤버(static 멤버)처럼 다루어 지며 주로 외부 클래스의 클래스 메서드내에서 사용될 목적으로 정의된다.

멤버 클래스들도 멤버 변수들과 비슷한 방법으로 접근 제어자 지정이 가능하며 동일 클래스에서 뿐만 아니라 외부 클래스의 밖에서도 접근하고자 하는 경우에는 각각 다음과 같은 방법으로 사용할 수 있다.

인스턴스 클래스	**A a =new A();** **A.B b = a.new B();** **b.멤버**
스태틱 클래스	**A.C.멤버**

■ **로컬 클래스**

메서드 내에 정의되는 클래스로서 로컬 변수와 비슷한 성격을 갖으며, 활용 범위가 정의되어 있는 메서드 블럭 내부로 제한된다. Interface 는 로컬로 정의될 수 없다. 포함하는 클래스의 멤버 변수와 포함하는 메서드의 final 로컬변수, final 매개변수사용이 가능하다.

로컬 클래스는 이름이 있는 로컬 클래스와 이름이 없는 로컬 클래스로 나뉘며 이름이 있는 로컬 클래스를 **로컬 클래스**, 이름이 없는 로컬 클래스를 **익명 클래스**라고 부른다.

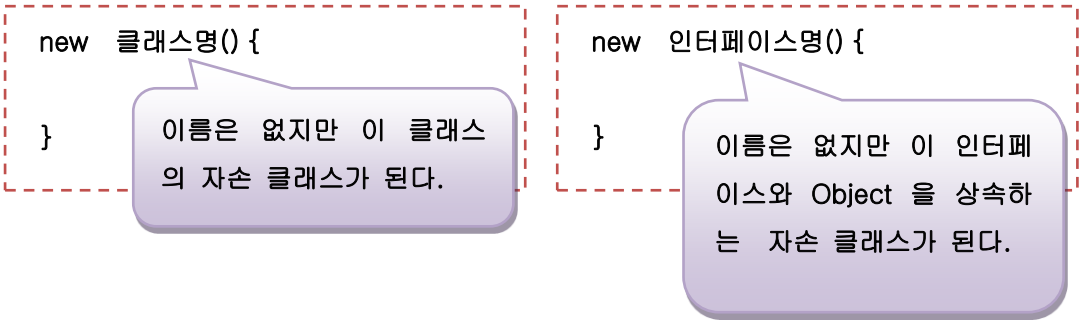
```
class X {
    iny num;
    void sam(final int i) {
        int total = 20;
        final String s="test";
        class Y {
            멤버들...
        }
        Y y = new Y();
        y.멤버들...;
    }
}
```

로컬 클래스이며 X1Y.class 명의 클래스 파일이 만들어진다. Y 클래스 내에서는 X 클래스의 멤버 변수 num, sam() 메서드의 final 지역변수 s 를 사용할 수 있다. Y 클래스는 sam() 메서드 내에서만 사용 가능하다.

```
class N {
    void pr(Test t) {
      ...
    }
    void sam() {
        pr(new Test() {
            멤버들...
        });
        ..
    }
}
```

익명 클래스이며 N$1.class 명의 클래스 파일이 만들어진다. 정의된 위치에서 한 번만 객체 생성이 가능하다. 클래스의 정의와 객체 생성을 동시에 하는 1회용 클래스라고 할 수 있다.

익명 클래스의 경우 new 키워드 뒤의 생성자 메서드의 명칭이 기존 클래스 명인 경우에는, 자동적으로 이 클래스의 자손 클래스가 되며, 인터페이스 명인 경우에는 이 인터페이스를 구현하여 추가 상속하는 클래스로서 부모 클래스는 Object 이 된다.



메서드 호출 시 매개변수의 타입이 추상 클래스 형이거나 인터페이스 형이어서 가볍게 자손 클래스를 구현하고 객체를 생성해서 전달하려는 경우 유용하게 사용될 수 있는 구문이다.

열거타입(enum)

열거타입은 한정된 데이터 값만을 저장할 수 있는 타입을 정의하는데 사용되는 또 다른 Java 프로그램의 구조로서 상수 정의를 목적으로 구현하는 클래스이다. 데이터 값을 비교할 때 타입까지 적용하고자 할 때 사용되며 열거타입으로 선언된 변수에는 열거 타입에 정의된 상수값 만을 데이터 값으로 저장할 수 있다. Java SE 5.0에서 새로이 추가된 구문으로 OCJP 6.0시험에서 다양한 형태의 문제에서 열거타입이 출제되고 있다.

.

■ **열거타입의 도입 배경**

프로그램에서 처리하고자 하는 데이터들 중에는 한정된 값만을 다루게 되는 경우가 있다. 요일에 대한 데이터의 경우 월,화,수...일 처럼 7개의 값을, 계절에 대한 데이터의 경우 봄, 여름, 가을 겨울이라는 4개의 값을 갖는 처리하게 된다. 이런한 데이터들는 정수나 문자 또는 문자열 리터럴을 이용해서 표현하거나 상수를 선언해서 사용하기도 하였다.

```
public class Season{
    public final static int SPRING = 1;
    public final static int SUMMER = 2;
    public final static int FALL = 3;
    public final static int WINTER = 4;
}
```

Season클래스에 선언되어 있는 SUMMER상수를 참조하기 위한 방법은 다음과 같다.

System.out.println(Season.SUMMER); → 2 가 출력된다.

Season클래스의 소스 내용을 파악하지 못했다면 2 라는 숫자가 어떠한 계절을 의미하는지 알 수 없다. 하여 Java SE 5.0에서는 이러한 단점을 보완하기 위해 열거타입이 Java의 기본 구문으로 추가되었다.

열거타입은 클래스나 인터페이스처럼 직접 만들어서 사용해야 하며 클래스로 취급되기 때문에 열거타입의 이름도 클래스명의 작성 관례에 따라 만들어야 한다.

■ **열거타입의 정의와 사용**

열거타입을 정의하는 형식은 다음과 같다.

```
enum 열거타입이름{
    열거상수리스트(콤마로 구분)
}
```

예를 들어 사계절을 열거타입으로 표현하면 다음과 같다.

```
enum Season{
      SPRING, SUMMER, FAIL, WINTER
}
```

```
Season   info = Season.SUMMER;
System.out.println(Season.SPRING);
```

클래스에 종속적인 열거타입을 정의할 수도 있다

```
class 클래스명 {
    enum 열거타입이름{
        열거상수리스트(콤마로 구분)
    }
    :
}
```

클래스의 멤버로 정의되는 열거타입은 정적 내부클래스로 만들어지므로 외부에서 클래스에 종속적인 열거타입을 사용하려면 **클래스명.열거타입이름.열거상수이름** 으로 사용한다.

```
class Tour  {
    enum Season{
        SPRING, SUMMER, FALL, WINTER
    }
}
```

```
Tour.Season   info = Tour.Season.SUMMER;
System.out.println(Tour.Season.SPRING); // SPRING 이 출력된다.
```

■ values() 메

values()메서드와 valueof() 메서드는 열거타입이 컴파일될 때 자동으로 생성되는 클래스 메서드들이다. values()메서드는 열거타입에 속하는 모든 열거 상수들을 배열에 담아서 리턴하고 valueOf()메서드는 열거상수의 이름을 문자열로 넘겨주면 그에 해당하는 열거상수를 리턴하는 메서드이다.

```
enum Season{
    SPRING, SUMMER, FALL, WINTER
}

class SeasonTest {
    public static void main(String args[]) {
        Season day[] = Season.values();
        for(Season value  : day)
        System.out.println(value);      // SPRING, SUMMER, FALL, WINTER 출력

        Season season = Season.valueOf("SUMMER");
        System.out.println(season);      //SUMMER 출력
    }
}
```

■ **열거상수를 다른 값과 매핑하기**

열거타입에 정의되는 상수들은 상수이름을 값으로 자동초기화 하지만 열거상수를 다른 값과 매핑하기 위해서는 상수 뒤에 괄호와 함께 값을 설정할 수 있다. 그런데 이 때에는

- 설정하려는 값을 저장하기 위한 private final 형 멤버변수
- 값을 설정할 수 있는 생성자 메서드
- 값을 리턴하는 메서드

를 같이 추가해 주어야 한다.
다음은 열거타입 Season에 정의되는 상수들 **SPRING, SUMMER, FALL 그리고 WINTER** 에 각각 **"봄", "여름", "가을" 그리고 "겨울"** 을 값으로 매핑하도록 구현하고 있는 열거타입 정의 예제이다.

```
enum Season{
    SPRING("봄"), SUMMER("여름"), FALL("가을"),   WINTER("겨울")
    private final String name;
    Season(String name){
        this.name = name;
    }
    String returnName(){
        return name;
    }
}
```