2023-03-23

내부 클래스 / 이너 클래스

클래스 내부에 정의되는 클래스로 특정 클래스의 내에서만 주로 사용되는 클래스.

내부 클래스에서는 외부 클래스의 멤버들을 접근할 수 있으며 캡슐화를 통해 코드의 복장성을 줄일 수 있다.

정의되는 위치에 따라 멤버 클래스와 로컬 클래스로 나뉜다.

멤버 변수와 지역 변수로 나뉘는 변수와 같이 내부클래스도 클래스의 멤버로 정의되는 멤버 클래스와 메서드내에 정의되는 로컬 클래스로 나뉜다.

각각 변수와 비슷한 유효범위와 성격을 지원한다.

annoymousInnerLocalClass는 객체를 하나만 생성할 수 있다.

내부 클래스의 종류	생성되는 클래스명의 규칙
인스턴스 클래스(이너멤버)	외부클래스$내부클래스.class
스태틱 클래스(이너멤버)	외부클래스$내부클래스.class
이름있는 로컬 클래스	외부클래스$N내부클래스.class
이름없는 로컬 클래스	외부클래스$N.class

멤버 클래스

동일클래스에서는 물론. 다른 클래스에서도 이 클래스를 사용할 수 있으며 멤버 변수와 비슷한 성격을 갖는다.

```
class A {
   // 인스턴스 클래스이며 A$B.class 명의 클래스 파일이 만들어진다.
  class B {
   멤버들...
   }
  // 스태틱 클래스이며 A$C.class 명의 클래스 파일이 만들어진다.
 static class C {
   멤버들...
 }
}
```

```
// 인스턴스 클래스
A a = new A();
A.B b = a.newB();
b.멤버
```

```
// 스태틱클래스
A.C.멤버
```

로컬 클래스

메서드 내에서 정의하는 클래스.

```
class X {
 iny num;
 void sam(final int i) {
 int total = 20;
 final String s="test";
  // 로컬 클래스인 x$1Y.class명의 클래스 파일이 만들어지며 Y 클래스 내에서 X클래스의 멤버변수
  // num, sam() 메서드의 final 지역변수 s를 사용할 수 있다. Y 클래스는 sam()메서드 내에서만 사용가능
 class Y {
 멤버들...
 }
 Y y = new Y();
 y.멤버들...;
 }
}
```

```
class N {
 void pr(Test t) {
 ...
 }
 void sam() {
// 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.
 pr(new Test() { // new Test() : 객체생성과 클래스 정의를 동시에 하는 것.
 멤버들...
 });
 ..
 }
```

여러 객체에서 필요하다면 익명로컬클래스로 만들기에는 적합하지않다.

```
new 클래스명() {
 // 이름은 없지만 이 클래스의 자식 클래스가 된다.
 }

new 인터페이스명() {
// 이름은 없지만 이 인터페이스와 Object 을 상속하는 자식 클래스가 된다.
 }
```

매서드 호출 시 매개변수의 타입이 추상 클래스 형이거나 인터페이스 형이어서 가볍게 구현(자식) 클래스를 만들고 객체를 생성해서 전달하려는 경우 유용하게 사용할 수 있다.

💡 이름없는 로컬클래스는 한번만 객체생성이 가능하다.

JDBC

JDBC의 경우 대부분의 API가 인터페이스이다.

- Connection, Statement, ResultSet, PreparedStatement ……

```
Connection : createStatement(), getMetaData()……
Statement : executeQuery(), executeUpdate()……
ResultSet : next(), getXXX()……
```

이 API들을 상속하고 구현하고 있는 자식 클래스가 필요하며 그 자식 클래스들을 JDBC드라이버에서 제공한다.

JDBC API 내에서는 JDBC 드라이버가 제공하는 각 인터페이스들의 자식클래스가 어떠한 이름의 클래스인지 모르더라도 프로그래밍 가능하도록 팩토리 메서드를 제공하고 있다.

일반 메서드로서 다른 클래스의 객체생성을 대신해주는 메서드를 팩토리 메서드라고 한다.

JDBC 기술의 구성

JDBC API (java.sql, javax.sql) -> 공통적(모든 DB 서버에 대해)
JDBC Driver -> DB 서버마다 달라진다.

```
[ 인터페이스 ]
Connection, Statement, PreparedStatement, ResultSet
DatabaseMetaData, ResultSetMetaData
 [ 클래스 ]
DriverManager, Date, Time, Timestamp
```

데이터 읽기

DriverManager를 이용해서 Connection인스턴스를 얻는다.

Connection을 통해서 Statement를 얻는다.

Statement를 이용해 ResultSet을 얻는다.

```
public class ConnectMySQL {
  public static void main(String[] args) {
    try {
      // 1. Driver 로드
      // DriverManager로 어떤 DB를 사용할 것인지 드라이버를 로드한다.
      Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException cnfe) {
      System.out.println("해당 클래스를 찾을 수 없습니다." + cnfe.getMessage());
```

```
    return;
    }
    // 2. Connection 얻기
    // DB를 결정 후, 연결을 위해 연결 정보(DB 서버 url, ID, PW등)를 입력한다
    String url = "jdbc:mysql://localhost:3306/edudb?characterEncoding=UTF-8&serverTimezone=UTC";
               // jdbc:DB서버이름:JDBC드라이버에대한정보와드라이버에게전달할정보
    String user = "jdbctest";
    String passwd = "jdbctest";
    try (Connection conn = DriverManager.getConnection(url, user, passwd);){
      DatabaseMetaData md = conn.getMetaData();
      System.out.println("DBMS 서버 명 : "+md.getDatabaseProductName());
      System.out.println("DBMS 서버 버전 : "+md.getDatabaseProductVersion());
      System.out.println("사용자명 : "+md.getUserName());
    } catch (SQLException se) {
        System.out.println(se.getMessage());
    }

    // 3. Statement 작성
    // DB 서버에 SQL 명령을 전달하여 실행시키기 위한 객체를 생성한다.
    Statement stmt = conn.createStatement();
    // 4. SELECT 명령을 실행하고 실행 결과를 ResultSet에 담기
    // SELECT 명령을 실행한 결과는 JDBC 드라이버가 ResultSet이라는 객체로 반환한다.
    ResultSet rs = stmt.excuteQuery("SELECT ename, sal FROM emp");
    // 5. ResultSet 객체에서 값 꺼내오기
    while(rs.next()) {
      System.out.println(rs.getString(1));
      System.out.println(rs.getInt(2));
    }
    while(rs.next()) {
      System.out.println(rs.getString("ename"));
      System.out.println(rs.getInt("sal"));
    }
    // 6. 커넥션 반환하기
    // 반환 작업은 사용했던 객체를 역순으로 닫는다.
    rs.close();
    stmt.close();
    conn.close();
    }
}
```

- ResultSet

  Query의 실행 결과를 담는 객체로 최초의 결과 집합에서의 0번째 줄을 가리키고 있다.

  다음줄을 가리키기 위해서는 next()메서드를 수행해야한다.

  ```
  while(rs.next() {
    // 컬럼단위로 데이터를 추출한다.
  }
  // next()의 return값이 false이다.
  ```

select 된 결과가 없을 땐 비어있는 result set 객체가 return 된다.