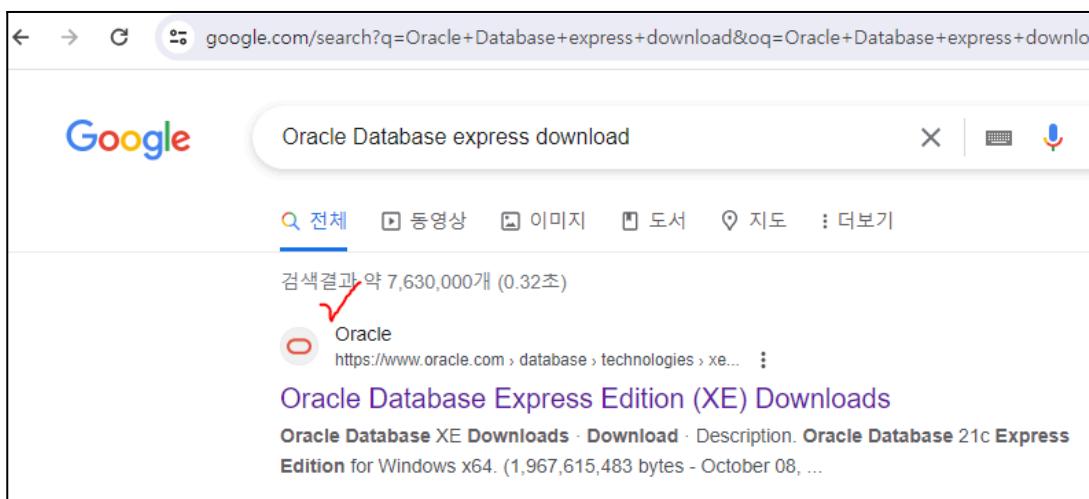


04. 데이터베이스 JDBC프로그래밍

> 01. 오라클 설치 및 계정 생성

오라클 데이터베이스를 설치하여 보자. www.oracle.com 사이트에 들어가 상단
돋보기 모양 아이콘을 클릭하고 검색부분에 Oracle Database express download 를
입력하고 검색하거나 크롬브라우저에서 검색하면 다음과 같이 체크된 부분의 경로를 얻을
수 있다. 아래 이미지는 크롬 브라우저로 검색한 결과이다. 버전이 업 될때 마다 검색
결과가 달라지니 잘 찾을 수 있는 방법을 선택하자.



시간이 지날때 마다 결과 화면들이 달라질수 있으니 잘 검색해 보자. 클릭하고 들어 가면 다음과 같은 화면이 보이는데 windows x64환경을 클릭해서 다운로드 받을 수 있다.

oracle.com/kr/database/technologies/xe-downloads.html

ORACLE

제품 산업 리소스 고객 파트너 개발자 기업

Oracle Database XE Downloads

Oracle Database 21c Express Edition

Download

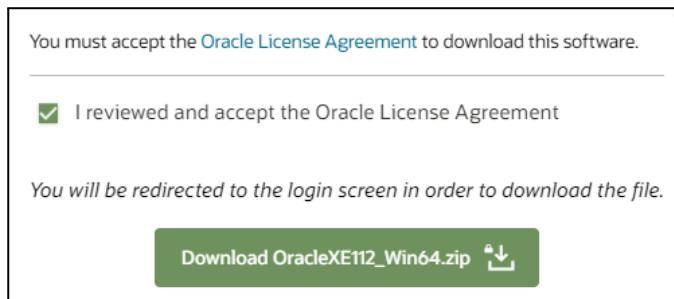
Oracle Database 21c Express Edition for Windows x64

(1,967,615,483 bytes)
[Sha256sum:
939742c3305c40]

Oracle Database 21c Express Edition for Linux x64 (OL8)

(2,339,651,768 bytes)
[Sha256sum:
f8357b432de334]

최신 버전이 발매 되어 다를 수 있으나 책과 같은 버전이 존재 한다면 되도록 책과 같은 버전을 이용하고 없을 때만 최신 버전을 설치해서 따라해 보자.



왼쪽 중간에 체크 박스를 클릭하고 하단에 download 버튼을 클릭하면 오라클 데이터베이스를 다운로드 받을 수 있다. 회원가입과 로그인을 해야 가능하다.

시도해 보자. 로그인 후에도 다운로드가 잘 안된다면 브라우저를 모두 닫은 후 다시 시도해 보자. 로그인 성공 후에도 잘못된 페이지로 이동할 수 있다. 사이트 자체에 버그가 많다.

검색과 설치가 잘 안되면 웹에 oracle 21c로 검색해서 설치 방법을 확인 하여 설치해 보자. 이미지가 18버전으로 되어 있는데 21c와 차이 없이 동일하다.



다운로드가 끝나고 압축을 풀면 setup.exe 파일이 있는데 클릭하면 오라클을 설치할 수 있다. 왼쪽 이미지처럼 창이 뜨면 라디오 버튼(동의함)을 클릭하고 next 버튼을 눌러 진행을 계속한다. 이후 질문에 긍정적인 답변으로 next를 누르며 설치를 계속 이어 나가면 된다.

다음은 system계정의 암호를 설정하는 창이다. 설치후 아이디 system 비밀번호 human 으로 로그인 할 수 있도록 설정 할 예정이다.

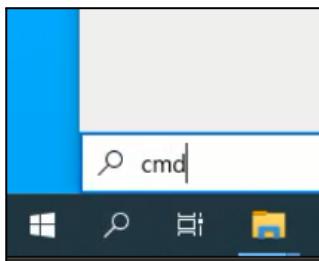


왼쪽 이미지는 관리자 계정의 암호를 입력하는 부분이다. human/human을 입력해서 암호를 human으로 설정 하고 설치를 이어 나가자.

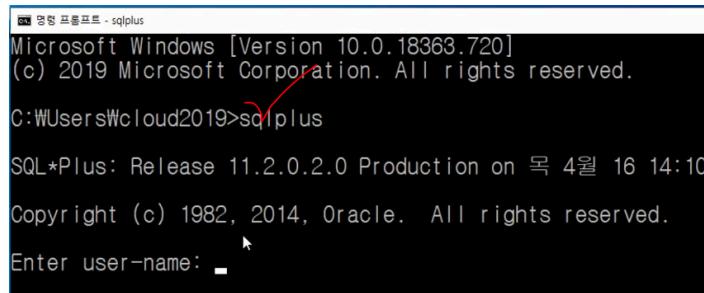
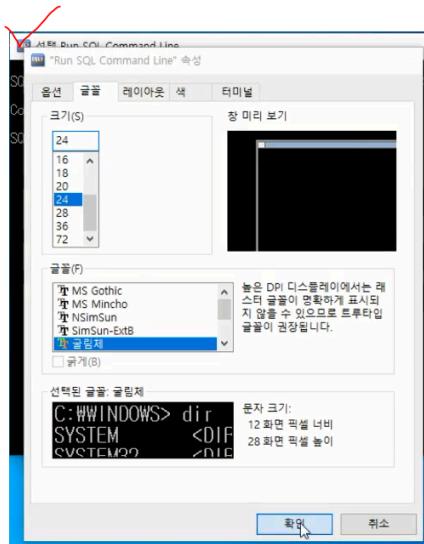
설치 중간에 browser 버튼을 눌러서 새로운 설치 경로를 지정 할 수 있다. 되도록 기본 위치에 설치하고 다른 드라이브에 설치 하려면 팝업으로 뜬 창에 설치할 폴더 위치를 기술하고 확인 버튼을 누르면 된다.

설치 중간에 네트워크 관련 팝업창이 뜨는데 반드시 허용하는 형태로 클릭 해야 이후 작업이

문제 없이 진행된다.



설치가 종료되고 윈도우키+R를 누르고 cmd를 입력하면 검정색 커맨드 창이 뜨는데 콘솔창이라고 부른다. sqlplus를 입력하고 엔터를 치면 우리가 설치한 데이터베이스 프로그램과 연결할 수 있는 sqlplus 프로그램이 실행된다. 윈도우 하단 돋보기 모양을 클릭하고 cmd를 입력 하여도 된다.



왼쪽 이미지 처럼 빨간색 체크부분 왼쪽 상단 이미지를 클릭하고 속성을 선택하면 글꼴이나 폰트 크기를 설정할 수 있다. 변경후 cmd창을 닫은 다음에 다시 열어야 설정한 내용이 적용된다. 검정색 콘솔창에서 작업하는 일이 쉽지 않으면 메모장에 기술 하였다가 복사 붙여넣기 해서 사용하자.

메모장에서 원하는 문자열을 작성 한 다음 shift+방향키로 작성한 문자열을 선택한 다음 ctrl+c(복사)로 선택한 문자열을 복사 한 다음 콘솔 창을 클릭한 후 ctrl+v(붙여넣기)하면 메모장에 작성한 문자열을 콘솔창에 붙여넣기 할 수 있다.

cmd 콘솔창에서 이전에 입력한 문자열을 다시 사용하고 싶다면 방향키를 이용하여 이전에 입력한 문자열을 찾을 수 있고 찾은 문자열을 방향키를 이용해서 편집할 수도 있다. 위쪽 방향키를 누르면 이전에 입력하여 실행한 문자열이 콘솔창에 생성되고 왼쪽 오른쪽 화살표를 이용하여 원하는 위치를 찾아가 수정할 수 있다.

sqlplus는 설치한 오라클 데이터베이스에 sql를 전달해서 데이터베이스를 조작하고 원하는 데이터를 사용할 수 있도록 제공해 주는 프로그램이다. 이 프로그램을 통해서 나중에 배울 sql로 데이터베이스에 테이블을 만들어 데이터를 저장하고 읽어올 수 있다.

sqlplus 자체가 데이터 베이스가 아니고 데이터베이스에게 사용자가 원하는 명령어를 편집해서 전달해주고 값을 받아 올 수 있는 db 관리 프로그램이다. 오라클이 설치 되면 오라클에 직접 접근 할 수 없고 관리 도구를 이용해서 접근해야 한다. sqlplus는 여러 관리 도구중 하나이다.

웹사이트에서 물건을 구매한다고 생각해 보자. 웹사이트 쇼핑몰에서 원하는 물건을 요청하면 해당 웹사이트에서 물류 창고에 있는 물건을 본인에게 전달해 준다. 여기서 물류 창고가 데이터베이스 서버에 해당되고 쇼핑몰이 sqlplus에 해당 된다.

sqlplus가 실행되면 사용자 아이디와 암호를 입력해 로그인 해야 데이터 베이스 오라클을 사용 할 수 있다. 아이디가 없거나 아이디와 암호를 모른다면 다음 아이디 없이 로그인 한 다음 아이디 계정을 만들 수 있는 권한인 sys 권한으로 로그인한 다음 새로운 계정을 만들어 새로운 계정을 사용할 수 있다. 다음을 따라서 계정을 만들어 데이터베이스를 사용해 보자.

1. 아래 처럼 명령 프롬프트가 실행된 상태에서 sqlplus /nolog를 입력하고 엔터를 치면 로그인 없이 sqlplus에 들어가 보자.

```
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Users\foxman12>sqlplus /nolog
```

```
Copyright (c) 1982, 2018
SQL> conn sys as sysdba
비밀번호 입력:
연결되었습니다.
SQL> show user;
USER은 "SYS"입니다
SQL>
```

2. sqlplus에 로그인 하지 않고 접속한 다음 왼쪽 처럼 conn sys as sysdba를 입력하면 sys계정으로 로그인할 수 있다. 중간에 password를 물으면 그냥 엔터를 치면 connected라는 메시지와 함께 로그인 된다. as sysdba를 통해서 이미 암호를 입력한 상태이기 때문이다. 제대로 로그인 되었는지 확인하고 싶다면 show user를 입력 하고 엔터를 치면 현재 유저가 sys라는 메시지를 확인할 수 있다. sys계정은 데이터 베이스 최상위 관리자 계정이다. 새로운 계정을 만들기 위해서 sys계정으로 로그인 하였다. 혹시 sys as sysdba로 로그인이 불가능 하다면 오라클 설치할 때 넣었던 암호가 system의 암호이다. 아이디 system 비번 human으로 sys대신 system계정을 사용해서 작업을 진행 할 수 있다.

3. sys계정으로 로그인 하였다면 create user c##다음에 원하는 계정명(c##human) identified by 암호(human)를 입력 하여 아이디 c##human, 암호 human인 계정을 만들어 보자. c##은 오라클 12g 버전이후 부터 반드시 넣도록 되어 있다. 모든 사용자 계정 앞에 c##를 붙여야 사용할 수 있다.

```
SQL> create user c##human identified by human;
```

같은 이름으로 하나의 계정만 만들 수 있다. 같은 이름으로 또 만들려고 하면 사용자 를 이름과 상충됩니다. 라는 에러 메시지가 나온다.

```
SQL> create user c##human identified by human;
create user c##human identified by human
*
1행에 오류:
ORA-01920: 사용자명 'C##HUMAN'(이)가 다른 사용자나 를 이름과 상충
됩니다
```

4. 정상적으로 작업을 하였는데 만든 계정으로 로그인을 하면 에러 메시지를 확인 할 수 있다. 로그인 할 수 없는 이유는 계정만 만들었을 뿐 사용 권한을 주지 않아서 이다. c##human계정에 connect 로그인 연결을 할 수 있는 권한, resource 데이터베이스 자원을 사용할 수 있는 권한, dba 데이터베이스 관리 할 수 있는 권한을 부여하여야 한다.

grant를 사용하여 사용자에게 권한을 부여 할 수 있다

sys로 로그인 한 다음 grant 권한 to 계정을 이용해서 human계정에 권한을 부여하고 다시 로그인 하면 다음처럼 c##human으로 로그인 된것을 확인 할 수 있다.

grant connect,resource,dba to c##human; ,(쉼표) .(마침표) :(콜론) ;(세미콜론)이 잘 식별이 되지 않는다. 자세히 구분해 식별해 보자.

```
SQL> grant connect,resource,dba to c##human;  
권한이 부여되었습니다.
```

sys계정에서 왼쪽처럼 권한을 부여 해야 c##human 계정으로 로그인 할 수 있다.

```
SQL> conn  
사용자명 입력: c##human  
비밀번호 입력:  
연결되었습니다.▶  
SQL>
```

5. conn c##human/human으로 로그인 하여 보자. 성공적으로 계정이 만들어 진것을 확인할 수 있다. 한번에 입력할 수 있고 왼쪽처럼 conn 엔터 c##human 엔터 human처럼 엔터를 치면서 순서대로 하나씩 입력 하여도 된다.

```
SQL> conn human/human  
ERROR:  
ORA-01017: 사용자명/비밀번호가 부적합.  
  
경고: 이제는 ORACLE에 연결되어 있지 않습니다.  
SQL> show user:  
USER은 ""입니다
```

만약에 로그인에 실패하거나 명령어에 실패하면 자동 로그 아웃 된다. 입력을 잘못 하였다면 show user를 입력하여 결과를 확인해 보면 아까는 sys였는데 지금은 “ ”으로 로그아웃 되어 있는 것을 확인 할 수 있다. 로그인후 사용하다가 문제가 발생하면 자동으로 로그아웃 되니 정상적으로 실행한

명령어가 동작하지 않으면 show user를 사용하여 로그인 상태인지 확인해 로그아웃 상태이면 다시 로그인 해야 한다.

```
SQL> select * from tab;  
선택된 레코드가 없습니다.
```

6. 로그인이 완료되면 select * from tab;를 입력하고 엔터를 쳐서 왼쪽과 같이 나오면 새로운 계정을 만들어 실행하는 작업이 완료된 것이다. select * from tab;은 만든 테이블을 모두 출력하라는 내용이다. 계정을 새로 만들면 새로운 테이블이 없어서 레코드가 없다고 뜬것이다.

혹시 sql를 종료하고 싶다면 exit를 입력하면 된다.

콘솔창은 컴퓨터 환경에서 사용자와 시스템 간 상호작용을 할 수 있는 텍스트 기반의 인터페이스를 제공하는 창 또는 화면입니다. 명령프롬프트(cmd)와 sqlplus 같은 검은 형태 창을 콘솔창이라고 한다.

이전에 만든 계정의 데이터 베이스의 암호가 만기되어 로그인 할 수 없으면 alter user 계정명 identified by 새비밀번호로 비밀 번호를 변경하면 해결 된다.

ex) alter user c##hr identified by hr -- c##hr 계정의 비밀번호를 hr로 변경하였다.

오라클을 설치하고 계정을 만들어 로그인하는 방법을 다음과 같이 정리하였다.

1. www.oracle.com 사이트에 방문해서 oracle ex를 다운로드 받아 설치한다.

2. cmd창에 sqlplus /nolog를 입력해서 로그인 없이 sqlplus에 접속한다.
3. 계정을 만들수 있는 권한이 있는 sys계정으로 로그인한다. conn sys as sysdba;
4. 만들고 싶은 계정을 생성한다. create user c##human identified by human;
5. 만든 계정에 권한을 설정한다. grant connect,resource,dba to c##human;
6. 만든 계정으로 로그인한다. conn c##human/human;
7. 로그인 되었는지 확인해본다. show user;
8. 정상적으로 작업이 되었는지 확인을 위해서 존재하는 테이블이 있는지 확인해 본다.
select * from tab;

연습문제

1. 본인 이름의 이니셜로 계정을 만드는 실습을해 보자.
 2. 로그인 없이 sqlplus에 접속하는 옵션은 무엇인가?
 3. 특정 계정에 권한 부여하는 명령어는 무엇인가?
 4. 새로운 계정으로 다시 로그인 할때 사용하는 명령어는 무엇인가?
 5. 테이블 만들 때 사용한 명령어와 계정 만들 때 사용하는 명령어는 동일하다. 계정을 삭제하고자 한다면 어떤 명령어를 사용해야 하는가?
 6. sqlplus와 oracle db와 관계를 설명해 보자.
- 명확하게 기억이 나지 않을 때 키워드를 웹에서 검색하는 습관을 기르자.

> 02. `LocalDateTime` 사용법

주요 시간 클래스

1. `LocalDate`: 날짜만 표현 (예: 2025-01-19).
2. `LocalTime`: 시간만 표현 (예: 14:30:00).
3. `LocalDateTime`: 날짜와 시간을 함께 표현 (예: 2025-01-19T14:30:00).
4. `ZonedDateTime`: 날짜와 시간을 포함하고 시간대 정보를 추가로 포함 (예: 2025-01-19T14:30:00+09:00[Asia/Seoul]).

```
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.LocalDateTime;
import java.time.ZonedDateTime;
import java.time.ZoneId;

public class Main {
    public static void main(String[] args) {
        // 현재 날짜
        LocalDate today = LocalDate.now();
        System.out.println("현재 날짜: " + today);

        // 현재 시간
```

```

LocalTime nowTime = LocalTime.now();
System.out.println("현재 시간: " + nowTime);

// 현재 날짜와 시간
LocalDateTime nowDateTime = LocalDateTime.now();
System.out.println("현재 날짜와 시간: " + nowDateTime);

// 현재 날짜와 시간 (시간대 포함)
ZonedDateTime zonedNow = ZonedDateTime.now(ZoneId.of("Asia/Seoul"));
System.out.println("현재 서울 시간: " + zonedNow);

}
}

```

2. 특정 날짜와 시간 생성

```

import java.time.LocalDate;
import java.time.LocalTime;
import java.time.LocalDateTime;

public class Main {
    public static void main(String[] args) {
        // 특정 날짜 생성
        LocalDate specificDate = LocalDate.of(2023, 12, 25);
        System.out.println("특정 날짜: " + specificDate);
    }
}

```

```
// 특정 시간 생성  
  
LocalTime specificTime = LocalTime.of(14, 30, 15);  
  
System.out.println("특정 시간: " + specificTime);  
  
  
// 특정 날짜와 시간 생성  
  
LocalDateTime specificDateTime = LocalDateTime.of(2023, 12, 25,  
14, 30, 15);  
  
System.out.println("특정 날짜와 시간: " + specificDateTime);  
  
}  
  
}
```

3. 문자열로부터 시간 데이터 생성

`DateTimeFormatter`를 사용하여 문자열을 파싱하고 시간 데이터를 생성합니다.

java

복사편집

```
import java.time.LocalDateTime;  
  
import java.time.format.DateTimeFormatter;  
  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        // 문자열을 LocalDateTime으로 변환  
  
        String dateTimeString = "2025-01-19T14:30:00";  
  
        LocalDateTime dateTime = LocalDateTime.parse(dateTimeString);  
  
        System.out.println("파싱된 날짜와 시간: " + dateTime);  
  
  
        // 사용자 정의 포맷으로 변환  
  
        DateTimeFormatter formatter =
```

```
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    String customDateTimeString = "2025-01-19 14:30:00";

    LocalDateTime customDateTime =
LocalDateTime.parse(customDateTimeString, formatter);

    System.out.println("사용자 정의 포맷 파싱: " + customDateTime);

}

}
```

시간을 원하는 문자열로 출력하기

Java에서 `LocalDateTime` 데이터를 원하는 형식으로 출력하려면 `DateTimeFormatter` 클래스를 사용하면 됩니다.

📌 기본적인 날짜 및 시간 포맷팅

```
import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class Main {

    public static void main(String[] args) {

        // 현재 날짜 및 시간

        LocalDateTime now = LocalDateTime.now();

        // 원하는 형식 지정

        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

        // 포맷 적용
```

```

        String formattedDateTime = now.format(formatter);

        // 출력
        System.out.println("포맷된 날짜 시간: " + formattedDateTime);
    }

}

```

출력 예시

yaml

복사편집

포맷된 날짜 시간: 2025-02-08 14:30:45

다양한 포맷 예제

패턴 문자열	설명	예제 출력
yyyy-MM-dd	연-월-일	2025-02-08
yyyy/MM/dd	연/월/일	2025/02/08
MM-dd-yyyy	월-일-연	02-08-2025
dd MMM yyyy	일 월 연	08 Feb 2025
E, MMM dd yyyy	요일, 월 일 연	Sat, Feb 08 2025
HH:mm:ss	시:분:초 (24시간제)	14:30:45
hh:mm a	시:분 AM/PM (12시간제)	02:30 PM

yyyy-MM-dd	밀리초 포함	2025-02-08
HH:mm:ss.SSS		14:30:45.123

📌 특정 날짜/시간 포맷 변환

문자열을 `LocalDateTime`으로 변환하는 경우 `parse()`를 사용합니다.

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Main {

    public static void main(String[] args) {
        String dateTimeStr = "2025-02-08 14:30:45";

        // 기존 문자열의 포맷 지정
        DateTimeFormatter formatter =
        DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

        // 문자열을 LocalDateTime으로 변환
        LocalDateTime dateTime = LocalDateTime.parse(dateTimeStr,
        formatter);

        // 변환된 객체 출력
        System.out.println("LocalDateTime: " + dateTime);
    }
}
```

출력 예시

yaml

복사편집

```
LocalDateTime: 2025-02-08T14:30:45
```

커스텀 포맷으로 변환

예를 들어, 기존 `yyyy-MM-dd HH:mm:ss` 형식을 `E MMM dd, yyyy hh:mm a`로 변경하려면:

java

복사편집

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Main {
    public static void main(String[] args) {
        String dateTimeStr = "2025-02-08 14:30:45";
        // 기존 포맷
        DateTimeFormatter inputFormatter =
        DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        // 새 포맷
        DateTimeFormatter outputFormatter =
        DateTimeFormatter.ofPattern("E MMM dd, yyyy hh:mm a");
        // 문자열을 LocalDateTime으로 변환 후 새로운 포맷 적용
```

```
    LocalDateTime dateTime = LocalDateTime.parse(dateTimeStr,  
inputFormatter);  
  
    String formattedDate = dateTime.format(outputFormatter);  
  
    // 출력  
  
    System.out.println("새로운 포맷: " + formattedDate);  
}  
}
```

출력 예시

yaml

복사편집

새로운 포맷: Sat Feb 08, 2025 02:30 PM

시간대(TimeZone) 포함 변환

`LocalDateTime`은 시간대 정보를 포함하지 않기 때문에, `ZonedDateTime`을 사용하여 특정 시간대(예: `Asia/Seoul`)를 포함할 수도 있습니다.

java

복사편집

```
import java.time.LocalDateTime;  
  
import java.time.ZoneId;  
  
import java.time.ZonedDateTime;  
  
import java.time.format.DateTimeFormatter;  
  
  
public class Main {
```

```
public static void main(String[] args) {  
    LocalDateTime now = LocalDateTime.now();  
    ZonedDateTime seoulTime = now.atZone(ZoneId.of("Asia/Seoul"));  
  
    // 포맷 지정  
    DateTimeFormatter formatter =  
    DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss z");  
  
    // 포맷 적용  
    String formattedDateTime = seoulTime.format(formatter);  
  
    // 출력  
    System.out.println("서울 시간대 포맷: " + formattedDateTime);  
}  
}
```

▣ 출력 예시

yaml

복사편집

서울 시간대 포맷: 2025-02-08 14:30:45 KST

◆ 정리

- `DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")` → 원하는 날짜/시간 형식으로 변환 가능.
- `format()` → `LocalDateTime`을 문자열로 변환.
- `parse()` → 문자열을 `LocalDateTime`으로 변환.

- 다양한 패턴을 활용하여 원하는 방식으로 출력 가능.
- 시간대를 포함하려면 `ZonedDateTime` 사용.

이제 원하는 날짜/시간 형식으로 자유롭게 변환할 수 있습니다! 

2. 날짜와 시간 정보 추출

`LocalDateTime` 객체에서 날짜와 시간 정보를 개별적으로 추출할 수 있습니다.

java

복사편집

```
import java.time.LocalDateTime;

public class Main {

    public static void main(String[] args) {
        LocalDateTime dateTime = LocalDateTime.now();

        int year = dateTime.getYear();
        int month = dateTime.getMonthValue(); // 월 (1~12)
        int day = dateTime.getDayOfMonth();
        int hour = dateTime.getHour();
        int minute = dateTime.getMinute();
        int second = dateTime.getSecond();

        System.out.println("년도: " + year);
        System.out.println("월: " + month);
        System.out.println("일: " + day);
        System.out.println("시: " + hour);
        System.out.println("분: " + minute);
        System.out.println("초: " + second);
    }
}
```

```
}
```

```
}
```

날짜와 시간 연산

`LocalDateTime`은 날짜와 시간에 대해 다양한 연산을 제공합니다.

```
import java.time.LocalDateTime;

public class Main {

    public static void main(String[] args) {
        LocalDateTime dateTime = LocalDateTime.now();
        System.out.println("현재 날짜와 시간: " + dateTime);

        // 10일 뒤로 이동
        LocalDateTime plusDays = dateTime.plusDays(10);
        System.out.println("10일 뒤: " + plusDays);

        // 3시간 전으로 이동
        LocalDateTime minusHours = dateTime.minusHours(3);
        System.out.println("3시간 전: " + minusHours);

        // 2개월 뒤로 이동
        LocalDateTime plusMonths = dateTime.plusMonths(2);
        System.out.println("2개월 뒤: " + plusMonths);
    }
}
```

5. 비교 및 검사

`LocalDateTime` 객체는 날짜와 시간을 비교하거나 특정 조건을 확인하는 메서드를 제공합니다.

java

복사편집

```
import java.time.LocalDateTime;

public class Main {
    public static void main(String[] args) {
        LocalDateTime dateTime1 = LocalDateTime.of(2025, 1, 19, 14, 30);
        LocalDateTime dateTime2 = LocalDateTime.of(2023, 12, 25, 10, 0);

        // 비교
        boolean isAfter = dateTime1.isAfter(dateTime2);
        boolean isBefore = dateTime1.isBefore(dateTime2);

        System.out.println("dateTime1이 dateTime2 이후인가? " + isAfter);
        System.out.println("dateTime1이 dateTime2 이전인가? " + isBefore);
    }
}
```

localDateTime과 다른 클래스 간 변환

다른 시간 관련 클래스와 변환이 가능합니다.

(1) `LocalDateTime` → `LocalDate` / `LocalTime`

```
java
복사편집

import java.time.LocalDateTime;
import java.time.LocalDate;
import java.time.LocalTime;

public class Main {
    public static void main(String[] args) {
        LocalDateTime dateTime = LocalDateTime.now();

        LocalDate date = dateTime.toLocalDate();
        LocalTime time = dateTime.toLocalTime();

        System.out.println("날짜: " + date);
        System.out.println("시간: " + time);
    }
}
```

(2) **LocalDateTime** → **ZonedDateTime**

```
java
복사편집

import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;

public class Main {
```

```
public static void main(String[] args) {  
    LocalDateTime dateTime = LocalDateTime.now();  
  
    ZonedDateTime zonedDateTime =  
dateTime.atZone(ZoneId.of("Asia/Seoul"));  
  
    System.out.println("시간대 포함 날짜와 시간: " + zonedDateTime);  
}  
}
```

년월일 시분초 차이구하는 프로그램

```
import java.time.Duration;  
import java.time.LocalDateTime;  
import java.time.Period;  
  
public class TimeDifferenceCalculator {  
    public static void main(String[] args) {  
        // 두 개의 LocalDateTime 값 직접 설정  
        LocalDateTime time1 = LocalDateTime.of(2024, 2, 5, 14, 30, 0);  
        // 2024년 2월 5일 14:30:00  
  
        LocalDateTime time2 = LocalDateTime.of(2024, 3, 8, 16, 45, 30);  
        // 2024년 3월 8일 16:45:30  
  
        // 두 시간의 차이 계산  
        printTimeDifference(time1, time2);  
    }  
}
```

```
// 두 시간의 차이 계산 및 출력 함수

private static void printTimeDifference(LocalDateTime start,
LocalDateTime end) {

    Period period = Period.between(start.toLocalDate(),
end.toLocalDate());

    Duration duration = Duration.between(start, end);

    long totalHours = duration.toHours();

    long totalMinutes = duration.toMinutes();

    long totalSeconds = duration.getSeconds();

    long hours = duration.toHoursPart();

    long minutes = duration.toMinutesPart();

    long seconds = duration.toSecondsPart();

    System.out.println("날짜 차이: " + period.getYears() + "년 " +
period.getMonths() + "개월 " + period.getDays() + "일");

    System.out.println("시간 차이: " + hours + "시간 " + minutes + "분
" + seconds + "초");

    System.out.println("총 시간 차이: " + totalHours + "시간");

    System.out.println("총 분 차이: " + totalMinutes + "분");

    System.out.println("총 초 차이: " + totalSeconds + "초");

}
```

실행 결과

복사편집

날짜 차이: 0년 1개월 3일

시간 차이: 2시간 15분 30초

총 시간 차이: 738시간

총 분 차이: 44355분

총 초 차이: 2661330초

일 차이가 1이 아닌 경우 만들기

```
import java.time.Duration;
import java.time.LocalDateTime;
import java.time.Period;

public class TimeDifferenceCalculator {
    public static void main(String[] args) {
        // 날짜는 하루 차이지만 시간 차이로 인해 일 차이가 1이 되지 않음
        LocalDateTime time1 = LocalDateTime.of(2024, 2, 5, 23, 59, 59);
        LocalDateTime time2 = LocalDateTime.of(2024, 2, 6, 00, 00, 00);

        printTimeDifference(time1, time2);
    }
}
```

```
// 두 시간의 차이 계산 및 출력 함수

private static void printTimeDifference(LocalDateTime start,
LocalDateTime end) {

    Period period = Period.between(start.toLocalDate(),
end.toLocalDate());

    Duration duration = Duration.between(start, end);

    long hours = duration.toHoursPart();

    long minutes = duration.toMinutesPart();

    long seconds = duration.toSecondsPart();

    System.out.println("날짜 차이 (Period 기준): " + period.getYears()
+ "년 " + period.getMonths() + "개월 " + period.getDays() + "일");

    System.out.println("시간 차이 (Duration 기준): " + hours + "시간 "
+ minutes + "분 " + seconds + "초");

}
```

실행 결과

scss

복사편집

날짜 차이 (Period 기준): 0년 0개월 0일

시간 차이 (Duration 기준): 0시간 0분 1초

→ 2024년 2월 5일 23:59:59 → 2024년 2월 6일 00:00:00 은 시간 차이는 있지만 날짜 차이는 없음!

→ `Period`는 날짜 차이만 보기 때문에 **0일**

→ `Duration`은 **1초 차이 발생**

✓ 날짜 차이가 1일이 되는 경우

```
LocalDateTime time1 = LocalDateTime.of(2024, 2, 5, 00, 00, 00);
```

```
LocalDateTime time2 = LocalDateTime.of(2024, 2, 6, 00, 00, 00);
```

이렇게 하면 `Period` 차이가 1일이 됩니다.

✓ 정리

- `Period.between()` → 연, 월, 일 차이만 계산 (시간 무시)
- `Duration.between()` → 시간까지 포함한 차이 계산
- 23:59:59에서 00:00:00으로 가면 날짜 차이가 0일
- 00:00:00에서 00:00:00으로 가야 날짜 차이가 1일

이제 `Period`와 `Duration`의 차이를 이해했을 겁니다! 

> 02. sql developer 설치하기

SQL Developer는 SQLPlus와 같은 작업을 하는 프로그램인데 GUI 그래픽 기반의 프로그램으로, 직관적이고 시각적으로 데이터베이스와 상호 작용할 수 있는 도구입니다.

sql developer 를 다운로드하여 프로그램을 설치해보고 sqlplus와 sql developer를 이용하여 scott계정과 hr계정을 사용 할 수 있도록 세팅해 보자.



www.oracle.com에 방문해서 sql developer download 를 검색어로 넣고 검색한 결과를 아래로 드래그 하면 sql Developer를 다운로드 받을 수 있는 링크가 제공 된다.

다음 이미지 처럼 제공 되는 링크를 클릭해서 들어가면 sql developer를 다운로드 받을 수 있는 버튼을 확인 할 수 있다.

Oracle SQL Developer Downloads
<https://www.oracle.com/database/sqldeveloper/technologies/download/>
Oracle SQL Developer Downloads

SQL Developer 23.1.1
Version 23.1.1.345.2114 - December 15, 2023
• Release Notes
• Documentation
• Forum

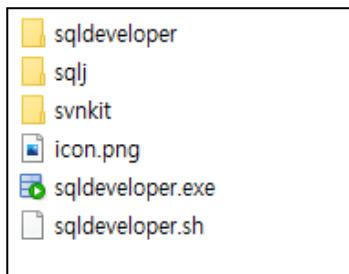
Platform	Download
Windows 64-bit with JDK 11 included	Download (465 MB)

링크를 클릭해 들어가 본인에 맞는 sql developer 를 다운로드해 보자.

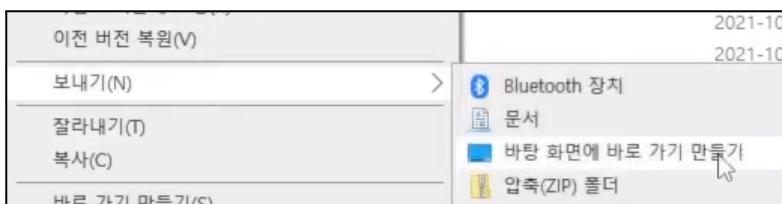
우리가 필요한 제품은 windows 64 JDK 11 included 버전이다. 해당 제품 download 버튼을 눌러 다운로드 하자.

다운로드 완료후 완료된

파일에 마우스를 올려놓고 마우스 오른쪽 버튼을 눌러 압축 풀기를 하면 다음과 같은 폴더안에 파일을 확인 할 수 있다.



실행방법은 압축 풀더안의 `sqldeveloper.exe`를 클릭하여 실행 시키면 `sqldeveloper`를 실행시킬 수 있다. 실행 도중에 나오는 메시지 박스들을 계속 진행시키면 문제 없이 실행이 될 것이다. 바로 가기를 바탕 화면에 만들고 사용하자.
`sqldeveloper.exe` 파일 위에서 마우스 오른쪽 클릭 >> 보내기 >> 바탕화면에 바로가기 만들기를 클릭하여 바로가기 아이콘을 바탕화면에 만들어 사용하자.

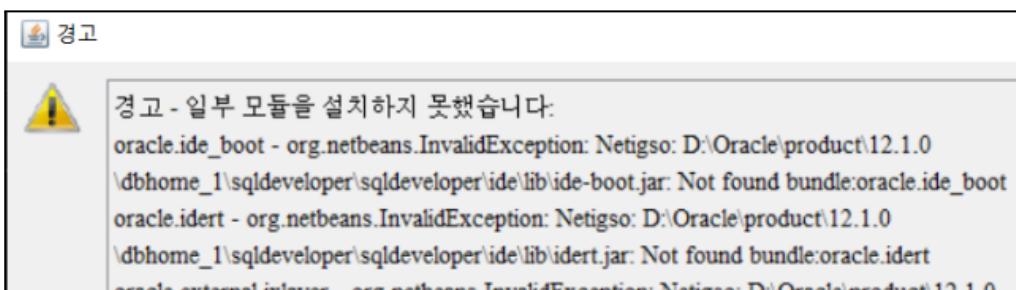


아이콘을 더블 클릭하여 실행하고 나면 다음 이미지 처럼 접속창이 보일 것이다. 접속창이 없다면 메뉴에 보기>>접속을 클릭하면 접속창이 열린다.



접속창 상단에 녹색 + 버튼을 누르면 오라클 데이터베이스에 접근할 수 있는 계정 등록하는 새로 만들기 창이 뜬다.

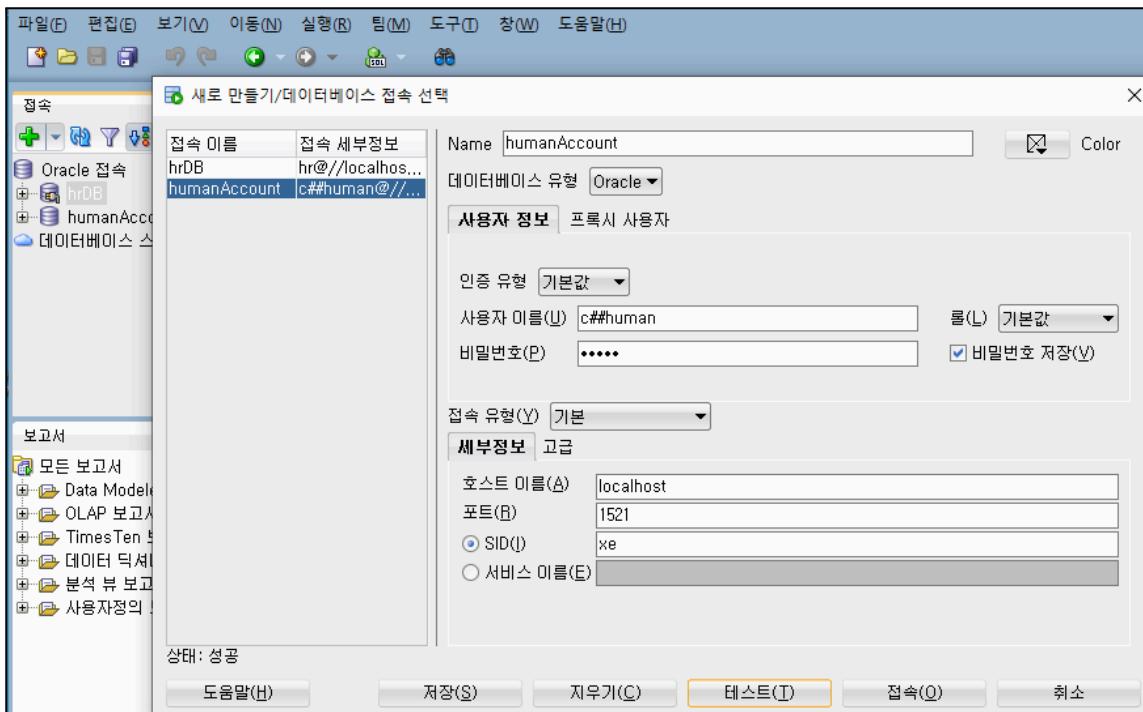
만약 실행시 다음과 같은 화면이 나오고 실행되지 않을 때는 C:\Users\사용자명\AppData\Roaming\SQL Developer 폴더 삭제 및 하위 파일을 제거 하고 다시 실행해 보자.



다음 페이지에 있는 이미지 처럼 새로 만들기 창이 뜨면 'human 계정 정보'를 name, 사용자이름, 비밀번호, 비밀번호저장에 입력 한 다음 접속 버튼을 눌러서 데이터베이스 human 계정 사용자를 등록 할 수 있다.

이전에 `sqlplus`에서 만든 `human`계정을 기반으로 `sql developer`에 등록 하는 예제를 따라해 보면 다음과 같다. 입력 부분 Name은 프로그램에서 식별 할때 사용 할 식별자를

넣는다. 보통 계정이랑 같은 이름으로 만든다. 여기서는 `humanAccount`로 만들었고 이름과 비밀번호를 `c##human/ human`으로 넣는다. 비밀번호 저장을 체크한다. 호스트이름에는 `oracle`이 설치된 ip주소를 입력한다. 현재 같은 컴퓨터에 `oracle`이 설치되어 있다면 `localhost`를 입력하고 포트는 기본으로 설치 하였다면 `1521`이고 변경하였다면 변경한 포트번호를 입력하면 된다. `sid`는 본인이 변경하지 않았다면 `xe`이고 2개 이상 설치 하거나 변경 하였다면 변경한 이름을 넣자. 하단 테스트 버튼을 눌러 확인해 보면 문제가 없으면 성공이라는 문자열이 왼쪽 하단 부분 상태창 부분에 출력 된다. 문제가 발생해서 에러 메시지가 생기면 해당 에러 메시지를 웹에 검색해서 문제점을 확인하여 문제를 없애 보자. 모두 완료되었다면 접속 버튼을 누르자.



호스트 이름 (Host Name):

호스트 이름은 컴퓨터나 장치를 식별하는 데 사용되는 읽기 쉬운 문자열입니다.

예를 들어, "www.example.com"은 웹 서버의 호스트 이름입니다.

IP 주소 (Internet Protocol Address):

IP 주소는 컴퓨터나 네트워크 장치를 고유하게 식별하는 숫자입니다.

IPv4 주소는 보통 "`xxx.xxx.xxx.xxx`" 형식으로 표현되며, 각 부분은 0부터 255까지의 숫자입니다.

예를 들어, "`192.168.0.1`"은 일반적인 로컬 네트워크에서 사용되는 IP 주소입니다.

"localhost"는 현재 사용 중인 컴퓨터를 가리키는 호스트 이름입니다.

자기 자신을 가리키기 위해 사용되며, 일반적으로 루프백 주소인 "127.0.0.1"로 매핑됩니다. 매핑이란? 데이터를 한 형태에서 다른 형태로 변환하는 것을 나타낼 수 있습니다. 변환전과 변환후를 1:1로 매핑한다고 이야기합니다.

상위 호스트 이름 입력부분에 localhost 대신에 IP주소를 넣어도 된다.

DNS(도메인 네임 시스템)은 컴퓨터 네트워크에서 사용되는 서비스로, 사람이 이해하기 쉬운 도메인 이름(예: www.example.com)을 컴퓨터가 이해할 수 있는 IP 주소(예: 192.168.0.1)로 변환하는 역할을 합니다. 네이버 사이트 접속시 124.12.132.12 이라 입력하지 않고 주소를 넣어도 처리되는 이유는 dns서버가 www.naver.com이라는 문자열을 ip주소로 변환하여 상대방 컴퓨터를 찾아 갈 수 있다.

포트 (Port):

포트는 네트워크 연결에서 서비스를 식별하기 위한 숫자입니다.

ip로 상대방 컴퓨터까지 찾아오면 컴퓨터 안에 네트워크를 사용하는 여러 응용프로그램이 있다. 여러 응용프로그램중 누구에게 데이터를 줄것인가를 포트로 결정할 수 있다.

택배 보내면 주소와 받을 사람 이름을 쓰는데 주소는 ip, 포트는 받을 사람 이름이라고 생각하면 된다.

컴퓨터가 여러 응용 프로그램들이 동작 할 때, 각 응용 프로그램은 고유한 포트 번호를 사용합니다.

예를 들어, 웹 서버는 일반적으로 80번 포트를 사용하며, 보안 연결에는 443번 포트, 파일서버는 21, 오라클은 1521를 사용한다.

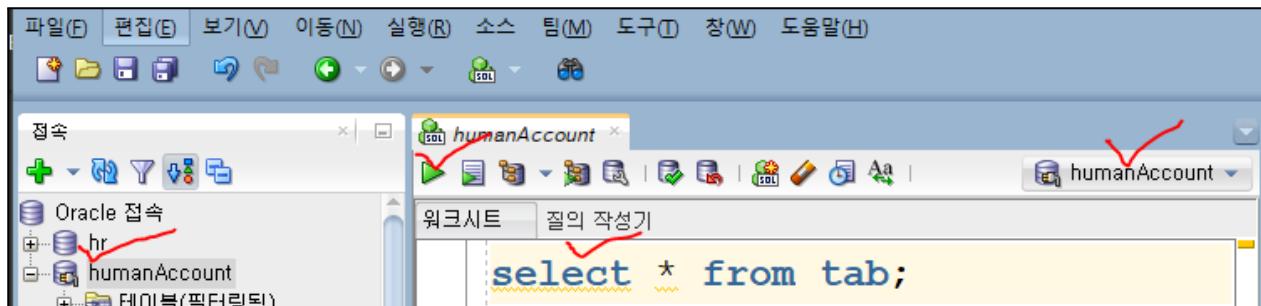
SID (System ID):

SID는 Oracle 데이터베이스를 한 컴퓨터에 여러개 설치 하였을때 각각의 데이터베이스를 식별하는 데 사용되는 고유한 이름 또는 식별자입니다. 본인이 변경하지 않았다면 일반적으로 xe라는 이름을 가지고 같은 이름으로 2개 이상 설치 할 수 없다. 설치시 마다 다른 이름으로 설치하여야 한다.

따라서, 호스트 이름, IP 주소, 포트, 그리고 SID는 데이터베이스 연결에 필요한 정보입니다. 이 정보를 사용하여 데이터베이스 클라이언트나 도구에서 데이터베이스 서버에

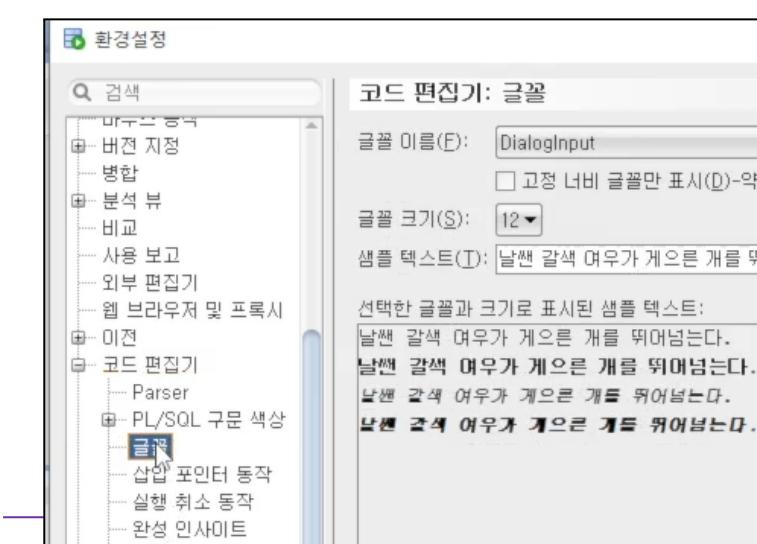
연결하여 sql을 이용해 원하는 데이터 처리가 가능하게 된다.

등록이 완료되면 아래 이미지 처럼 접속창에 등록된 아이디가 생긴다. 생성한 human Account를 더블 클릭하여 아이디와 패스워드를 입력 하면 sql을 작성 할 수 있는 창이 생긴다. 원하는 sql을 기술하고 기술한 내용을 실행시키고 싶다면 해당 sql을 마우스로 드래그한 다음 상단 화살표 모양인 실행 버튼을 누르면 선택한 부분의 쿼리문만 실행 된다.



그냥 녹색 화살표 모양 실행 버튼을 누르면 작성된 모든 sql이 실행된다. 실행하고 싶은 sql에 커서를 위치시키고 **ctrl+엔터**를 누르면 커서가 위치한 sql이 실행되는데 앞뒤 sql문이 제대로 종료되어 있지 않으면 문제가 발생한다. 가장 많이 발생하는 에러는 이전 sql이나 다음 sql 끝에 ;를 찍지 않고 sql를 실행 시킬때 발생한다. 에러가 발생하면 앞뒤 문맥에 문제가 없는지 확인하자.

오른쪽 상단에 접속된 사용자가 뜨고 사용자를 변경해서 작성한 sql을 실행하고 싶다면 클릭해서 사용자를 변경하면 된다.



메뉴에서 파일 > 저장 을 선택하거나 메뉴 밑에 디스켓 모양의 저장아이콘을 선택해서 작성한 sql문을 원하는 위치에 저장하였다가 다음에 불러와서 사용할 수 있다. 이전에 sqlplus로 하던 human 테이블 만드는 작업을 sql developer에서 해보자.

도구>>환경설정>>글꼴에서 원하는 폰트와 크기를 설정할 수 있다.

> 03. JDBC 사용하기

JDBC란? java database connectivity 약어로 자바에서 데이터베이스를 연결하여 사용하는 방법을 의미한다. 자바에서 데이터 베이스를 연결하여 사용하고 싶다면 데이터베이스 회사에서 제공하는 JDBC관련 JAR파일을 추가한 다음 jdbc API를 사용하여 프로그램을

구현하면 된다.

다음을 따라서 jdbc프로그램을 완성해 보자.

1. 다음 human table sql를 c##human/human 계정에 만들어 보자.

```
drop table human;

create table human(
    name nvarchar2(30), age number(3), height number(7,3), birthday date
);

insert into human(name,age,height,birthday)
values('김수호',20,160.4,to_date('2005:05:05 02:25:50','YYYY:MM:DD
HH24:MI:SS'));

insert into human(name,age,height,birthday)
values('나수호',24,170.8,to_date('2000:10:15 12:25:10','YYYY:MM:DD
HH24:MI:SS'));

insert into human(name,age,height,birthday)
values('박수호',27,188.6,to_date('1995:12:04 13:45:14','YYYY:MM:DD
HH24:MI:SS'));

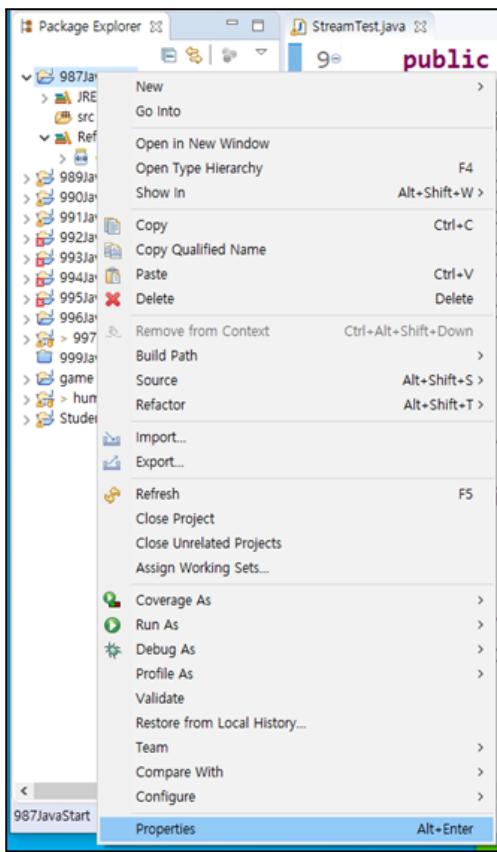
commit; -- 반드시 커밋을 해야 한다.

select * from human;
```

2. jdbc프로그램을 위한 OJDBC8_g.jar를 추가해 보자.

JDBC프로그램을 하려면 오라클에서 제공하는 OJDBC8_g.jar 파일이 있어야 한다.

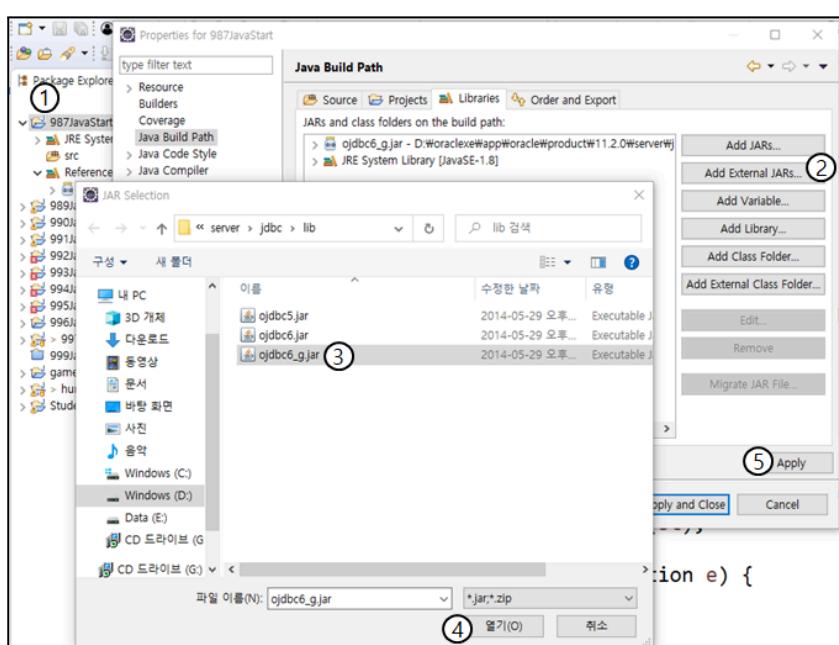




오라클이 설치되어 있다면 오라클이 설치되어 있는 상위 이미지의 폴더로 이동하여 ojdbc8_g.jar 파일을 찾아 사용하면 된다. 18.0.0은 설치한 버전이여서 본인이 어떤 버전을 설치하였는지에 따라 다를 수 있다. 실제로 프로젝트에서 사용하려면 eclipse를 실행하고 다음 이미지처럼 프로젝트 명에서 오른쪽 마우스 클릭을 한다음 properties를 선택하면 아래 이미지와 같은 창이 뜰 것이다. 이미지 안에서 1번 약간 오른쪽으로 Java Build Path 항목을 클릭 하고 오른쪽에 뜬 창 부분에서 Libraries 탭을 선택하면 외부 jar 파일을 등록 할 수 있는 버튼이 나온다. 이미지의 2번 위치에 add External jars에 해당 한다.

2번 버튼을 누르면 파일을 등록할 수 있는 창이 뜨는데 ojdbc8_g.jar 파일이 존재하는 폴더로 이동하여 3번처럼 파일을 찾을 수 있고 선택한 다음 4번 버튼을 누르면 파일을 등록 할 수 있다. 등록이 끝나면 5번 버튼을 눌러 적용시키면 해당 jar파일이 등록된 것을 확인할 수 있다. 등록 결과는 package

Explore 안에 Referenced Libraries 및에서 찾을 수 있다. 일부분은 아래 이미지 1번 밑으로 살짝 Referenced Libraries 항목이 보이는걸 확인 할 수 있다. 클릭해서 추가한 ojdbc8_g.jar 가 있는지 확인해 보자. 이렇게 하면 JDBC사용 준비가 모두 끝났다. 이후에는 평상시 프로그램하는 것과 동일하게 작업을 이어나가면 된다.



다음 코드를 com.human.ex.JdbcTest 클래스에 기술하고 실행하면 데이터 베이스의 내용을 확인 할 수 있다. 만약에 출력되지 않으면 에러 메시지를 확인해서 웹에 검색해서 처리해 보자.

```
package com.human.ex;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.LocalDateTime;
public class JdbcTest {
    public static void main(String[] args) {
        Connection conn=null;
        String sql=null;
        Statement st=null;
        ResultSet rs=null;
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            String url="jdbc:oracle:thin:@localhost:1521:xe";
            String user="c##human";
            String pw="human";
            conn=DriverManager.getConnection(url,user,pw);
            sql="select * from human";
            st=conn.createStatement();
            rs=st.executeQuery(sql);
            while(rs.next()) {
                String name =rs.getString("name");
                int age=rs.getInt("age");
                double height=rs.getDouble("height");
                LocalDateTime birthday=rs.getTimestamp("birthday").toLocalDateTime();
                System.out.println("name:"+rs.getString(1));
                System.out.println("age:"+rs.getInt(2));
                System.out.println("height:"+rs.getDouble(3));
                System.out.println("birth:"+rs.getTimestamp(4));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            try {
                if(rs!=null)rs.close();
                if(st!=null) st.close();
                if(conn!=null) conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
    }
}
```

JDBC프로그램 코드순서는 다음과 같다.

1. 사용 할 DB종류 선택 (우리는 오라클을 사용할 예정이다)

```
Class.forName("oracle.jdbc.driver.OracleDriver");//오라클 사용시 사용
```

```
Class.forName("com.mysql.jdbc.driver");//mysql사용시 사용
```

사용자가 어떤 데이터베이스 제품을 사용할 것인지 결정해서 해당 객체를 내부적으로 사용할 수 있도록 문자열로 생성하는 부분이다.

2. db 연결을 위한 정보 저장 객체 생성(Connection)

```
Connection con = DriverManager.getConnection(url,user,pw);
```

데이터 베이스 연결 정보를 가지고 해당 데이터베이스에 연결할 수 있는 정보를 가지는 객체 Connection을 생성한다.

3. 연결 정보 객체를 이용해서 DB에 연결된 상태인 객체(Statement)를 얻어 온다.

createStatement메소드를 이용하여 데이터를 주고 받을때 사용하는 Statement객체를 얻어 온다. 해당 인스턴스를 이용하여 데이터를 주고 받는다.

```
Statement st = con.createStatement();
```

4. 연결 상태 객체를 이용해서 원하는 데이터를 요청하여 요청 결과를 ResultSet에 담아 처리 한다. ResultSet rs = st.executeQuery(sql);

5. ResultSet의 .next메소드를 이용하여 select문을 이용해서 받아온 데이터를 출력한다.

```
while(rs.next()) {//다음 데이터가 있으면 이동하여 다음 데이터를 읽어 온다.
```

```
//인덱스는 1부터 시작한다. //인덱스는 컬럼 만들때의 순서이다.
```

```
rs.getInt("no");//컬럼명 또는 인덱스 숫자를 매개변수로 사용
```

```
int no=rs.getInt(1); String name=rs.getString(2);
```

```
double height=rs.getDouble("height");
```

```
LocalDateTime birthday=rs.getTimestamp("birthday").toLocalDateTime();
```

```
}
```

매개변수 문자열은 해당 데이터베이스의 테이블의 컬럼명으로 데이터를 읽어오는 것이다.

숫자는 해당 테이블을 만들때 기술한 컬럼 순서로 데이터를 읽어오는 것이다. 1이면

테이블 만들때의 첫번째 컬럼, 2 이면 테이블을 만들때의 두번째 컬럼에 해당한다.

자료형의 종류에 따라 getString, getInt, getDouble를 사용한다.

6. 사용한 자원을 생성한 역순으로 모두 반납한다.

```
If(rs!=null)rs.close();If(st!=null)st.close();If(con!=null)con.close();
```

지금까지 JDBC사용방법을 설명하였다. 익숙해 지도록 내용을 몰라도 계속 반복해 보자.

좀더 이해를 높이기 위해서 각각의 경우를 핸드폰으로 전화거는 것과 비교해 볼 예정이다.

1. 사용 할 DB종류 선택은 다른 사람에게 전화를 하려고 하는데 핸드폰이 2개 일 경우 둘 중에 하나를 선택하여야 한다. 마찬가지로 어떤 데이터베이스를 사용할 것인지 결정하여 문자열로 기술하여야 한다. `Class.forName("oracle.jdbc.driver.OracleDriver");`
2. 원하는 db 연결 정보 객체 생성은 전화를 걸려면 전화할 곳의 정보를 주소록에서 찾아 세팅하여야 한다.

```
Connection con = DriverManager.getConnection(url,user,pw);
```

3. 연결 정보 객체로 부터 연결된 상태 객체 Statement st 를 얻어 오는 과정은 전화번호부를 선택한 다음 send 버튼을 눌러서 상대방이 전화를 받으면 서로 대화 가능한 상태가 되어 지속적으로 대화를 나눌수 있는 상태를 만든 것이다.

```
Statement st = con.createStatement();
```

4. 연결 객체를 통해서 원하는 데이터를 요청하고 결과 객체에 담는 경우는 send버튼을 눌러 통신상태가 되면 전화기로 연결된 상태를 이용해서 서로 번갈아 대화를 하여 원하는 정보를 주고 받는 것을 의미한다. 이때, 단순 대화같은 경우 executeUpdate메소드를 사용하고 택배와 같이 실질적으로 받을 데이터들이 발생하려면 executeQuery메소드를 사용한다.

5. 받은 결과를 출력하는 경우 본인이 받은 택배의 포장을 풀어 원하는 물건을 얻어 내는 것과 같은 작업을 ResultSet 인스턴스를 가지고 해주어야 한다.

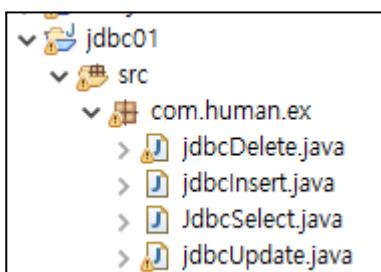
6. 사용한 자원을 모두 반납하는 경우는 전화 통화가 끝나면 종료 버튼을 눌러 통화가 끝났음을 알려주어야 비로소 통화가 종료되고 통화가 완전히 종료되면 핸드폰을 닫고 핸드폰을 원래 위치에 가져다 둔다.

간단히 테이블을 만들고 데이터를 넣고 빼는 프로그램을 만들어볼 예정이다. 다음 테이블을 만들고 데이터를 입력한 다음 코드를 작성해서 결과물을 확인해 보자.

3. 다음 CRUD 코드를 확인해 보자.

다음 코드는 human테이블에 insert, update, delete, select작업을 한 코드이다.

데이터 베이스에 연결해서 select를 통해 원하는 데이터 읽어오는 방법



다음 코드는 Java에서 JDBC(Java Database Connectivity)를 사용하여 Oracle 데이터베이스에 연결하고, SELECT 문을 실행하여 결과를 출력하는 jdbcSelect.java 파일 예제입니다.

드라이버 로딩

`Class.forName("oracle.jdbc.driver.OracleDriver");`: Oracle JDBC 드라이버를 로딩합니다. JDBC 드라이버를 로딩하지 않으면 데이터베이스에 접속할 수 없습니다.

데이터베이스 연결

`DriverManager.getConnection(url,user,pw);`: Oracle 데이터베이스에 연결합니다. url은 데이터베이스 서버의 주소와 포트, SID/Service Name을 포함한 URL입니다. user와 pw는 데이터베이스에 접속할 사용자 이름과 비밀번호입니다.

데이터베이스 SQL문 전송을 위한 쿼리 생성과 담당 객체 생성

`String sql="select * from human";`: human 테이블에서 모든 데이터를 조회하는 SELECT 쿼리를 작성합니다.

`conn.createStatement();`: SQL 문을 데이터베이스로 전송하는 Statement 객체를 생성합니다.

데이터베이스에 SQL문을 전송

`st.executeQuery(sql);`: SQL 문을 데이터베이스로 전송하고 결과를 ResultSet 객체로 받아 옵니다. sql실행 결과 생성되는 데이터가 있다면 executeQuery 없다면 executeUpdate를 생성 한다. executeQuery는 select에 사용, executeUpdate는 insert, update, delete에 사용한다.

ResultSet 출력하기

`while(rs.next()) { ... }`: ResultSet 객체에 조회 결과가 있는 동안 반복합니다.

ResultSet의 next() 메서드를 호출하면 조회 결과를 한 행씩 읽어옵니다.

`rs.getString(1);, rs.getInt(2);, rs.getDouble(3);, rs.getTimestamp(4);`:
ResultSet에서 테이블의 각 컬럼의 값을 가져오는 메서드입니다. 인덱스나 컬럼명을 이용해서 값을 가져올 수 있습니다. 인덱스는 sql에서 테이블 만들때 컬럼 기술한 순서이다.

`rs.getString("name");, rs.getInt("age");, rs.getDouble("height");,`
`rs.getTimestamp("birthday");`: ResultSet에서 각 컬럼의 값을 가져오는 메서드입니다.
컬럼명을 이용해서 값을 가져올 수 있습니다.

데이터베이스와 연결된 자원을 반납

`rs.close();, st.close();, conn.close();`: ResultSet, Statement, Connection 등 데이터베이스와 관련된 자원을 반납합니다. 이를 하지 않으면 데이터베이스에 대한 연결이 종료되지 않고 남아있을 수 있습니다.

import 할 패키지를 잘 확인해서 추가하자.

```
package com.human.ex;
```

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.LocalDateTime;
public class JdbcSelect {
    public static void main(String[] args) {
        try {
            //1. 드라이버 로딩
            //class.forName == new 클래스() 문자열로 클래스 생성하는 메소드
            Class.forName("oracle.jdbc.driver.OracleDriver");
            System.out.println("드라이버 연결");
            //2. 데이터베이스 연결 Connection클래스
//jdbc:oracle:thin:(제품명)@localhost(주소):1521(포트):xe(씨드아이디)
            String url="jdbc:oracle:thin:@localhost:1521:xe";
            String user="c##human";
            String pw="human";
            Connection conn=DriverManager.getConnection(url,user,pw);
            System.out.println("데이터베이스에 접속 성공");
            //3. 데이터베이스 sql문 전송을위한 쿼리 생성과 담당 객체 생성(statement)
            String sql="select * from human";
            System.out.println("sql 에러시 여기서 문자열을 확인:"+sql);
            Statement st=conn.createStatement();
            //4. 데이터베이스에 sql문을 전송
            ResultSet rs=st.executeQuery(sql);
            //5. resultSet 출력하기
            //ResultSet rs에 sql문 실행 결과가 테이블 형태로 담겨져 있다.
            //읽어올때 사용하는 메소드
            //getInt() .getDouble() .getString() .getTimestamp()
            //읽어 올때 여러 컬럼중 어떤 컬럼의 값을 가져올 것인가?
            //2가지 방법이 있는데
            //1. 테이블 만들때 기술한 순서의 숫자
            //2. 테이블 만들때 사용한 컬럼명으로 읽어오는 방법
            while(rs.next()) {
                //1. 테이블 만들때 기술한 순서의 숫자
                String name =rs.getString(1);
                int age=rs.getInt(2);
                double height=rs.getDouble(3);
                LocalDateTime birthday=rs.getTimestamp(4).toLocalDateTime();
                System.out.println("name:"+name);
                System.out.println("age:"+age);
                System.out.println("height:"+height);
                System.out.println("birth:"+birthday);
                System.out.println("-----");
                //2. 테이블 만들때 사용한 컬럼명으로 읽어오는 방법
                String name =rs.getString("name");

```

```

        int age=rs.getInt("age");
        double height=rs.getDouble("height");
        LocalDateTime birthday=rs.getTimestamp("birthday").toLocalDateTime();
        System.out.println("name:"+name);
        System.out.println("age:"+age);
        System.out.println("height:"+height);
        System.out.println("birth:"+birthday);
        System.out.println("-----");
    }
    //6. 데이터베이스와 연결된 자원을 반납
    if(rs!=null) rs.close();
    if(st!=null) st.close();
    if(conn!=null) conn.close();
} catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (SQLException e) {
    e.printStackTrace();
}
}
}
}

데이터 베이스에 연결해서 insert를 통해 원하는 데이터 입력하는 방법
```

select의 경우는 executeQuery를 사용하지만 리턴할 데이터가 없는 insert, delete, select의 경우에는 executeUpdate를 사용한다.

executeQuery의 경우 검색 결과가 ResultSet형태로 리턴되지만 executeUpdate의 경우 실행결과 변경된 데이터의 개수를 리턴하고 이를 이용해서 에러 체크 코드로 사용한다.

```

package com.human.ex;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
public class jdbcInsert {
    public static void main(String[] args) {
        try {
            // JDBC 드라이버 로드
            Class.forName("oracle.jdbc.driver.OracleDriver");
            // 데이터베이스 접속 정보 설정
            String url="jdbc:oracle:thin:@localhost:1521:xe";
            String user="c##human";
            String pw="human";
            Connection conn=DriverManager.getConnection(url,user,pw);
            // SQL문 실행을 위한 Statement 객체 생성
            Statement st=conn.createStatement();
        }
    }
}
```

```

        // INSERT할 데이터 설정
        String name ="홍길도";
        int age=15;
        double height=166;
        LocalDateTime birthday=LocalDateTime.now();
        // SQL문 작성 - 값 대신 변수를 사용하여 SQL문을 더욱 유연하게 작성
        String sql=String.format("insert into human values('"+
        "%s',%d,%f,to_date('%s', 'YYYY:MM:DD HH24:MI:SS'))"
        , name,age,height,birthday.format(DateTimeFormatter.ofPattern(
                    "yyyy-MM-dd HH:mm:ss"))));
        System.out.println("sql 에러시 여기서 문자열을 확인:"+sql);
        // SQL문 실행 및 결과 처리
        int cnt=st.executeUpdate(sql); // executeUpdate 메서드는
변경된 데이터 개수가 리턴된다.
        System.out.println(cnt+"개 데이터가 입력되었습니다.");
        // 자원 해제
        if(st !=null) st.close();
        if(conn !=null) conn.close();
    } catch (ClassNotFoundException e) {
        // JDBC 드라이버 로드 실패시 예외처리
        e.printStackTrace();
    } catch (SQLException e) {
        // 데이터베이스 접속, SQL 실행 실패시 예외처리
        e.printStackTrace();
    }
}
}
}

```

어렵더라도 코드를 쳐보고 돌려본 다음 데이터베이스와 실행 결과물을 확인하여 이해하여 보자. 다음은 `jdbcDelete.java` 코드이다.

```

package com.human.ex;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
public class jdbcDelete {
    public static void main(String[] args) {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            String url="jdbc:oracle:thin:@localhost:1521:xe";
            String user="c##human";
            String pw="human";
            Connection conn=DriverManager.getConnection(url,user,pw);
            //java.sql로 import해야 한다.
            Statement st=conn.createStatement();

```

```

        String name ="홍길도";
        String sql=String.format("delete human where name='%s'", name);
        //executeUpdate는 변경된 데이터 개수가 리턴된다.
        System.out.println("sql 에러시 여기서 문자열을 확인:"+sql);
        int cnt=st.executeUpdate(sql);
        System.out.println(cnt+"개 데이터가 삭제되었습니다.");
        if(st !=null) st.close();
        if(conn !=null) conn.close();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

다음 코드는 jdbcUpdate.java 방법이다.

```

package com.human.ex;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
public class jdbcUpdate {
    public static void main(String[] args) {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            String url="jdbc:oracle:thin:@localhost:1521:xe";
            String user="c##human";
            String pw="human";
            Connection conn=DriverManager.getConnection(url,user,pw);
            //java.sql로 import해야 한다.
            Statement st=conn.createStatement();
            String name ="홍길도";
            int age=39;
            String sql=String.format("update human set age=%d where
name='%s'", age,name);
            System.out.println("sql 에러시 여기서 문자열을 확인:"+sql);
            //executeUpdate는 변경된 데이터 개수가 리턴된다.
            int cnt=st.executeUpdate(sql);
            System.out.println(cnt+"개 데이터가 변경되었습니다.");
            if(st !=null) st.close();
            if(conn !=null) conn.close();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {

```

```

        e.printStackTrace();
    }
}
}

```

다음 테이블 CRUD작업이 가능한 JDBC프로그램을 만들어 보자.

```

CREATE TABLE employees (
    employee_id NUMBER(5) PRIMARY KEY,
    first_name VARCHAR2(50),
    last_name VARCHAR2(50),
    hire_date DATE,
    salary NUMBER(10, 2)
);

```

-- 다음 시퀀스를 만들고 입력시 pk값으로 넣어 보자.

```
CREATE SEQUENCE employee_seq;
```

ex)

```
insert into employees values(employee_seq.nextval, 'park', 'su', sysdate, 12.3);
```

commit을 잊지말자.

> 04. prepareStatement



statement를 업그레이드해서 preparedStatement로 만들 수 있다. preparedStatement는 전송할 sql 문자열에서 데이터 부분을 ?표를 사용하여 값을 넣는 방법이다. statement를 사용하는 preparedStatement를 사용하는 실행결과의 차이는 없다. 두개의 코드를 각각 확인해서 두 코드의 차이를 꼭 확인해 보자.

다음은 select문을 사용한 코드이다.

```
package com.human.prepared;
```

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Timestamp;
import java.time.LocalDateTime;
public class jdbcselect {
    //throws 예외처리를 해당 메소드를 호출한 사람에게 위임한다.
    public static void main(String[] args) throws Exception{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url="jdbc:oracle:thin:@localhost:1521:xe";
        String user="c##human";
        String pw="human";
        Connection conn=DriverManager.getConnection(url,user,pw);

        String sql="select * from human";
        System.out.println("sql 에러시 여기서 문자열을 확인:"+sql);
        PreparedStatement ps=conn.prepareStatement(sql);

        ResultSet rs=ps.executeQuery();
        while(rs.next()) {
            String name =rs.getString("name");
            int age=rs.getInt("age");
            double height=rs.getDouble("height");
            LocalDateTime birthday=rs.getTimestamp("birthday").toLocalDateTime();

            System.out.println("name:"+name);
            System.out.println("age:"+age);
            System.out.println("height:"+height);
            System.out.println("birth:"+birthday);
            System.out.println("-----");
        }
        if(rs!=null)rs.close();
        if(ps !=null) ps.close();
        if(conn !=null) conn.close();
    }
}

```

다음은 insert문을 사용한 코드이다.

```

package com.human.prepared;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.Timestamp;
import java.time.LocalDateTime;
public class jdbcinsert {
    //throws 예외처리를 해당 메소드를 호출한 사람에게 위임한다.
    public static void main(String[] args) throws Exception{

```

```

        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url="jdbc:oracle:thin:@localhost:1521:xe";
        String user="c##human";
        String pw="human";
        Connection conn=DriverManager.getConnection(url,user,pw);
        String sql="insert into human values(?, ?, ?, ?)";
        System.out.println("sql 에러시 여기서 문자열을 확인:"+sql);
        PreparedStatement ps=conn.prepareStatement(sql);
        ps.setString(1, "홍길동");
        ps.setInt(2, 10);
        ps.setDouble(3, 42.2);
        //LocalDateTime >> Timestamp
        //java.time.LocalDateTime 자바에서 시간을 다루는 클래스
        //java.sql.Timestamp 데이터베이스에 넣을때 사용되는 시간 클래스
        ps.setTimestamp(4, Timestamp.valueOf(LocalDateTime.now()));
        int n = ps.executeUpdate();
        System.out.println(n+"개의 로우 변경");
        if(ps !=null) ps.close();
        if(conn !=null) conn.close();
    }
}

```

다음은 delete문을 사용한 코드이다.

```

package com.human.prepared;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.Timestamp;
import java.time.LocalDateTime;
public class jdbcdelete {
    public static void main(String[] args) throws Exception{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url="jdbc:oracle:thin:@localhost:1521:xe";
        String user="c##human";
        String pw="human";
        Connection conn=DriverManager.getConnection(url,user,pw);
        String sql="delete from human where name=?";
        System.out.println("sql 에러시 여기서 문자열을 확인:"+sql);
        PreparedStatement ps=conn.prepareStatement(sql);
        ps.setString(1, "홍길동");
        int n = ps.executeUpdate();
        System.out.println(n+"개의 로우 변경");
        if(ps !=null) ps.close();
        if(conn !=null) conn.close();
    }
}

```

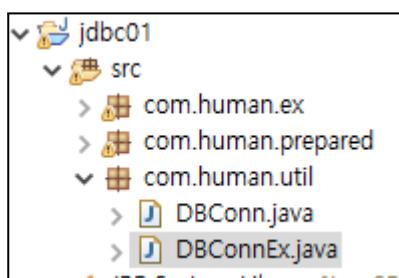
다음은 update문을 사용한 코드이다.

```

package com.human.prepared;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.Timestamp;
import java.time.LocalDateTime;
public class jdbcupdate {
    //throws 예외처리를 해당 메소드를 호출한 사람에게 위임한다.
    //throws 예외처리를 해당 메소드를 호출한 사람에게 위임한다.
    public static void main(String[] args) throws Exception{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url="jdbc:oracle:thin:@localhost:1521:xe";
        String user="c##human";
        String pw="human";
        Connection conn=DriverManager.getConnection(url,user,pw);
        String sql="update human set age=? where name=?";
        System.out.println("sql 에러시 여기서 문자열을 확인:"+sql);
        PreparedStatement ps=conn.prepareStatement(sql);
        ps.setInt(1, 13);
        ps.setString(2, "홍길동");
        int n = ps.executeUpdate();
        System.out.println(n+"개의 로우 변경");
        if(ps !=null) ps.close();
        if(conn !=null) conn.close();
    }
}

```

> 05. DBConn과 userInput



DBConn 클래스는 복잡한 JDBC 사용 방법을 쉽게 구현하기 위해서 만든 클래스이고 DBConnEx는 DBConn를 사용하는 클래스이다. DBConn를 사용하면 기존에 복잡하게 사용하던 JDBC 프로그램을 다음과 같이 간단하게 사용할 수 있다. `ResultSet rs=DBConn.createStatement("select * from human");`를 이용해서 간단하게 사용할 수 있다.

DBConn 객체는 statementQuery나 statementUpdate 안에서 sql 문자열로 데이터베이스를 조작하고 작업이 끝나면 생성된 DB 조작 관련 데이터를 반납한다.

DBConn.java 파일

코드는 DB 연결과 쿼리 실행에 대한 클래스(DBConn)를 정의한 것입니다. 기존 작성된 CRUD작업 코드를 확인해 보면 sql문자열만 변경되고 나머지 코드는 비슷하다는 것을 확인할 수 있다. DBConn은 간단한 방법으로 CRUD작업을 할 수 있도록 만든 클래스이다.

해당 코드를 이해하는 데는 지금까지 공부한 내용으로 불가능 하니 완성된 코드를 그냥 가져다 사용하자. 내용을 몰라도 DBConnEx.java에서 사용 하는데는 문제가 없다.

코드에서 중요한 부분은 다음과 같은 DB연결관련 코드이다. 본인이 사용하는 데이터 베이스 연결정보와 사용자 id,pw가 다르다면 반드시 이부분을 변경해야 한다.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
String url = "jdbc:oracle:thin:@localhost:1521:xe";
String id = "c##human";
String pw = "human";
```

```
//ctl+a 전체 선택
//ctl+shift +f 코드 정리

package com.human.util;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class DBConn {
    private DBConn() {
    }
    private static Connection dbConn = null;
    private static Statement st = null;
    private static ResultSet rs = null;
    public static Connection getInstance() {
        if (dbConn == null) {
            try {
                Class.forName("oracle.jdbc.driver.OracleDriver");
                String url = "jdbc:oracle:thin:@localhost:1521:xe";
                String id = "c##human";
                String pw = "human";
                dbConn = DriverManager.getConnection(url, id, pw);
                System.out.println("DBConnection....");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return dbConn;
    }
    public static void dbClose() {
        try {
```

```

        if (rs != null) rs.close();
        if (st != null) st.close();
        if (dbConn != null) dbConn.close();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        rs = null; st = null; dbConn = null;
    }
}

public static int statementUpdate(String sql) {
    DBConn.getInstance();
    int rValue = -1;
    if (dbConn != null) {
        try {
            if (st == null) st = dbConn.createStatement();
            rValue = st.executeUpdate(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    } else {
        System.out.println("not connected...");
    }
    return rValue;
}

public static ResultSet statementQuery(String sql) {
    DBConn.getInstance();
    if (DBConn.dbConn != null) {
        try {
            if (st == null) st = dbConn.createStatement();
            //insert, delete, update
            rs = st.executeQuery(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    } else {
        System.out.println("not connected...");
    }
    return rs;
}
}

```

DBConnEx.java

이 코드는 DBConn 클래스를 이용하여 데이터베이스에 접근하고, 데이터를 추가, 수정, 삭제, 조회하는 예시 코드입니다.

먼저, DBConn 클래스에서는 데이터베이스 작업을 수행하는 `statementUpdate()`와 `statementQuery()` 메서드가 있다.

이 예시 코드에서는 DBConn 클래스를 이용하여 다음과 같은 작업을 수행합니다.

insert: "human" 테이블에 새로운 데이터를 추가합니다.

update: "human" 테이블에서 "홍길나"의 나이를 50으로 수정합니다.

delete: "human" 테이블에서 이름이 "홍길나"인 데이터를 삭제합니다.

select: "human" 테이블의 모든 데이터를 조회하고, 결과를 출력합니다.

위 코드에서는 ResultSet 객체를 사용하여 조회한 데이터를 처리하고 있습니다. ResultSet 객체는 데이터베이스에서 쿼리 실행 결과를 담은 객체이며, next() 메서드를 호출하여 다음 행을 가져올 수 있습니다. 이 예시 코드에서는 while 문을 사용하여 모든 행을 순회하고, getString(), getInt(), getDouble(), getTimestamp() 등의 메서드를 사용하여 해당 컬럼의 데이터를 가져와 출력합니다.

마지막으로, dbClose() 메서드를 호출하여 데이터베이스 연결을 종료합니다. 이 때, ResultSet, Statement, Connection 객체를 차례대로 닫아주어야 합니다.

```
package com.human.util;
import java.sql.ResultSet;
import java.time.LocalDateTime;
public class DBConnEx {
    public static void main(String[] args) throws Exception{
//DBConn사용방법
//        DBConn.createStatement//DB사용 insert,delete,update
//        DBConn.createStatement //DB사용 select
//        DBConn.dbClose();      //DB종료
//resultSet사용을 완료후 DB를 닫아야 resultSet사용에 문제가 없다.

//insert
DBConn.createStatement("insert into human values('홍길나',52,151,sysdate)");
//update
DBConn.createStatement("update human set age= 50 where name='홍길나'");
//delete
DBConn.createStatement("delete human where name='홍길나'");
//select
ResultSet rs=DBConn.createStatement("select * from human");
while(rs.next()) {
    String name =rs.getString("name");
    int age=rs.getInt("age");
    double height=rs.getDouble("height");
    LocalDateTime birthday=rs.getTimestamp("birthday").toLocalDateTime();

    System.out.println("name:"+name);
    System.out.println("age:"+age);
    System.out.println("height:"+height);
    System.out.println("birth:"+birthday);
}
```

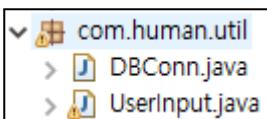
```

        System.out.println("-----");
    }
    //db닫는 작업
    DBConn.dbClose();
}
}

```

DBConn클래스를 만들어서 사용하면 이전 코드 보다 더욱 간단한 형태로 CRUD작업을 할 수 있다.

사용자입력 클래스 UserInput.java



이 코드는 UserInput 클래스를 정의하는 코드입니다. 이 클래스는 다양한 유형의 사용자 입력을 처리하는 메서드를 제공합니다.

`inputInt()`: 인자로 전달된 문자열을 출력한 뒤, 정수를 입력받아 반환합니다.

`inputDouble()`: 인자로 전달된 문자열을 출력한 뒤, 실수를 입력받아 반환합니다.

`inputString()`: 인자로 전달된 문자열을 출력한 뒤, 문자열을 입력받아 반환합니다.

`inputLocalDateTime()`: 인자로 전달된 문자열을 출력한 뒤, yyyy-MM-dd HH:mm:ss 형식으로 입력된 문자열을 파싱하여 LocalDateTime 객체로 반환합니다.

`dateToString()`: LocalDateTime 객체를 인자로 받아, yyyy-MM-dd HH:mm:ss 형식의 문자열로 변환하여 반환합니다.

이 클래스는 사용자 입력 처리를 캡슐화하여 코드의 재사용성을 높이고, 입력값의 유효성 검사와 같은 추가적인 기능을 제공할 수 있습니다.

다음 코드도 메소드를 배우지 않아서 이해하기 어려울 수 있으니 코드를 이해하기 보다 그냥 가져다가 사용방법만 익혀서 사용하자.

```

package com.human.util;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;
public class UserInput {
    private static Scanner sc=new Scanner(System.in);
    public static int inputInt(String st) {
        System.out.println(st+" 정수입력>>");
        return Integer.parseInt(sc.nextLine());
    }
    public static double inputDouble(String st) {

```

```

        System.out.println(st+" 실수입력>>");
        return Double.parseDouble(sc.nextLine());
    }
    public static String inputString(String st) {
        System.out.println(st+" 문자입력>>");
        return sc.nextLine();
    }
    public static LocalDateTime inputLocalDateTime(String str) {
        System.out.println(str+" 시간입력(yyyy-MM-dd HH:mm:ss)>>");
        return LocalDateTime.parse(sc.nextLine(),
            DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
    }
    public static String dateToString(LocalDateTime date) {
        return date.format(DateTimeFormatter.ofPattern(
            "yyyy-MM-dd HH:mm:ss"));
    }
}

```

다음은 com.human.dto.HumanDto.java 클래스이다. Dto 클래스는 데이터베이스 테이블에서 읽어온 하나의 데이터를 담는 클래스이다. 클래스 이름에 Dto를 넣어 다음과 같이 만들면된다.

```

package com.human.dto;
import java.time.LocalDateTime;
import java.util.Objects;
//dto에 추가해야 할것들
//1. getter,setter//2. 생성자//3. equals hashCode//4. toString
public class HumanDto {
    private String name;
    private Integer age;
    private Double height;
    private LocalDateTime birthday;
    @Override
    public String toString() {
        return "HumanDto [name=" + name + ", age=" + age + ", height=" +
height + ", birthday=" + birthday + "]";
    }
    @Override
    public int hashCode() {
        return Objects.hash(name);
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;
        HumanDto other = (HumanDto) obj;

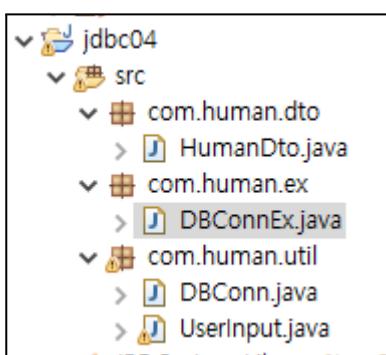
```

```

        return Objects.equals(name, other.name);
    }
    public HumanDto() {}
    public HumanDto(String name, Integer age, Double height, LocalDateTime
birthday) {
        super();
        this.name = name;          this.age = age;
        this.height = height;       this.birthday = birthday;
    }
    public HumanDto(HumanDto dto) {
        super();
        this.name = dto.name;      this.age = dto.age;
        this.height = dto.height;   this.birthday = dto.birthday;
    }

    public String getName() {    return name;    }
    public void setName(String name) { this.name = name; }
    public Integer getAge() {     return age;     }
    public void setAge(Integer age) { this.age = age; }
    public Double getHeight() {   return height;  }
    public void setHeight(Double height) { this.height = height; }
    public LocalDateTime getBirthday() {
        return birthday;
    }
    public void setBirthday(LocalDateTime birthday) {
        this.birthday = birthday;
    }
}

```



상위 코드는 UserInput클래스를 사용해서 사용자 입력을 받아 HumanDto클래스의 인스턴스에 값을 채우는 예제이다.

Dto, DBConn, UserInput 를 사용하여 사용자 입력을 넣어서 Human데이터 베이스를 조작하는 프로그램은 다음과 같다.

다음 코드를 확인한후 Human테이블 관리 프로그램을 구현해 보자.

```

package com.human.ex;
import java.sql.ResultSet;
import java.util.ArrayList;
import com.human.dto.HumanDto;
import com.human.util.DBConn;
import com.human.util.UserInput;

```

```

public class DBConnEx {
    public static void main(String[] args) {
        // 1. select
        ArrayList<HumanDto> resultDtos = new ArrayList<HumanDto>();
        ResultSet rs = DBConn.createStatementQuery(String.format("select * from human"));
        try {
            while (rs.next()) {
                resultDtos.add(new HumanDto(rs.getString("name"),
                    rs.getInt("age"),
                    rs.getDouble("height"),
                    rs.getTimestamp("birthday").toLocalDateTime()));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        for (HumanDto dto : resultDtos) {
            System.out.println(dto);
        }
        // 2. insert
        System.out.println("입력하고 자하는 Human데이터 입력");
        HumanDto dto=new HumanDto();
        dto.setName(UserInput.inputString("이름"));
        dto.setAge(UserInput.inputInt("나이"));
        dto.setHeight(UserInput.inputDouble("키"));
        dto.setBirthday(UserInput.inputLocalDateTime("생일"));

        String sql=String.format("insert into human values('"+
            "%s',%d,%f,to_date('%s', 'YYYY:MM:DD HH24:MI:SS'))"
            , dto.getName(),dto.getAge(),dto.getHeight()
            ,dto.getBirthday().format(DateTimeFormatter.ofPattern(
                "yyyy-MM-dd HH:mm:ss")));
        DBConn.createStatementUpdate(sql);
        // 3. delete
        System.out.println("삭제할 데이터 입력");
        String name =UserInput.inputString("이름 : ");
        sql=String.format("delete human where name='%s'", name);
        DBConn.createStatementUpdate(sql);
        // 4. update
        System.out.println("수정할 데이터 입력");
        name =UserInput.inputString("이름 : ");
        int age=UserInput.inputInt("나이 : ");
        sql=String.format("update human set age=%d where name='%s'", age,name);
        DBConn.createStatementUpdate(sql);
        System.out.println("프로그램 종료");
        //휴먼테이블 관리 프로그램을 완성해보자.
    }
}

```

상위 코드를 확인해서 Human 테이블의 관리형 프로그램을 구현 해보자.

```
package com.human.ex;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import com.human.dto.HumanDto;
import com.human.util.DBConn;
import com.human.util.UserInput;
public class DBConnEx {
    public static void main(String[] args) {
        int input = 0;
String strMainMenu = "-HumanTable 관리 프로그램-----\n";
        strMainMenu += "| 1.출력 | 2.입력 | 3.삭제 | 4.수정 | 5.종료 |\n";
        strMainMenu += "메뉴 번호 ";
        while (input != 5) {
            input = UserInput.inputInt(strMainMenu);
            switch (input) {
                case 1:
System.out.println("-----");
                    // 1.Select
                    ArrayList<HumanDto> resultDtos = new ArrayList<HumanDto>();
                    ResultSet rs = DBConn.createStatementQuery(String.format("select * from human"));
                    try {
                        while (rs.next()) {
                            resultDtos.add(new HumanDto(rs.getString("name")
                                , rs.getInt("age"), rs.getDouble("height"),
                                rs.getTimestamp("birthday").toLocalDateTime()));
                        }
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                    for (HumanDto dto : resultDtos) {
                        System.out.println(dto);
                    }
                    break;
                case 2:
                    // 2. insert
System.out.println("-----");
                    System.out.println("입력할 Human데이터 입력");
                    HumanDto dto = new HumanDto();
                    dto.setName(UserInput.inputString("이름 입력"));
                    dto.setAge(UserInput.inputInt("나이 입력"));
                    dto.setHeight(UserInput.inputDouble("키 입력"));
                    dto.setBirthday(UserInput.inputLocalDateTime("생일 입력"));
                    String sql = String.format("insert into human values('" +

```

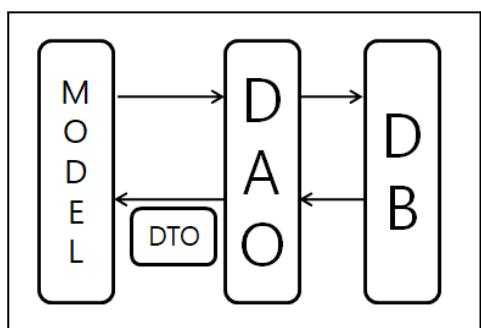
```
        "%s',%d,%f,to_date('%s', 'YYYY:MM:DD HH24:MI:SS'))",
        dto.getName(), dto.getAge(), dto.getHeight(),
        dto.getBirthday().format(DateTimeFormatter.ofPattern(
            "yyyy-MM-dd HH:mm:ss")));
    DBConn.createStatementUpdate(sql);
    System.out.println("[입력 완료]");
    break;
case 3:
    // 3.Delete
    System.out.println("-----");
    System.out.println("삭제할 데이터 입력");
    String name = UserInput.inputString("이름 입력");
    sql = String.format("delete human where name='%s'", name);
    DBConn.createStatementUpdate(sql);
    System.out.println("[삭제 완료]");
    break;
case 4:
    // 4.Update
    System.out.println("-----");
    String updateName = UserInput.inputString("수정할 이름 입력");
    int updateAge = UserInput.inputInt("수정할 나이 입력");
    sql = String.format("update human set age=%d where name='%s'",
        updateAge, updateName);
    DBConn.createStatementUpdate(sql);
    System.out.println("[수정 완료]");
    break;
case 5:
    System.out.println("-----");
    System.out.println("프로그램을 종료합니다.");
    input = 5;
    break;
default:
    System.out.println("보기에 있는 숫자를 입력하세요");
    break;
}
}
```

다음 문제를 풀어보자.

1. 데이터베이스를 이용한 은행 프로그램을 구현해 보자.

04. 데이터베이스 JDBC프로그래밍

> 6. model 1을 이용한 dto, dao



다음은 데이터베이스를 좀더 효율 적으로 접근 할 수 있는 구조화된 `model1`, `model2`를 만들 예정 이다. 용어들을 먼저 이해해 보자.

모델1은 이전 로직에서 데이터베이스 접근관련 코드 부분을 클래스로 분리한 형태라고 생각해도 된다.

`model1`이란? 사용자 입력을 처리하여 사용자가 원하는 비즈니스로직을 사용하여 원하는 데이터를 만들어주는

코드블럭을 의미한다. 비즈니스로직은 사용자가 프로그램에 요구하는 요구사항이라고 생각하면 된다.

`dto(database transfer object)`란? 데이터베이스의 테이블에 들어있는 하나의 row데이터를 java에서 사용할 수 있도록 만든 클래스 이다.

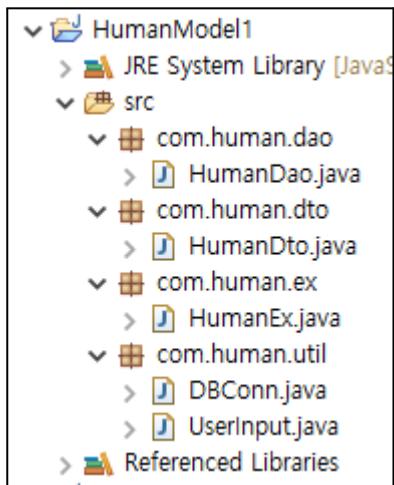
`dao(database access object)`란? 자바에서 데이터베이스를 조작하는 데 필요한 작업들을 모아놓은 클래스를 의미한다.

model1은 데이터를 처리할 때 DAO를 통해 DB의 데이터를 DTO에 담아서 바로 처리하는 방식을 의미한다. 장점은 코드를 이해하기 쉽게 관련된 코드를 끼리 분리할 수 있다. 단점은 사용자 입력이나 출력을 보여주는 view GUI화면과 model 비즈니스로직 코드가 섞여 있어 유지보수하기 불편하다. 이를 보완하고자 model과 view를 코드에서 분리하고 controller를 통해서 필요할 때 합치는 MVC모델 형태의 model2가 나왔다.

다음은 dto, dao를 이용한 model을 만드는 과정이다. 따라서 만들어보자.

처음 작업 하였던 human테이블을 가지고 dao, dto를 이용한 model1형태의 프로그램을 만들어 보자.

1. 처음에 만든 human테이블과 테이블안의 데이터가 c##human/human계정에 제대로 만들어져 있고 human테이블이 있는지 sqldeveloper로 확인하여 보자.



2. HumanModel1이라는 프로젝트명으로 프로젝트를 만든어 보자. 왼쪽은 완성되었을때의 프로젝트안의 패키지 모양이다. 새로 만들면 아무것도 없고 이후 하나씩 따라하면서 만들때 마다 추가될 것이다. ojdbc8_g.jar 파일도 새로 추가해야 한다.

3. human테이블의 데이터를 담을 수 있는 dto클래스를 만들어보자. 왼쪽 src폴더에서 마우스 오른쪽 클릭 new >> class 를 선택한 다음에 com.human.dao패키지에 HumanDto클래스를 만든 다음 아래와 같이 입력하여 HumanDto 클래스를 만들어 보자. 코드 중간에 name, age, height, birthday를 기술하고 예전에 배운 자동 완성을 이용하여 getter, setter, 생성자, toString를 추가해보자.

이전에 만든 HumanDto를 참조하자.

4. 다음에는 사용자 입력과 출력을 편하게 사용할 수 있는 입출력 클래스인 UserInput 클래스를 추가하자. 패키지는 com.human.util 클래스이름은 UserInput이다.

UserInput클래스는 모두 static 메소드로만 이루어진 클래스여서 따로 인스턴스를 생성할 필요없이 클래스이름으로 바로 접근해서 사용자로 부터 입력을 받는다.

5. 다음 만들 클래스는 반복해서 사용하는 JDBC프로그램에 특정 코드를 쉽게 사용할 수 있도록 클래스로 만든 것이다. 패키지는 com.human.util 클래스 이름은 DBConn로 만들어 추가하자.

DBConn클래스를 만들고 나면 이전 복잡하게 쓰던 코드를 간단하게 접근해서 사용할 수

있다.

6. 다음 HumanDao클래스는 com.human.dao 패키지에 추가하여 데이터 베이스의 human 테이블을 조작하는 클래스이다. 이전에는 데이터베이스 조작이 필요하면 여기 저기 선언해 데이터베이스에 접근 하였는데 HumanDao클래스를 만들어 human테이블과 관련있는 데이터베이스 작업을 모아 메소드로 구현해 놓고 필요할 때 생성해서 사용하는 클래스이다.

HumanDao 클래스는 HumanDto 객체를 데이터베이스와 연결하여 CRUD(Create, Read, Update, Delete) 작업을 수행하는 클래스입니다.

HumanDao 클래스에서는 다음과 같은 메서드들이 구현되어 있어 Human테이블관련 DB작업을 할 수 있다.

public ArrayList<HumanDto> select(): human 테이블의 모든 데이터를 조회하여 HumanDto 객체에 담아서 ArrayList에 추가하고, 그 ArrayList를 반환합니다.

public void insert(HumanDto dto): 입력받은 HumanDto 객체를 human 테이블에 추가합니다.

public void delete(String name): name 값을 기준으로 human 테이블에서 데이터를 삭제합니다.

public void update(int updateAge, String updateName): updateName 값을 기준으로 human 테이블에서 age 값을 updateAge 값으로 수정합니다.

com.human.dao.HumanDao.java클래스의 코드는 다음과 같다.

```
package com.human.dao;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import com.human.dto.HumanDto;
import com.human.util.DBConn;
public class HumanDao {
    public ArrayList<HumanDto> select() {
        ArrayList<HumanDto> resultDtos=new ArrayList<HumanDto>();
        ResultSet rs = DBConn.createStatementQuery(String.format("select * from human"));
        try {
            while (rs.next()) {
                resultDtos.add(new HumanDto(rs.getString("name")
                    , rs.getInt("age"), rs.getDouble("height"),
                    rs.getTimestamp("birthday").toLocalDateTime()));
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

```

        }
        return resultDtos;
    }
    public void insert(HumanDto dto) {
        String sql = String.format(
            "insert into human values('" + "%s',%d,%f,to_date('%s',
'YYYY:MM:DD HH24:MI:SS'))",
            dto.getName(), dto.getAge(), dto.getHeight(),
            dto.getBirthday().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss")));
        DBConn.createStatementUpdate(sql);
        System.out.println("[입력 완료]");
    }
    public void delete(String name) {
        String sql = String.format("delete human where name='%s'", name);
        DBConn.createStatementUpdate(sql);
        System.out.println("[삭제 완료]");
    }
    public void update(int updateAge, String updateName) {
        String sql = String.format("update human set age=%d where
name='%s'", updateAge, updateName);
        DBConn.createStatementUpdate(sql);
        System.out.println("[수정 완료]");
    }
}

```

상위 코드는 HumanDao 클래스에 insert, update, delete, select 관련 처리를 할 수 있는 메소드를 뮤어놓은 클래스이다. 이전에 만든 DBConn 객체를 사용하였다.

7. 다음 코드는 지금까지 만든 프로그램을 사용 운영하는 프로그램이다. com.human.ex 패키지에 DBConnEx 클래스에서 dao를 운영할 수 있도록 만들었다.

```

package com.human.ex;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import com.human.dao.HumanDao;
import com.human.dto.HumanDto;
import com.human.util.DBConn;
import com.human.util.UserInput;
public class DBConnEx {
    public static void main(String[] args) {
        HumanDao dao=new HumanDao();
        int input = 0;
String strMainMenu = "-HumanTable 관리 프로그램-----\n";
        strMainMenu += "| 1.출력 | 2.입력 | 3.삭제 | 4.수정 | 5.종료 |\n";
        strMainMenu += "메뉴 번호 ";

```

```

while (input != 5) {
    input = UserInput.inputInt(strMainMenu);
    switch (input) {
        case 1:
            System.out.println("-----");
            // 1.Select
            //ArrayList<HumanDto> resultDtos = new ArrayList<HumanDto>();
            ArrayList<HumanDto> resultDtos = dao.select();
            //ResultSet rs = DBConn.createStatementQuery(String.format("select * from human"));
            // try {
            //     while (rs.next()) {
            //         resultDtos.add(new HumanDto(rs.getString("name"),
            //             rs.getInt("age"), rs.getDouble("height"),
            //             rs.getTimestamp("birthday").toLocalDateTime()));
            //     }
            // } catch (SQLException e) {
            //     e.printStackTrace();
            // }
            for (HumanDto dto : resultDtos) {
                System.out.println(dto);
            }
            break;
        case 2:
            // 2. insert
            System.out.println("-----");
            System.out.println("입력할 Human데이터 입력");
            HumanDto dto = new HumanDto();
            dto.setName(UserInput.inputString("이름 입력"));
            dto.setAge(UserInput.inputInt("나이 입력"));
            dto.setHeight(UserInput.inputDouble("키 입력"));
            dto.setBirthday(UserInput.inputLocalDateTime("생일 입력"));
            dao.insert(dto);
            // String sql = String.format(
            //     "insert into human values('" + "%s',%d,%f,to_date('%s',
            'YYYY:MM:DD HH24:MI:SS'))",
            //     dto.getName(), dto.getAge(), dto.getHeight(),
            //     dto.getBirthday().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
            HH:mm:ss")));
            // DBConn.createStatementUpdate(sql);
            // System.out.println("[ 입력 완료 ]");
            break;
        case 3:
            // 3.Delete
            System.out.println("-----");
            System.out.println("삭제할 데이터 입력");
            String name = UserInput.inputString("이름 입력");
            dao.delete(name);
    }
}

```

```

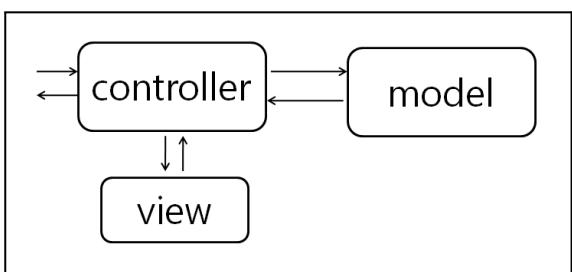
//           sql = String.format("delete human where name='%s'", name);
//           DBConn.statementUpdate(sql);
//           System.out.println("[삭제 완료]");
//           break;
case 4:
    // 4.Update
    System.out.println("-----");
    String updateName = UserInput.inputString("수정할 이름 입력");
    int updateAge = UserInput.inputInt("수정할 나이 입력");
    dao.update(updateAge, updateName);
//           sql = String.format("update human set age=%d where name='%s'",
updateAge, updateName);
//           DBConn.statementUpdate(sql);
//           System.out.println("[수정 완료]");
//           break;
case 5:
    System.out.println("-----");
    System.out.println("프로그램을 종료합니다.");
    input = 5;
    break;
default:
    System.out.println("보기에 있는 숫자를 입력하세요");
    break;
}}}

```

문제1) EMP테이블 관리프로그램을 구현해 보자.

문제2) 이전에 만든 은행 프로그램을 데이터베이스를 사용하여 데이터를 유지할 수 있는 형태로 업그레이드 해보자.

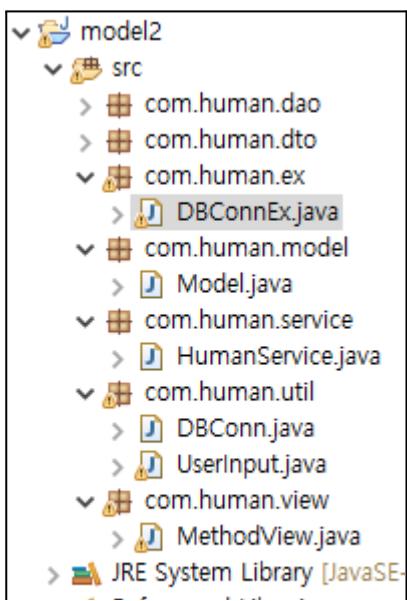
> 7. MVC model2 메소드이용



model1은 데이터베이스 접근시 dto와 dao를 사용하여 구조화하여 만든 프로그램을 의미하고, model2는 mvc패턴을 이용한 방법이다.

mvc패턴이란? 프로그램을 구현할때 하나로 만들지 않고 나누어 개발하는 방식중 하나로 MVC를 명확히 구분하여 프로그램 하는것을 의미한다.

m은 model로 사용자의 요청을 처리하는 비즈니스로직을 의미하고, v는 view로 사용자에게 데이터를 입력받거나 결과를 보여주는 화면을 의미하고 c는 컨트롤러로 사용자가 요청한 데이터를 판별하여 알맞은 데이터와 화면을 연결해 주는 작업을 한다. mvc를 명확하게 구분하여 프로그램을 하므로 코드 이해와 유지보수가 쉽게 된다. 전 페이지에서 작성한 model1을 업그레이드 하여 model2를 만들어 보자.



- 일단 model2라는 프로젝트를 만들고 이전 작업한 model1과 동일한 작업을 해보자. 시간이 부족 하다면 복사 붙여 넣기 해서 이후 과정을 진행해 보자. jdbc드라이버를 추가하는 것도 잊지 말자.
- 왼쪽과 비교해서 model1과 비교해서 추가된 부분은 com.human.service.HumanService.java와 com.human.model.Model.java와 com.human.view.MethodView.java이고 수정된 부분은

com.human.ex.DBConnEx.java 파일이다.

3. com.human.model.Model.java 클래스를 확인해보자.

다음 코드는 Java에서 MVC(Model-View-Controller) 패턴에서 Model 클래스를 구현한 예시입니다. Model 클래스는 Controller와 View 간에 데이터를 공유하는 용도로 사용된다.

Model 클래스는 HashMap을 이용하여 데이터를 저장하고, setAttribute() 메소드를 통해 key-value 쌍으로 데이터를 저장합니다. getAttribute() 메소드를 통해 저장된 데이터를 반환하며, removeAttribute() 메소드를 통해 저장된 데이터를 삭제합니다.

이렇게 Model 클래스를 이용하여 매개변수를 map을 이용한 형태로 주고 받는 것은, 메소드 호출의 매개변수 개수가 동적일 때 유용하며, 메소드의 유연성과 확장성을 높일 수 있습니다. 또한, Model 클래스를 통해 데이터를 저장하고 전달함으로써, 데이터의 일관성과 안전성을 보장할 수 있습니다.

```
package com.human.model;
import java.util.HashMap;
//1. MVC패턴을 만들때 많은 메소드들이 사용되는데 동일한 모양으로
// 메소드를 호출에서 데이터를 주고 받기 위해서
//매개변수를 map을 이용한 Model 클래스로 작성한다.
public class Model {
    private HashMap<String, Object> hashmap = new HashMap<String, Object>();
    public void setAttribute(String key, Object value) {
        hashmap.put(key, value);
    }
    public Object getAttribute(String key) {
        return hashmap.get(key);
    }
    public void removeAttribute(String key) {
        hashmap.remove(key);
    }
}
```

4. com.human.view.MethodView.java 를 만들어 보자.

다음 코드는 MVC(Model-View-Controller) 패턴에서 View 클래스를 구현한 예시입니다. View 클래스는 사용자가 입력한 데이터를 Controller에 전달하고, Controller가 처리한 결과를 사용자에게 보여주는 역할을 담당합니다.

MethodView 클래스에서는 mainMenuView(), selectResultView(), insertInputView(), deleteInputView(), updateInputView(), exitOutputView(), errorOutputView() 등의 메소드를 구현하고 있습니다. 각각의 메소드는 사용자에게 입력을 받거나, 처리 결과를 출력하는 역할을 수행합니다. 이때 각 View에서 발생한 데이터들을 호출한 곳에서 공유하기 위해서 매개변수로 Model 클래스가 온다.

Model 클래스의 객체를 매개변수로 받아서, setAttribute() 메소드를 이용하여 데이터를

저장하고, `getAttribute()` 메소드를 이용하여 저장된 데이터를 가져와서 사용합니다. 이렇게 함으로써, Controller와 View 간의 데이터 전달이 일관적이고 안전하게 이루어지며, 유연성과 확장성을 높일 수 있습니다.

View 클래스는 일반적으로 사용자 인터페이스를 구현하는 역할을 담당여 사용자의 입력을 받거나 사용자가 원하는 결과를 보여줄때 사용한다.

프로그램 실행시 사용자에게 보여줄 화면들을 각각 view메소드로 만들었다. 하나의 메소드는 하나의 프로그램 입출력 화면이 된다.

```
package com.human.view;
import java.util.ArrayList;
import com.human.dto.HumanDto;
import com.human.model.Model;
import com.human.util.UserInput;
public class MethodView {
    public static void mainMenuView(Model model) {
        String strMainMenu = "-HumanTable 관리 프로그램-----\n";
        strMainMenu += "| 1. 출력 | 2. 입력 | 3. 삭제 | 4. 수정 | 5. 종료 |\n";
        strMainMenu += "메뉴 번호 ";
        model.setAttribute("input", UserInput.inputInt(strMainMenu));
        //input = UserInput.inputInt(strMainMenu);
    }
    public static void selectResultView(Model model) {
        System.out.println("-----");
        for (HumanDto dto :
            (ArrayList<HumanDto>)model.getAttribute("resultDtos")) {
            System.out.println(dto);
        }
    }
    public static void insertInputView(Model model) {
        System.out.println("-----");
        System.out.println("입력할 Human데이터 입력");
        HumanDto dto = new HumanDto();
        dto.setName(UserInput.inputString("이름 입력"));
        dto.setAge(UserInput.inputInt("나이 입력"));
        dto.setHeight(UserInput.inputDouble("키 입력"));
        dto.setBirthday(UserInput.inputLocalDateTime("생일 입력"));
        model.setAttribute("dto", dto);
    }
    public static void deleteInputView(Model model) {
        System.out.println("-----");
        System.out.println("삭제할 데이터 입력");
        String name = UserInput.inputString("이름 입력");
        model.setAttribute("name", name);
    }
    public static void updateInputView(Model model) {
        System.out.println("-----");
```

```

        String updateName = UserInput.inputString("수정할 이름 입력");
        int updateAge = UserInput.inputInt("수정할 나이 입력");
        model.setAttribute("updateName", updateName);
        model.setAttribute("updateAge", updateAge);
    }
    public static void exitOutputView(Model model) {
        System.out.println("-----");
        System.out.println("프로그램을 종료합니다.");
        model.setAttribute("input", 5);
    }
    public static void errorOutputView(Model model) {
        System.out.println("보기에 있는 숫자를 입력하세요");
    }
}

```

5. com.human.service.HumanService.java 파일 만들기

다음 코드는 HumanDto 객체를 조작하는 CRUD (create, read, update, delete) 작업을 제공하는 HumanService의 자바 클래스 파일입니다. 보통 서비스에서는 dao작업과 비즈니스 로직을 담당하는데 이프로그램은 비즈니스로직을 가지고 있지 않아서 서비스안에 dao만 존재 한다. 이 클래스는 데이터베이스에서 HumanDto 객체를 조작하기 위한 데이터 액세스 객체인 HumanDao의 인스턴스를 가지고 있습니다.

비즈니스 로직은 다음과 같은 의미를 가진다. 비즈니스 로직은 어떤 비즈니스의 목적을 달성하기 위해 수행되는 컴퓨터화된 시스템의 동작 규칙을 의미합니다. 즉, 비즈니스 로직은 어떤 업무를 처리하기 위해 필요한 데이터 처리, 검증, 연산, 조건 분기 등과 같은 로직을 말하며, 이를 구현하는 것이 소프트웨어 개발에서 중요한 역할을 합니다.

이 클래스에는 다음과 같은 네 가지 공개 메서드가 있습니다.

`select()` : 이 메서드는 HumanDao의 `select()` 메서드에서 가져온 HumanDto 객체의 `ArrayList`를 반환합니다.

`insert(HumanDto dto)` : 이 메서드는 제공된 HumanDto 객체를 HumanDao의 `insert()` 메서드를 사용하여 데이터베이스에 삽입합니다.

`delete(String name)` : 이 메서드는 HumanDao의 `delete()` 메서드를 사용하여 제공된 이름에 해당하는 HumanDto 객체를 데이터베이스에서 삭제합니다.

`update(int updateAge, String updateName)` : 이 메서드는 HumanDao의 `update()` 메서드를 사용하여 제공된 이름에 해당하는 HumanDto 객체의 나이를 수정합니다.

```

package com.human.service;
import java.util.ArrayList;
import com.human.dao.HumanDao;

```

```

import com.human.dto.HumanDto;
public class HumanService {
    private HumanDao dao=new HumanDao();
    public ArrayList<HumanDto> select() {
        return dao.select();
    }
    public void insert(HumanDto dto) {
        // dto값을 변경해서 insert할 수 있다.
        dao.insert(dto);
    }
    public void delete(String name) {
        dao.delete(name);
    }
    public void update(int updateAge, String updateName) {
        dao.update(updateAge, updateName);
    }
}

```

6. com.human.ex.DBConnEx.java 파일의 기존 코드를 새로 추가한 클래스들을 이용해서 이전과 동일하게 동작하도록 구현하였다. 코드는 JDBC를 사용하여 데이터베이스에서 데이터를 가져오고, 수정하고, 삭제하는 간단한 CRUD (Create, Read, Update, Delete) 프로그램입니다. HumanService 클래스에는 select, insert, delete, update 메서드가 있습니다. DBConnEx 클래스는 이러한 서비스를 사용하여 데이터베이스와 상호 작용하고 사용자 입력을 처리합니다. 프로그램은 사용자가 5를 입력할 때까지 메인 메뉴를 계속해서 표시하며, 각 메뉴 옵션에 따라 적절한 작업을 수행합니다. 이를 통해 비즈니스 로직이 수행됩니다. 이 코드는 Model-View-Controller (MVC) 디자인 패턴을 사용하여 작성되었으며, 데이터베이스 연결을 처리하는 DBConn 클래스, DAO(Data Access Object)를 처리하는 HumanDao 클래스, DTO(Data Transfer Object)를 처리하는 HumanDto 클래스, 사용자 입력을 처리하는 UserInput 클래스, 출력을 처리하는 MethodView 클래스를 사용합니다.

```

package com.human.ex;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import com.human.dao.HumanDao;
import com.human.dto.HumanDto;
import com.human.model.Model;
import com.human.service.HumanService;
import com.human.util.DBConn;
import com.human.util.UserInput;
import com.human.view.MethodView;
public class DBConnEx {
    public static void main(String[] args) {
        HumanService humanService=new HumanService();

```

```

int input = 0;
while (input != 5) {
    Model model=new Model();
    MethodView.mainMenuView(model);
    input = (int)model.getAttribute("input");
    switch (input) {
        case 1:
            // 1.Select
            //ArrayList<HumanDto> resultDtos = new ArrayList<HumanDto>();
            ArrayList<HumanDto> resultDtos = humanService.select();
            model=new Model();
            model.setAttribute("resultDtos", resultDtos);
            MethodView.selectResultView(model);

//System.out.println("-----");
//        for (HumanDto dto : resultDtos) {
//            System.out.println(dto);
//        }
//            break;
        case 2:
            // 2. insert
            System.out.println("-----");
            System.out.println("입력할 Human데이터 입력");
            HumanDto dto = new HumanDto();
            dto.setName(UserInput.inputString("이름 입력"));
            dto.setAge(UserInput.inputInt("나이 입력"));
            dto.setHeight(UserInput.inputDouble("키 입력"));
            dto.setBirthday(UserInput.inputLocalDateTime("생일 입력"));
            model=new Model();
            MethodView.insertInputView(model);
            humanService.insert((HumanDto)model.getAttribute("dto"));
//
//                String sql = String.format(
//                    "insert into human values('"+ +
//"%s',%d,%f,to_date('%s', 'YYYY:MM:DD HH24:MI:SS'))",
//                    dto.getName(), dto.getAge(), dto.getHeight(),
//                    dto.getBirthday().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
//HH:mm:ss")));
//
//                DBConn.createStatementUpdate(sql);
//                System.out.println("[ 입력 완료 ]");
//                break;

        case 3:
            // 3.Delete
//        System.out.println("-----");
//        System.out.println("삭제할 데이터 입력");
//        String name = UserInput.inputString("이름 입력");
//        model=new Model();

```

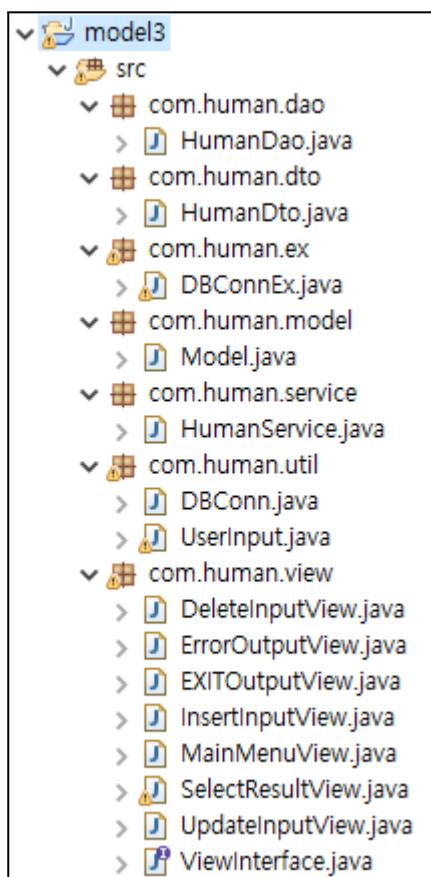
```

        MethodView.deleteInputView(model);
        humanService.delete((String)model.getAttribute("name"));
//      sql = String.format("delete human where name='%s'", name);
//      DBConn.statementUpdate(sql);
//      System.out.println("[삭제 완료]");
//      break;
    case 4:
        // 4.Update
//      System.out.println("-----");
//      String updateName = UserInput.inputString("수정할 이름 입력");
//      int updateAge = UserInput.inputInt("수정할 나이 입력");
        model=new Model();
        MethodView.updateInputView(model);
        humanService.update((int)model.getAttribute("updateAge"),
        (String)model.getAttribute("updateName"));
//      sql = String.format("update human set age=%d where
name='%s'", updateAge, updateName);
//      DBConn.statementUpdate(sql);
//      System.out.println("[수정 완료]");
//      break;
    case 5:
        model=new Model();
        MethodView.exitOutputView(model);
        input=(int)model.getAttribute("input");
        break;
    default:
        model=new Model();
        MethodView.errorOutputView(model);
        break;
    }
}
}
}

```

프로그램을 실행해보면 `model1`과 차이가 없이 `model2`가 동일하게 동작한다. 차이점은 `model2`가 더욱 관리하기 쉬운 코드가 된다. 실무에서 데이터 베이스 테이블이 300개 넘어가는 프로그램은 쉽게 볼 수 있다. MVC패턴을 이용해서 많은 파일로 프로젝트를 관리할 수 있게 된다.

> 8. MVC model2 인터페이스이용



view메소드들 인터페이스로 변경해서 MVC모델을 구현해보자.

MethodView.java클래스가 삭제되고 ViewInterface.java 인터페이스와 이를 상속 받은 많은 view클래스들이 com.human.view에 많이 만들어져 있고, DBConnEx.java파일 안에서 MethodView대신에 view클래스를 인터페이스에 상속해서 execute메소드를 실행하는 형태로 개선하였다.

```
ViewInterface view =null;  
view = new SelectResultView();  
view.execute(model);
```

1. 다음 코드는 인터페이스와 인터페이스를 상속받은 클래스들이다.

```
package com.human.view;  
  
import com.human.model.Model;  
public interface ViewInterface {
```

```

        public void execute(Model model);
    }

package com.human.view;
import com.human.model.Model;
import com.human.util.UserInput;
public class UpdateInputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        String updateName = UserInput.inputString("수정할 이름 입력");
        int updateAge = UserInput.inputInt("수정할 나이 입력");
        model.setAttribute("updateName", updateName);
        model.setAttribute("updateAge", updateAge);
    }
}

package com.human.view;
import java.util.ArrayList;
import com.human.dto.HumanDto;
import com.human.model.Model;
public class SelectResultView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        for (HumanDto dto :
            (ArrayList<HumanDto>)model.getAttribute("resultDtos")) {
            System.out.println(dto);
        }
    }
}

package com.human.view;
import com.human.model.Model;
import com.human.util.UserInput;
public class MainMenuView implements ViewInterface {
    @Override
    public void execute(Model model) {
        String strMainMenu = "-HumanTable 관리
프로그램-----\n";
        strMainMenu += "| 1.출력 | 2.입력 | 3.삭제 | 4.수정 | 5.종료 |\n";
        strMainMenu += "메뉴 번호 ";
        model.setAttribute("input", UserInput.inputInt(strMainMenu));
    }
}

package com.human.view;

```

```

import com.human.dto.HumanDto;
import com.human.model.Model;
import com.human.util.UserInput;
public class InsertInputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        System.out.println("입력할 Human데이터 입력");
        HumanDto dto = new HumanDto();
        dto.setName(UserInput.inputString("이름 입력"));
        dto.setAge(UserInput.inputInt("나이 입력"));
        dto.setHeight(UserInput.inputDouble("키 입력"));
        dto.setBirthday(UserInput.inputLocalDateTime("생일 입력"));
        model.setAttribute("dto", dto);
    }
}

package com.human.view;
import com.human.model.Model;
public class EXITOutputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        System.out.println("프로그램을 종료합니다.");
        model.setAttribute("input", 5);
    }
}

package com.human.view;
import com.human.model.Model;
public class ErrorOutputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("보기에 있는 숫자를 입력하세요");
    }
}

package com.human.view;
import com.human.model.Model;
import com.human.util.UserInput;
public class DeleteInputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        System.out.println("삭제할 데이터 입력");
        String name = UserInput.inputString("이름 입력");
        model.setAttribute("name", name);
    }
}

```

```
    }  
}
```

> 9. customer

13,14,15,16 장에서 customer,hobby 관련 조인테이블 처리방법에 관한 프로그램을 만들었다. 없는 코드 부분은 이전 코드에서 확인해서 만들어 보자.

사용할 테이블은 다음과 같다.

-- 취미 테이블 (HobbyDto)

```
drop table hobby;
```

```
CREATE TABLE Hobby (
```

```
    id number,
```

```
    hobby VARCHAR(255)
```

```
);
```

-- 고객 테이블 (CustomerDto)

```
drop table customer;
```

```
CREATE TABLE Customer (
```

```
    id number,
```

```
    name VARCHAR(255),
```

```
    height number,
```

```
birthday DATE  
);  
-- 시퀀스
```

```
create sequence id_counter;
```

두 개의 테이블인 Hobby와 Customer 간의 관계를 설명

일대다 (One-to-Many) 관계:

Hobby 테이블은 취미 정보를 저장하는 테이블입니다. 이 테이블은 id와 hobby 열을 가지고 있습니다.

Customer 테이블은 고객 정보를 저장하는 테이블입니다. 이 테이블은 id (고객 식별), name (고객 이름), height (고객 키), birthday (고객 생일) 열을 가지고 있습니다.

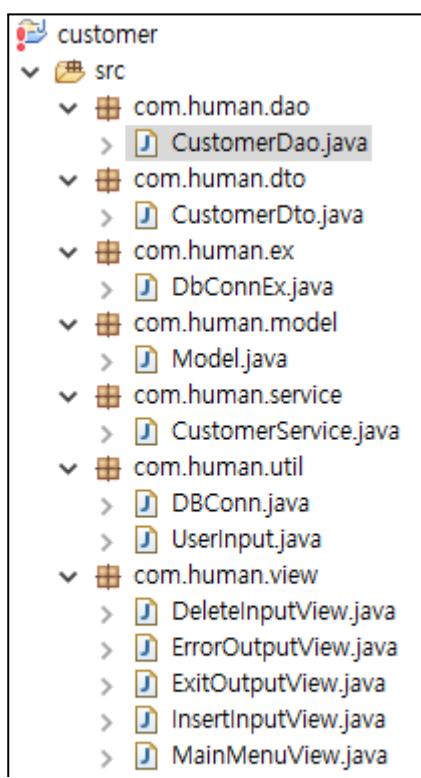
Hobby와 Customer 간의 관계는 일대다 관계입니다. 이것은 하나의 취미가 여러 명의 고객과 관련될 수 있다는 것을 의미합니다. 다시 말해, 하나의 취미는 여러 고객이 가질 수 있습니다.

외래 키 (Foreign Key):

이 관계를 데이터베이스에서 구현하기 위해 Customer 테이블에 hobby_id 열을 추가할 수 있습니다. 이 열은 Hobby 테이블의 id 열을 참조하며, 취미와 고객 사이의 관계를 설정합니다. 이러한 Hobby 테이블의 id 열을 외래 키(Foreign Key)라고 합니다.

외래 키를 사용하면 각 고객 레코드가 특정 취미 레코드를 참조할 수 있습니다. 따라서 어떤 고객이 어떤 취미를 가지고 있는지 추적할 수 있게 됩니다.

이것은 간단한 일대다 관계 예시입니다. 이 관계를 통해 데이터베이스에서 취미와 고객 간의 연관성을 유지하고 고객이 어떤 취미를 가지고 있는지 추적할 수 있습니다.



```
//customerDao.java  
package com.human.dao;  
  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.time.format.DateTimeFormatter;  
import java.util.ArrayList;  
  
import com.human.dto.CustomerDto;  
import com.human.util.DBConn;  
  
public class CustomerDao {
```

```

public ArrayList<CustomerDto> select() {
    ArrayList<CustomerDto> resultDtos=new ArrayList<CustomerDto>();
    ResultSet rs=DBConn.createStatementQuery(String.format("select * from
customer"));
    try {
        while(rs.next()) {
            resultDtos.add(new CustomerDto(rs.getInt("id"),
rs.getString("name"), rs.getDouble("height"),

rs.getTimestamp("birthday").toLocalDateTime()));
        }
    }catch (SQLException e) {
        e.printStackTrace();
    }
    return resultDtos;
}
public CustomerDto selectId(int id) {
    CustomerDto resultDtos=new CustomerDto();
    ResultSet rs=DBConn.createStatementQuery(String.format("select * from
customer where id=%d", id));
    if(rs!=null) {
        try {
            rs.next();
            resultDtos=new
CustomerDto(rs.getInt("id"),rs.getString("name"),rs.getDouble("height"),

rs.getTimestamp("birthday").toLocalDateTime());
        }catch (SQLException e) {
            e.printStackTrace();
        }catch(Exception e) {
            e.printStackTrace();
        }
    }
    return resultDtos;
}
public void insert(CustomerDto dto) {
    String sql=String.format("insert into customer
values("+ "%d, '%s', %f, to_date('%s', 'yyyy-mm-dd HH24:mi:ss'))",
    dto.getId(),dto.getName(),dto.getHeight(),dto.getBirthDay().format(DateTimeFo
rmatter.ofPattern("yyyy-MM-dd hh:mm:ss")));
    DBConn.createStatementUpdate(sql);
    System.out.println("[ 입력 완료 ]");
}
public void update(String updatename, int updateid) {
    String sql=String.format("update customer set id='%d' where

```

```

name='%s'", updateid, updatename);
        DBConn.statementUpdate(sql);
        System.out.println("[수정 완료]");
    }
    public void delete(int id) {
        String sql=String.format("delete customer where id=%d",id);
        DBConn.statementUpdate(sql);
        System.out.println("[삭제 완료]");
    }

    public ArrayList<Integer> getIds() {
        ArrayList<Integer> ids=new ArrayList<Integer>();
        ResultSet rs=DBConn.statementQuery(String.format("select * from
customer"));
        try {
            while(rs.next()) {
                ids.add(rs.getInt("id"));
            }
        }catch (SQLException e) {
            e.printStackTrace();
        }
        return ids;
    }

    public int getMaxId() {
        int maxIdValue=-1;
        ResultSet rs=DBConn.statementQuery(String.format("select max(id)
as maxId from customer"));
        if(rs!=null) {
            try {
                rs.next();
                maxIdValue=rs.getInt("maxId");
            }catch (SQLException e) {
                e.printStackTrace();
            }catch(Exception e) {
                e.printStackTrace();
            }
        }
        return maxIdValue;
    }
}

//CustomerDto.java

package com.human.dto;
import java.time.LocalDateTime;
import java.util.Objects;
public class CustomerDto {

```

```
private Integer id;
private String name;
private Double height;
private LocalDateTime birthday;
@Override
public int hashCode() {
    return Objects.hash(birthday, height, id, name);
}
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    CustomerDto other = (CustomerDto) obj;
    return Objects.equals(birthday, other.birthday) &&
Objects.equals(height, other.height)
        && Objects.equals(id, other.id) &&
Objects.equals(name, other.name);
}
@Override
public String toString() {
    return "CustomerDto [id=" + id + ", name=" + name + ", height=" +
height + ", birthday=" + birthday + "]";
}
public CustomerDto() {}
public CustomerDto(Integer id, String name, Double height,
LocalDateTime birthday) {
    super();
    this.id = id;
    this.name = name;
    this.height = height;
    this.birthday = birthday;
}
public Integer getId() {           return id;  }
public void setId(Integer id) {      this.id = id;   }
public String getName() {           return name;  }
public void setName(String name) {    this.name = name; }
public Double getHeight() {          return height; }
public void setHeight(Double height) { this.height = height; }
public LocalDateTime getBirthday() {     return birthday; }
public void setBirthday(LocalDateTime birthday) {
    this.birthday = birthday;
}
}
```

```

//DeleteInputView.java
public class DeleteInputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        System.out.println("삭제할 데이터 입력");
        int id = UserInput.inputInt("id 입력");
        model.setAttribute("id", id);
    }
}

//ErrorOutputView.java
public class ErrorOutputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("보기에 있는 숫자를 입력하세요");
    }
}

//ExitOutputView.java
package com.human.view;
import com.human.model.Model;
public class ExitOutputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        System.out.println("프로그램을 종료합니다.");
        model.setAttribute("input", 5);
    }
}

//InsertInputView.java
package com.human.view;
import com.human.dto.CustomerDto;
import com.human.model.Model;
import com.human.util.UserInput;
public class InsertInputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        System.out.println("생성할 Customer 데이터 입력");
        CustomerDto dto=new CustomerDto();
        dto.setId(UserInput.inputInt("id 입력"));
        dto.setName(UserInput.inputString("이름 입력"));
        dto.setHeight(UserInput.inputDouble("키 입력"));
        dto.setBirthday(UserInput.inputLocalDateTime("생일 입력"));
        model.setAttribute("dto", dto);
    }
}

```

```

//MainMenuView.java
package com.human.view;
import com.human.model.Model;
import com.human.util.UserInput;
public class MainMenuView implements ViewInterface {
    public void execute(Model model) {
        String strMainMenu = "~ Customer Service 프로그램 ~";
strMainMenu += "| 1.Select | 2.Insert | 3.Update | 4.Delete | 5.Exit |";
        model.setAttribute("input",UserInput.inputInt(strMainMenu));
    }
}

//SelectResultView.java
package com.human.view;
import java.util.ArrayList;
import com.human.dto.CustomerDto;
import com.human.model.Model;
public class SelectResultView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        for (CustomerDto dto :
(ArrayList<CustomerDto>)model.getAttribute("resultDtos")) {
            System.out.println(dto);
        }
    }
}

//UpdateInputView.java
package com.human.view;
import com.human.model.Model;
import com.human.util.UserInput;
public class UpdateInputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        String updatename = UserInput.inputString("수정할 이름 입력");
        int updateid = UserInput.inputInt("수정할 id 입력");
        model.setAttribute("updateid", updateid);
        model.setAttribute("updatename", updatename);
    }
}

//CustomerService.java
package com.human.service;
import java.util.ArrayList;
import com.human.dao.CustomerDao;
import com.human.dto.CustomerDto;
public class CustomerService {
    private CustomerDao dao=new CustomerDao();

```

```
public ArrayList<CustomerDto> select() {
    return dao.select();
}
public void insert(CustomerDto dto) {
    dao.insert(dto);
}
public void update(String updateName, int updateid) {
    dao.update(updateName, updateid);
}
public void delete(int id) {
    dao.delete(id);
}
}

//DBConnEx.java
package com.human.ex;
import java.util.ArrayList;
import com.human.dto.CustomerDto;
import com.human.model.Model;
import com.human.service.CustomerService;
import com.human.view.DeleteInputView;
import com.human.view.ErrorOutputView;
import com.human.view.ExitOutputView;
import com.human.view.InsertInputView;
import com.human.view.MainMenuView;
import com.human.view.SelectResultView;
import com.human.view.UpdateInputView;
import com.human.view.ViewInterface;
public class DbConnEx {
    public static void main(String[] args) throws Exception {
        CustomerService customerService=new CustomerService();
        ViewInterface view=null;
        int input=0;
        while(input!=5) {
            Model model=new Model();
            view=new MainMenuView();
            view.execute(model);
            input=(int)model.getAttribute("input");
            switch(input) {
                case 1:
                    ArrayList<CustomerDto> resultDtos= customerService.select();
                    model=new Model();
                    model.setAttribute("resultDtos", resultDtos);
                    view=new SelectResultView();
                    view.execute(model);
                    break;
                case 2:
                    model=new Model();

```

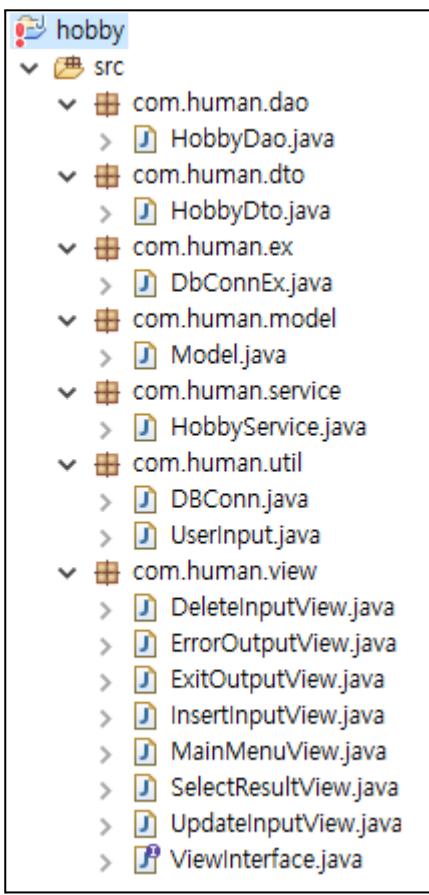
```

        view=new InsertInputView();
        view.execute(model);
customerService.insert((CustomerDto)model.getAttribute("dto"));
        break;
    case 3:
        model=new Model();
        view=new UpdateInputView();
        view.execute(model);
customerService.update((String)model.getAttribute("updatename"),
(int)model.getAttribute("updateid"));
        break;
    case 4:
        model=new Model();
        view=new DeleteInputView();
        view.execute(model);
customerService.delete((int)model.getAttribute("id"));
        break;
    case 5:
        model=new Model();
        view=new ExitOutputView();
        view.execute(model);
        input=(int)model.getAttribute("input");
        break;
    default:
        model=new Model();
        view=new ErrorOutputView();
        view.execute(model);
        break;
    }
}

}

```

> 10. hobby



```
//HobbyDao.java
package com.human.dao;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import com.human.dto.HobbyDto;

import com.human.util.DBConn;
public class HobbyDao {
    public ArrayList<HobbyDto> select() {
        ArrayList<HobbyDto> resultDtos=new ArrayList<HobbyDto>();
        ResultSet rs=DBConn.createStatementQuery(String.format("select * from hobby"));
    }
}
```

```

        try {
            while(rs.next()) {
                resultDtos.add(new HobbyDto(rs.getInt("id"), rs.getString("hobby")));
            }
        }catch (SQLException e) {
            e.printStackTrace();
        }
        return resultDtos;
    }
    public HobbyDto selectId(int id) {
        HobbyDto resultDtos=new HobbyDto();
        ResultSet rs=DBConn.createStatementQuery(String.format("select * from
hobby where id=%d", id));
        if(rs!=null) {
            try {
                rs.next();
            resultDtos=new HobbyDto(rs.getInt("id"),rs.getString("hobby"));
            }catch (SQLException e) {
                e.printStackTrace();
            }catch(Exception e) {
                e.printStackTrace();
            }
        }
        return resultDtos;
    }
    public void insert(HobbyDto dto) {
        String sql=String.format("insert into hobby values("+ "%d, '%s')",
                dto.getId(),dto.getHobby());
        DBConn.createStatementUpdate(sql);
        System.out.println("[입력 완료]");
    }
    public void update(String upHobby, int id) {
        String sql=String.format("update hobby set hobby='%s' where
id=%d",id,upHobby);
        DBConn.createStatementUpdate(sql);
        System.out.println("[수정 완료]");
    }
    public void delete(int id) {
        String sql=String.format("delete hobby where id=%d",id);
        DBConn.createStatementUpdate(sql);
        System.out.println("[삭제 완료]");
    }
}
//HobbyDto.java
package com.human.dto;
import java.util.Objects;
public class HobbyDto {

```

```

private Integer id;
private String hobby;
@Override
public String toString() {
    return "HobbyDto [id=" + id + ", hobby=" + hobby + "]";
}
@Override
public int hashCode() {
    return Objects.hash(hobby, id);
}
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null) return false;
    if (getClass() != obj.getClass())
        return false;
    HobbyDto other = (HobbyDto) obj;
    return Objects.equals(hobby, other.hobby) && Objects.equals(id,
other.id);
}
public HobbyDto() {}
public HobbyDto(Integer id, String hobby) {
    super();
    this.id = id;
    this.hobby = hobby;
}
public Integer getId() { return id; }
public void setId(Integer id) { this.id = id; }
public String getHobby() { return hobby; }
public void setHobby(String hobby) {
    this.hobby = hobby;
}
}

//HobbyService.java
package com.human.service;
import java.util.ArrayList;
import com.human.dao.HobbyDao;
import com.human.dto.HobbyDto;
public class HobbyService {
    private HobbyDao dao=new HobbyDao();
    public ArrayList<HobbyDto> select() {
        return dao.select();
    }
    public void insert(HobbyDto dto) {
        dao.insert(dto);
    }
    public void update(String upHobby, int id) {

```

```
        dao.update(upHobby, id);
    }
    public void delete(int id) {
        dao.delete(id);
    }
}

//DBConnEx.java
package com.human.util;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class DBConn {
    private DBConn() {
    }
    private static Connection dbConn=null;
    private static Statement st=null;
    private static ResultSet rs=null;
    public static Connection getInstance() {
        if (dbConn ==null) {
            try {
                Class.forName("oracle.jdbc.driver.OracleDriver");
                String url="jdbc:oracle:thin:@localhost:1521:xe";
                String id="c##human";
                String pw="human";
                dbConn=DriverManager.getConnection(url,id,pw);
                System.out.println("DBConnection....");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return dbConn;
    }
    public static void dbClose() {
        try {
            if(rs!=null)
                rs.close();
            if(st!=null)
                st.close();
            if(dbConn!=null)
                dbConn.close();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            rs=null;
            st=null;
        }
    }
}
```

```

        dbConn=null;
    }
}

public static int statementUpdate(String sql) {
    DBConn.getInstance();
    int rValue=-1;
    if(dbConn!=null) {
        try {
            if(st==null) {
                st=dbConn.createStatement();
            }
            rValue=st.executeUpdate(sql);
        } catch(SQLException e) {
            e.printStackTrace();
        }
    }else {
        System.out.println("not connected...");
    }
    return rValue;
}

public static ResultSet statementQuery(String sql) {
    DBConn.getInstance();
    if(DBConn.dbConn!=null) {
        try {
            if(st==null) {
                st=dbConn.createStatement();
            }
            rs=st.executeQuery(sql);
        }catch (SQLException e) {
            e.printStackTrace();
        }
    }else {
        System.out.println("not connected...");
    }
    return rs;
}
}

//DeleteInputView.java
package com.human.view;
import com.human.model.Model;
import com.human.util.UserInput;
public class DeleteInputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        System.out.println("삭제할 데이터 입력");
        int id = UserInput.inputInt("id 입력");
    }
}

```

```

        model.setAttribute("id", id);
    }
}

//ErrorOutputView.java
package com.human.view;
import com.human.model.Model;
public class ErrorOutputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("보기에 있는 숫자를 입력하세요");
    }
}

//ExitOutputView.java
package com.human.view;
import com.human.model.Model;
public class ExitOutputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        System.out.println("프로그램을 종료합니다.");
        model.setAttribute("input", 5);
    }
}

//InsertInputView.java
package com.human.view;
import com.human.dto.HobbyDto;
import com.human.model.Model;
import com.human.util.UserInput;
public class InsertInputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        System.out.println("생성할 Hobby 데이터 입력");
        HobbyDto dto=new HobbyDto();
        dto.setId(UserInput.inputInt("id 입력"));
        dto.setHobby(UserInput.inputString("hobby 입력"));
        model.setAttribute("dto", dto);
    }
}

//MainMenuView.java
package com.human.view;
import com.human.model.Model;
import com.human.util.UserInput;
public class MainMenuView implements ViewInterface {
    public void execute(Model model) {
        String strMainMenu = "~ Hobby 입력 프로그램 ~";
        strMainMenu += "| 1.Select | 2.Insert | 3.Update | 4.Delete | 5.Exit |";
    }
}

```

```

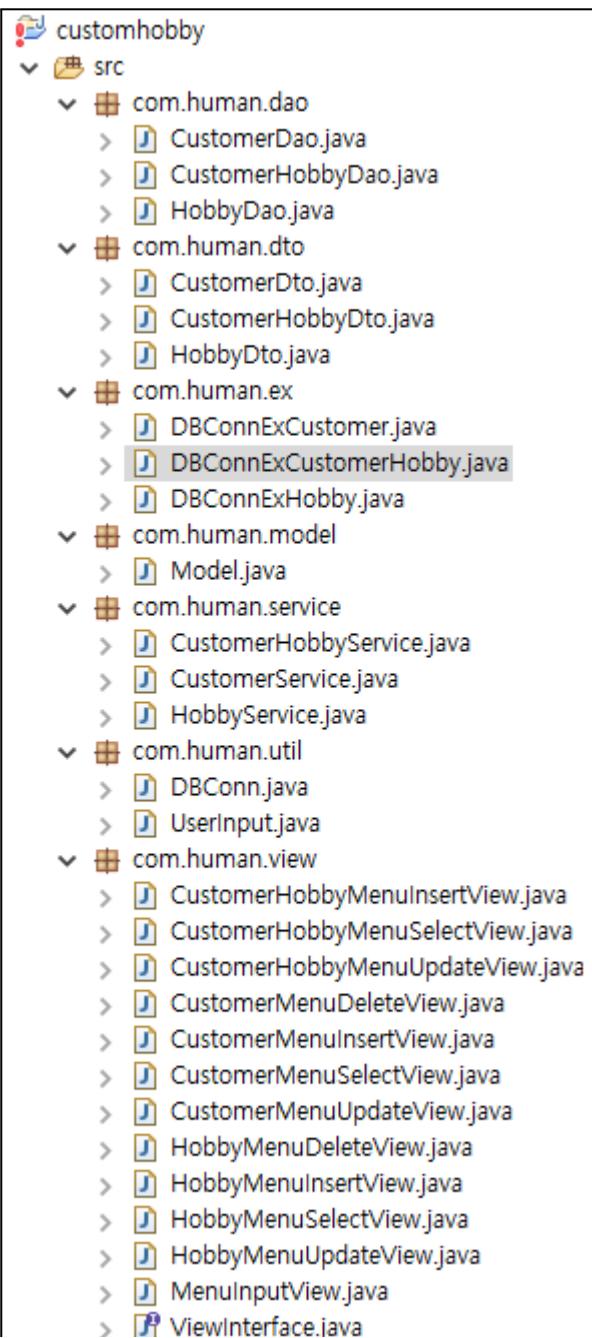
        //strMainMenu += "번호를 입력하세요. : ";
        model.setAttribute("input",UserInput.inputInt(strMainMenu));
    }
}

//SelectResultView.java
package com.human.view;
import java.util.ArrayList;
import com.human.dto.HobbyDto;
import com.human.model.Model;
public class SelectResultView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        for (HobbyDto dto :
            (ArrayList<HobbyDto>)model.getAttribute("resultDtos")) {
            System.out.println(dto);
        }
    }
}

//UpdateInputView.java
package com.human.view;
import com.human.model.Model;
import com.human.util.UserInput;
public class UpdateInputView implements ViewInterface {
    @Override
    public void execute(Model model) {
        System.out.println("-----");
        int id = UserInput.inputInt("수정할 id 입력");
        String upHobby = UserInput.inputString("수정할 Hobby 입력");
        model.setAttribute("id", id);
        model.setAttribute("upHobby", upHobby);
    }
}

```

> 11. customerhobby



```
public class CustomerHobbyDto {  
    private int id;  
    private String name;  
    private double height;  
    private LocalDateTime birthday;  
    private HobbyDto hobby;  
public class CustomerHobbiesDto {  
    private int id;  
    private String name;  
    private double height;  
    private LocalDateTime birthday;  
    private List<HobbyDto> hobbies;
```

두 클래스는 Customer테이블과 Hobby 테이블을 조인한 결과를 저장하는 방법이다.

CustomerHobbyDto 는 조인한 결과를 하나의 데이터로 담아서 처리하는 방법이고, **CustomerHobbiesDto** 는 Customer 데이터에 해당하는 여려개의 Hobby데이터를 검색한 후 hobbies List에 담아서 처리하는 방법이다.

CustomerHobbyDto 의 경우 로직은 간단하다는 장점과 반복된 데이터를 많이 가진다는 단점이 있고, **CustomerHobbiesDto** 의 경우 로직은 복잡하나 중복이 없어서 CRUD 작업이 쉬워진다는 장점이 있다.

```

//CustomerHobbyDao.java
package com.human.dao;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

import com.human.dto.CustomerHobbyDto;
import com.human.dto.HobbyDto;
import com.human.util.DBConn;
public class CustomerHobbyDao {
    public ArrayList<CustomerHobbyDto> selectAllCustomerAndHobby()
    {
        ArrayList<CustomerHobbyDto> resultDtos = new
ArrayList<CustomerHobbyDto>();
        ResultSet rs = DBConn.createStatementQuery(String.format
            ("select customer.* , hobby.hobby "
            + "from customer, hobby "
            + "where customer.id = hobby.id(+)"));
        try {
            while (rs.next()){
                resultDtos.add(new CustomerHobbyDto(
                    rs.getInt("id"),
                    rs.getString("name"),
                    rs.getDouble("height"),
                    rs.getTimestamp("birthday").toLocalDateTime(),
                    new HobbyDto(rs.getInt("id"),rs.getString("hobby"))
                ));
            }
        }catch(SQLException e){
            e.printStackTrace();
        }
        return resultDtos;
    }
    public CustomerHobbyDto selectOneCustomerAndHobby(int id, String hobby)
    {
        CustomerHobbyDto resultDto = null;
        String sql = "";
        if(hobby.equals("")){
            sql = "select customer.* , hobby.hobby from customer, hobby " +
                  "where customer.id = hobby.id(+) and customer.id = " +
            + id + " and hobby.hobby is null";
        }else {
            sql = "select customer.* , hobby.hobby from customer, hobby " +
                  "where customer.id = hobby.id(+) and customer.id = " + id + " and
hobby.hobby = '" + hobby + "'";
        }
        ResultSet rs = DBConn.createStatementQuery(sql);

```

```

        try {
            while(rs.next()) {
                resultDto = new CustomerHobbyDto(
                    rs.getInt("id"),
                    rs.getString("name"),
                    rs.getDouble("height"),
                    rs.getTimestamp("birthday").toLocalDateTime(),
                    new HobbyDto(rs.getInt("id"), rs.getString("hobby"))
                );
            }
        }catch(SQLException e){
            e.printStackTrace();
        }
        return resultDto;
    }
}

//CustomerHobbyDto.java
package com.human.dto;
import java.time.LocalDateTime;
public class CustomerHobbyDto {
    private int id;
    private String name;
    private double height;
    private LocalDateTime birthday;
    private HobbyDto hobby;
    public CustomerHobbyDto() {}
    public CustomerHobbyDto(int id, String name, double height,
    LocalDateTime birthday, HobbyDto hobby) {
        super();
        this.id = id;
        this.name = name;
        this.height = height;
        this.birthday = birthday;
        this.hobby = hobby;
    }
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((birthday == null) ? 0 :
        birthday.hashCode());
        long temp;
        temp = Double.doubleToLongBits(height);
        result = prime * result + (int) (temp ^ (temp >>> 32));
        result = prime * result + ((hobby == null) ? 0 :
        hobby.hashCode());
        result = prime * result + id;
    }
}

```

```
        result = prime * result + ((name == null) ? 0 : name.hashCode());
    return result;
}
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    CustomerHobbyDto other = (CustomerHobbyDto) obj;
    if (birthday == null) {
        if (other.birthday != null)
            return false;
    } else if (!birthday.equals(other.birthday))
        return false;
    if (Double.doubleToLongBits(height) !=
Double.doubleToLongBits(other.height))
        return false;
    if (hobby == null) {
        if (other.hobby != null)
            return false;
    } else if (!hobby.equals(other.hobby))
        return false;
    if (id != other.id)
        return false;
    if (name == null) {
        if (other.name != null)
            return false;
    } else if (!name.equals(other.name))
        return false;
    return true;
}
public int getId() {      return id;  }
public void setId(int id) {      this.id = id;  }
public String getName() {      return name;  }
public void setName(String name) {      this.name = name; }
public double getHeight() {      return height;  }
public void setHeight(double height) {
    this.height = height;
}
public LocalDateTime getBirthday() {
    return birthday;
}
public void setBirthday(LocalDateTime birthday) {
    this.birthday = birthday;
}
```

```

    }
    public HobbyDto getHobby() { return hobby; }
    public void setHobby(HobbyDto hobby) { this.hobby = hobby;}
    @Override
    public String toString() {
        return "CustomerHobbyDto [id=" + id + ", name=" + name + ", height=" + height + ", birthday=" + birthday + ", hobby=" + hobby + "]";
    }
}
//CustomerHobbyService.java
package com.human.service;
import java.util.ArrayList;
import com.human.dao.CustomerDao;
import com.human.dao.CustomerHobbyDao;
import com.human.dao.HobbyDao;
import com.human.dto.CustomerDto;
import com.human.dto.CustomerHobbyDto;
import com.human.dto.HobbyDto;
public class CustomerHobbyService {
    public CustomerDao cutomerDao = new CustomerDao();
    public HobbyDao hobbyDao = new HobbyDao();
    public CustomerHobbyDao customerHobbyDao = new CustomerHobbyDao();
    public ArrayList<CustomerHobbyDto> selectAll(){
        ArrayList<CustomerHobbyDto> dtos = new
ArrayList<CustomerHobbyDto>();
        dtos = customerHobbyDao.selectAllCustomerAndHobby();
        return dtos;
    }
    public void insertDB(CustomerDto customerDto, HobbyDto hobbyDto){
        cutomerDao.insert(customerDto);
        hobbyDao.insert(hobbyDto);
    }
    public void update(int id, double height, String beforehobby, String afterhobby){
        cutomerDao.update(id, height);
        hobbyDao.update(id, beforehobby, afterhobby);
    }
    public void deleteHobby(HobbyDto hobby){
        hobbyDao.delete(hobby.getId(), hobby.getHobby());
    }
    public void deleteCustomer(int id){
        hobbyDao.delete(id);
        cutomerDao.delete(id);
    }
}
//DBConnExCustomerHobby.java

```

```
package com.human.ex;
import java.util.ArrayList;
import com.human.dto.CustomerDto;
import com.human.dto.CustomerHobbyDto;
import com.human.dto.HobbyDto;
import com.human.model.Model;
import com.human.service.CustomerHobbyService;
import com.human.view.CustomerHobbyMenuInsertView;
import com.human.view.CustomerHobbyMenuSelectView;
import com.human.view.CustomerHobbyMenuUpdateView;
import com.human.view.CustomerMenuDeleteView;
import com.human.view.HobbyMenuDeleteView;
import com.human.view.MenuInputView;
import com.human.view.ViewInterface;
public class DBConnExCustomerHobby {
    static int menuState = 0;
    final static int DEFAULT = 0;
    final static int SELECT = 1;
    final static int INSERT = 2;
    final static int UPDATE = 3;
    final static int DELETEHB = 4;
    final static int DELETECM = 5;
    final static int EXIT = 6;
    public static void main(String[] args) {
        CustomerHobbyService service = new CustomerHobbyService();
        Model model = new Model();
        ViewInterface view = null;
        while(menuState != -1){
            view = new MenuInputView();
            view.execute(model);
            menuState = (int)model.getAttribute("input");
            switch (menuState) {
                case SELECT:
                    model = new Model();
                    ArrayList<CustomerHobbyDto> resultDots = service.selectAll();
                    model.setAttribute("resultDtos", resultDots);
                    view = new CustomerHobbyMenuSelectView();
                    view.execute(model);
                    menuState = DEFAULT;
                    break;
                case INSERT:
                    model = new Model();
                    view = new CustomerHobbyMenuInsertView();
                    view.execute(model);
                    service.insertDB((CustomerDto)model.getAttribute("customerdto"),
( HobbyDto)model.getAttribute("hobbydto"));
                    menuState = DEFAULT;
            }
        }
    }
}
```

```

        break;
    case UPDATE:
        model = new Model();
        view = new CustomerHobbyMenuUpdateView();
        view.execute(model);
        service.update((int)model.getAttribute("id"),
                      (double)model.getAttribute("height"),
                      (String)model.getAttribute("beforehobby"),
                      (String)model.getAttribute("afterhobby"));
        menuState = DEFAULT;
        break;
    case DELETEHB:
        model = new Model();
        view = new HobbyMenuDeleteView();
        view.execute(model);
        service.deleteHobby(new HobbyDto((int)model.getAttribute("id"),
                                         String)model.getAttribute("hobby")));
        menuState = DEFAULT;
        break;
    case DELETECM:
        model = new Model();
        view = new CustomerMenuDeleteView();
        view.execute(model);
        service.deleteCustomer((int)model.getAttribute("id"));
        break;
    case EXIT:
        System.out.println("종료");
        menuState = -1;
        break;
    default:
        System.out.println("잘못된 입력");
        menuState = DEFAULT;
        break;
    }
}
}

//CustomerHobbyMenuInsertView.java
package com.human.view;
import com.human.dto.CustomerDto;
import com.human.dto.HobbyDto;
import com.human.model.Model;
import com.human.util.UserInput;
public class CustomerHobbyMenuInsertView implements ViewInterface{
    @Override
    public void execute(Model model) {
        CustomerDto customerdto = new CustomerDto();

```

```

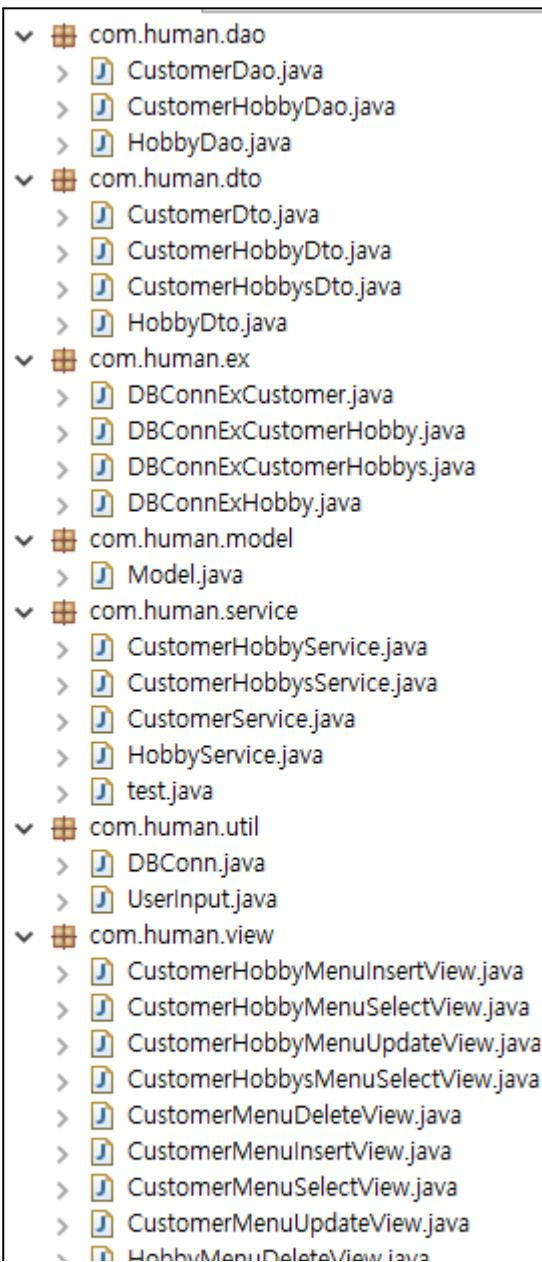
        customerdto.setName(UserInput.inputString("이름"));
        customerdto.setHeight(UserInput.inputDouble("키"));
        customerdto.setBirthday(UserInput.inputLocalDateTime("생일"));
        HobbyDto hobbydto = new HobbyDto();
        hobbydto.setId(customerdto.getId());
        hobbydto.setHobby(UserInput.inputString("취미"));
        model.setAttribute("customerdto", customerdto);
        model.setAttribute("hobbydto", hobbydto);
    }
}

//CustomerHobbyMenuSelectView.java
package com.human.view;
import java.util.ArrayList;
import com.human.dto.CustomerHobbyDto;
import com.human.model.Model;
public class CustomerHobbyMenuSelectView implements ViewInterface{
    @Override
    public void execute(Model model) {
        for(CustomerHobbyDto dto :
        (ArrayList<CustomerHobbyDto>)model.getAttribute("resultDtos"))
            System.out.println(dto);
    }
}

//CustomerHobbyMenuUpdateView.java
package com.human.view;
import com.human.model.Model;
import com.human.util.UserInput;
public class CustomerHobbyMenuUpdateView implements ViewInterface{
    @Override
    public void execute(Model model) {
        int id = UserInput.inputInt("찾을 id");
        double height = UserInput.inputDouble("바꿀 키");
        String beforehobby = UserInput.inputString("예전취미");
        String afterhobby = UserInput.inputString("바꿀 취미");
        model.setAttribute("id", id);
        model.setAttribute("height", height);
        model.setAttribute("beforehobby", beforehobby);
        model.setAttribute("afterhobby", afterhobby);
    }
}
}

```

> 12. customerhobbys



```
//CustomerHobbysDto.java
package com.human.dto;
import java.time.LocalDateTime;
import java.util.List;
public class CustomerHobbysDto {
    private int id;
    private String name;
    private double height;
    private LocalDateTime birthday;
    private List<HobbyDto> hobbys;
    public CustomerHobbysDto() {}
    public CustomerHobbysDto(int id, String
name, double height, LocalDateTime
birthday, List<HobbyDto> hobbys) {
        super();
        this.id = id;
        this.name = name;
        this.height = height;
        this.birthday = birthday;
        this.hobbys = hobbys;
    }
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + id;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        result = prime * result + (int) (height ^ (height > 0 ? 1 : -1));
        result = prime * result + ((birthday == null) ? 0 : birthday.hashCode());
        result = prime * result + ((hobbys == null) ? 0 : hobbys.hashCode());
        return result;
    }
}
```

```

        result = prime * result + ((birthday == null) ? 0 :
birthday.hashCode());
        long temp;
        temp = Double.doubleToLongBits(height);
        result = prime * result + (int) (temp ^ (temp >>> 32));
        result = prime * result + ((hobbys == null) ? 0 :
hobbys.hashCode());
        result = prime * result + id;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
    return result;
}
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    CustomerHobbysDto other = (CustomerHobbysDto) obj;
    if (birthday == null) {
        if (other.birthday != null)
            return false;
    } else if (!birthday.equals(other.birthday))
        return false;
    if (Double.doubleToLongBits(height) !=
Double.doubleToLongBits(other.height))
        return false;
    if (hobbys == null) {
        if (other.hobbys != null)
            return false;
    } else if (!hobbys.equals(other.hobbys))
        return false;
    if (id != other.id)
        return false;
    if (name == null) {
        if (other.name != null)
            return false;
    } else if (!name.equals(other.name))
        return false;
    return true;
}
public int getId() {      return id;  }
public void setId(int id) {      this.id = id;      }
public String getName() {      return name;      }
public void setName(String name) {      this.name = name; }
public double getHeight() {      return height;      }

```

```

        public void setHeight(double height) {           this.height = height;}
        public LocalDateTime getBirthday() {
            return birthday;
        }
        public void setBirthday(LocalDateTime birthday) {
            this.birthday = birthday;
        }
        public List<HobbyDto> getHobbys() {
            return hobbys;
        }
        public void setHobbys(List<HobbyDto> hobbys) {
            this.hobbys = hobbys;
        }
    }
    @Override
    public String toString() {
        return "CustomerHobbysDto [id=" + id + ", name=" + name +",
height=" + height + ", birthday=" + birthday
                + ", hobbys=" + hobbys + "]";
    }
    public CustomerDto toCustomerDto() {
        return new CustomerDto(id, name, height, birthday);
    }
}
//CustomerHobbysService.java
package com.human.service;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import com.human.dao.CustomerDao;
import com.human.dao.CustomerHobbyDao;
import com.human.dao.HobbyDao;
import com.human.dto.CustomerDto;
import com.human.dto.CustomerHobbysDto;
import com.human.dto.HobbyDto;
import com.human.util.DBConn;
public class CustomerHobbysService {
    public CustomerDao customerDao = new CustomerDao();
    public HobbyDao hobbyDao = new HobbyDao();
    public CustomerHobbyDao customerHobbyDao = new CustomerHobbyDao();
    public void insertDB(CustomerDto customerDto, HobbyDto hobbyDto){
        customerDao.insert(customerDto);
        hobbyDao.insert(hobbyDto);
    }
    public ArrayList<CustomerHobbysDto> selectAll(){
        ArrayList<CustomerHobbysDto> customerhobbysDtos = new
ArrayList<CustomerHobbysDto>();

```

```

        ArrayList<CustomerDto> customerDtos = new
ArrayList<CustomerDto>();
        customerDtos = customerDao.selectAll();
        for(CustomerDto dto : customerDtos){
            ArrayList<HobbyDto> hobbyDtos = hobbyDao.selectId(dto.getId());
            customerhobbysDtos.add(new CustomerHobbysDto(
                dto.getId(),dto.getName(),
                dto.getHeight(), dto.getBirthday(),
                hobbyDtos));
        }
        return customerhobbysDtos;
    }
    public void update(int id, double height, String beforehobby, String
afterhobby){
        customerDao.update(id, height);
        hobbyDao.update(id, beforehobby, afterhobby);
    }
    public void deleteHobby(HobbyDto hobby){
        hobbyDao.delete(hobby.getId(), hobby.getHobby());
    }
    public void deleteCustomer(int id){
        hobbyDao.delete(id);
        customerDao.delete(id);
    }
    public void insertDB(CustomerHobbysDto dto) {
//시퀀스 currval를 사용하려면 nextVal를 사용한 후 사용할 수 있다.
//1.DBConn.createStatement("SELECT id_counter.nextVal FROM dual");
//2.ResultSet rs = DBConn.createStatement("SELECT id_counter.CURRVAL FROM
dual");
// 1,2코드 둘다 실행하면 currval를 사용할 수 있지만 2번만 사용하면 사용할 수 없다.
//이런 문제를 해결하기 위해서 가장 마지막에 넣은값이 가장 크므로
customerDao.getMaxId();로 원하는
currval를 얻어 올수 있다.
//완벽하게 동작하려면 해당 메소드는 트랜잭션 처리를 해줘야한다.
        customerDao.insert(dto.toCustomerDto());
        int maxId=customerDao.getMaxId();
        for(HobbyDto hobbyDto: dto.getHobbys()) {
            hobbyDto.setId(maxId);
            hobbyDao.insert(hobbyDto);
        }
    }
}
//CustomerHobbysMenuSelectView.java
package com.human.view;
import java.util.ArrayList;
import com.human.dto.CustomerHobbysDto;
import com.human.model.Model;

```

```

public class CustomerHobbysMenuSelectView implements ViewInterface{
    @Override
    public void execute(Model model) {
        for(CustomerHobbysDto dto :
            (ArrayList<CustomerHobbysDto>)model.getAttribute("resultDtos"))
            System.out.println(dto);
    }
}

//DBConnExCustomerHobbys.java
package com.human.ex;
import java.util.ArrayList;
import com.human.dto.CustomerDto;
import com.human.dto.CustomerHobbysDto;
import com.human.dto.HobbyDto;
import com.human.model.Model;
import com.human.service.CustomerHobbysService;
import com.human.view.CustomerHobbyMenuInsertView;
import com.human.view.CustomerHobbyMenuUpdateView;
import com.human.view.CustomerHobbysMenuSelectView;
import com.human.view.CustomerMenuDeleteView;
import com.human.view.HobbyMenuDeleteView;
import com.human.view.MenuInputView;
import com.human.view.ViewInterface;
public class DBConnExCustomerHobbys {
    static int menuState = 0;
    final static int DEFAULT = 0;
    final static int SELECT = 1;
    final static int INSERT = 2;
    final static int UPDATE = 3;
    final static int DELETEHB = 4;
    final static int DELETECM = 5;
    final static int EXIT = 6;
    public static void main(String[] args) {
        CustomerHobbysService service = new CustomerHobbysService();
        Model model = new Model();
        ViewInterface view = null;

        while(menuState != -1){
            view = new MenuInputView();
            view.execute(model);
            menuState = (int)model.getAttribute("input");
            switch (menuState) {
                case SELECT:
                    model = new Model();
                    ArrayList<CustomerHobbysDto> resultDots =
service.selectAll();
                    model.setAttribute("resultDtos", resultDots);
            }
        }
    }
}

```

```

        view = new CustomerHobbysMenuSelectView();
        view.execute(model);
        menuState = DEFAULT;
        break;

    case INSERT:
        model = new Model();
        view = new CustomerHobbyMenuInsertView();
        view.execute(model);
        service.insertDB((CustomerHobbysDto)model.getAttribute("customerHobbysDto"));
        menuState = DEFAULT;
        break;
    case UPDATE:
        model = new Model();
        view = new CustomerHobbyMenuUpdateView();
        view.execute(model);
        service.update((int)model.getAttribute("id"),
                      (double)model.getAttribute("height"),
                      (String)model.getAttribute("beforehobby"),
                      (String)model.getAttribute("afterhobby"));
        menuState = DEFAULT;
        break;
    case DELETEHB:
        model = new Model();
        view = new HobbyMenuDeleteView();
        view.execute(model);
        service.deleteHobby(new
HobbyDto((int)model.getAttribute("id"),
          (String)model.getAttribute("hobby")));
        menuState = DEFAULT;
        break;
    case DELETECM:
        model = new Model();
        view = new CustomerMenuDeleteView();
        view.execute(model);
        service.deleteCustomer((int)model.getAttribute("id"));
        break;
    case EXIT:
        System.out.println("종료");
        menuState = -1;
        break;
    default:
        System.out.println("잘못된 입력");
        menuState = DEFAULT;
        break;
    }
}

```

```
    }  
}
```

> 17. 트랜잭션

다음은 JDBC로 트랜잭션 하는 방법이다.

트랜잭션이란? 여러 개의 sql문 작업을 하나의 작업 처럼 만드는 방법이다.

자세히 설명 하자면 하나의 insert 문을 실행 시키면 이것은 하나의 작업이고 실행에 성공하면 데이터베이스에 데이터가 들어가고 작업에 실패 하면 데이터가 들어가 있지 않을 것이다.

insert문 2개를 실행 시킬때 하나는 맞고 하나는 틀렸을 경우 맞은 데이터베이스에 들어가고 틀린 데이터는 데이터베이스에 들어가지 않을 것이다. 하지만, 둘중에 하나만 실행되면 안되는 상황이라면 둘다 맞을 때만 데이터 베이스에 둘다 넣고, 둘중 하나라도 실행되지 않은 데이터가 있을때 둘다 실행되지 않게 해야 한다. 이럴 때 2개의 insert문을 하나로 묶어 하나의 실행단위가 되도록 해주어야 하는데 이런 작업들을 트랜잭션이라고 한다. 트랜잭션은 여러개의 작업을 쪼갤수 없는 하나의 작업으로 만드는 것을 의미한다.

트랜잭션 용어중 commit과 rollback이 있는데 commit은 지금까지 작업한 내용을 적용한다는 의미이고 rollback은 모두 취소 한다는 이야기이다.

한명이 다른 사람에게 입금해 주는 프로그램에서 본인 계좌에서 다른 사람 계좌로 보내려면 2개 이상의 sql이 필요하고 둘다 실행 되거나 실행 되지 않는다면 문제가 발생 할 것이다. 이때 트랜잭션을 사용하여 2개의 작업을 묶는다면 하나만 실행 되는 일은 없을 것이다.

다음 코드는 트랜잭션을 사용하여 2개의 데이터를 DB에 넣는 작업을 하는 코드이다. 트랜잭션으로 묶었기 때문에 둘다 실행되거나 둘다 실행되지 않는 경우는 있을 수 있어도 둘중 하나만 실행되는 경우는 있을 수 없다.

프로그램에서는 기본적으로 auto commit 이어서 자동으로 sql문이 데이터베이스에 적용된다. 따라서, 기존 사용한 방식대로 insert문을 2개 전송하면 insert문 각각 따로 따로 트랜잭션이 적용된다. insert문 2개가 하나의 트랜잭션이 적용되게 하려면 auto

commit이 되지 않게 만든 다음 2개의 insert문을 전송하고 강제로 commit시켜야 한다.

수동으로 commit상태를 설정 하려면 19번 라인 처럼 트랜잭션 수동 설정을 하여야 한다.
수동 설정 이후 하는 작업들은 모두 하나의 트랜잭션이되어서 도중에 문제가 발생한다면
모두 취소된다. 모든 작업이 정상 마무리 되어 적용시키고 싶다면 35번 라인 처럼
commit해주어야 하고 중간에 문제가 발생해서 작업을 취소 하려면 39번 라인 처럼
rollback해 주어야 한다. 19,35번 라인 둘다 SQLException 예외처리 하여야 한다.

```
1 package com.human.ex;
2 import java.sql.Connection;□
10 public class JDBCExinsert3 {
11     public static void main(String[] args) {
12         Connection conn=null;
13         Statement st=null;
14         try {
15             Class.forName("oracle.jdbc.driver.OracleDriver");
16             String url="jdbc:oracle:thin:@localhost:1521:xe";
17             String user="human";      String pw="human";
18             conn=DriverManager.getConnection(url,user,pw);
19             conn.setAutoCommit(false); //commit설정
20             st=conn.createStatement();
21             int n=st.executeUpdate("insert into member values"
22                             + "(6,'100','a4@hanmail.net','kang')");
23             if(n==1) { //실행후 변경된 데이터의 개수가 리턴된다.
24                 System.out.println("정상동작");
25             }else { //데이터 1개를 넣었으니 데이터가 1개 변경되면 정상, 아니면 비정상
26                 System.out.println("비정상동작");
27             }
28             n=st.executeUpdate("insert into member values"
29                             + "(7,'100','a4@hanmail.net','kang')");
30             if(n==1) { //실행후 변경된 데이터의 개수가 리턴된다.
31                 System.out.println("정상동작");
32             }else { //데이터 1개를 넣었으니 데이터가 1개 변경되면 정상, 아니면 비정상
33                 System.out.println("비정상동작");
34             }
35             conn.commit(); //정상완료
```

```

36         }catch (Exception e) {
37             e.printStackTrace();
38             try {
39                 conn.rollback(); //취소
40             } catch (SQLException e1) { // rollback에 대한 예외처리
41                 e1.printStackTrace();
42             }
43         }finally {
44             try {
45                 if(st!=null)st.close();
46                 if(conn!=null)conn.close();
47             } catch (SQLException e) {
48                 e.printStackTrace();
49             }
50         }
51         System.out.println("종료");
52     }
53 }
```

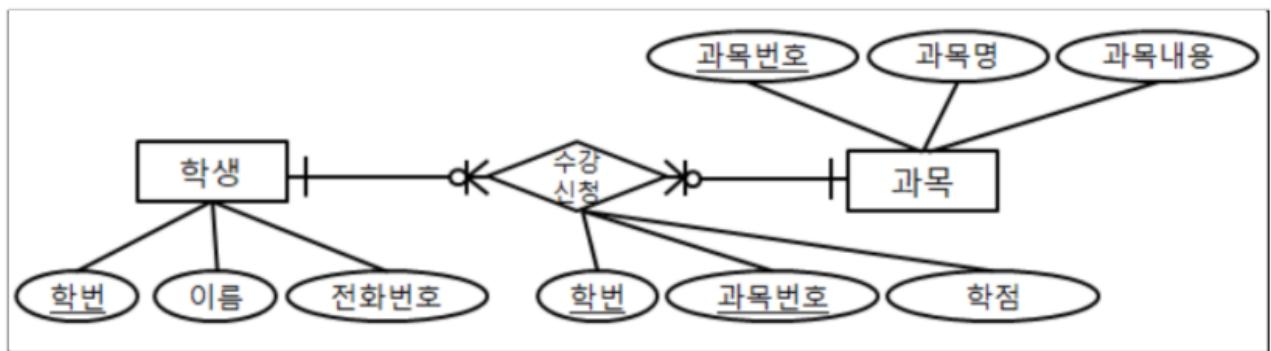
19,35,38~42 라인을 주석하여 트랜잭션 처리를 제거한 다음 28번 라인의 sql문을 틀리게 변경한 다음 실행 시켜보면 21번 라인에서 실행 시킨 데이터만 들어가 있는 것을 확인 할 수 있다.

sqldeveloper를 사용하여 테이블의 내용을 삭제하고 반드시 commit 한 다음에 이전 주석한 부분을 풀고 28번 라인에 문제가 있는 상태에서 실행 시키면 다른 쪽이 정상적인 sql문을 가지고 있다고 하여도 둘다 실행되지 않아 아무것도 들어가 있지 않을 것이다.

결과적으로 주석한 부분을 풀고 실행 시키면 트랜잭션이 적용되어 2개의 insert문이 둘다 적용되거나 둘다 취소 된다.

하지만, 주석을 하고 실행 시키면 둘중에 하나만 실행되어 DB에 데이터가 데이터가 1개만 들어가는 일도 생길 수 있다. 이런 상황이 일어나지 않게 하기 위해서 트랜잭션을 사용한다.

> 13. 수강신청 프로젝트



상위 ERD를 사용하여 수강신청 프로젝트를 완성해보자.

그동안 자바를 공부하느냐 고생이 많았습니다. 이 책은 웹 프로그램에 필요한 최소한의 내용을 다루어 처음 자바를 공부하는 사람들에게 도움을 주려고 많은 내용들을 다루고 있지 않다. 나중에 좀더 전문도서를 읽어보고 여유가 된다면 시간을 내어서 자바로 자료구조, 스레드, 네트워크 순으로 좀 더 공부를 해 보자. 자료구조를 먼저 공부하고 스레드는 그냥 간단히 뭔지 확인하고 네트워크로 채팅 프로그램을 만들어 보자.

프로그램은 항상 변화한다 jdbc를 깊이있게 공부할 필요는 없다. 이미 현장에서 jdbc를 사용하는 회사는 없다. 하지만, 프로그램은 돌고 돈다. 새롭게 등장한 방법을 적응하는데에는 jdbc가 가장 현명한 선택일 수 있을 것이다.

간단히 뭔지 확인하고 네트워크로 채팅 프로그램을 만들어 보자.

다음표를 통해서 10진수가 2진수로 표기되는 것을 확인할 수 있다.

구 분	10진수	2진수	8진수	16진수	5진수
0	0	0	0	0	0
1	1	1	1	1	1
2	2	10	2	2	2
3	3	11	3	3	3
4	4	100	4	4	4
5	5	101	5	5	10
6	6	110	6	6	11
7	7	111	7	7	12
8	8	1000	10	8	13
9	9	1001	11	9	14
10	10	1010	12	A	20
11	11	1011	13	B	21
12	12	1100	14	C	22
13	13	1101	15	D	23
14	14	1110	16	E	24
15	15	1111	17	F	30
16	16	10000	20	10	31
17	17	10001	21	11	32
18	18	10010	22	12	33
19	19	10011	23	13	34
20	20	10100	24	14	40

진수란? 수를 표현하는 여러가지 방법을 의미하는데 방법을 살펴보면, 10진수란? 0~9까지 10개의 문자를 가지고 수를 표현하는 방법이고 2진수란? 0, 1 2개의 문자를 가지고 수를 표현하는 방법이고 8진수란? 0~7까지 8개의 문자를 가지고 수를 표현하는 방법이다. 우리가 현실세계에서 사용하고 있는 숫자는 10진수이고 아래 표를 확인하여 이해해 보자. 2진수를 10진수로 10진수를 2진수로 8진수를 10진수로 진수 변환 방법은 인터넷 웹사이트를 검사해서 공부해 보자.

현실세계의 데이터를 컴퓨터에 넣는 방법은 숫자 데이터는 컴퓨터에 넣을 때 0과 1만 사용할 수 있기 때문에 2진수로 바꾸어서 사용하고 문자 같은 경우 문자 코드표를 사용하여 2진수로 바꾸어서 컴퓨터에서 사용한다.

코드표란? 약속된 기호를 기록한 표이다. 예을 들면 1을 a, 2를 b, 3를 n이라 약속 한다면 바나나를 기록 표시할때 banana를 213131과 같은 숫자로 표현한 결과로도 바나나라는 것을 이해할 것이다. 이처럼 현실세계의 문자를 2진수로 표현할 수 있도록 기록해 놓은 대표적인 문자 코드표가 아스키코드와 유니코드이다. 초창기 컴퓨터 시스템에서는 아스키코드를 사용하였지만 더 많은 문자를 처리하기 위해서 요즘은 표준으로 유니코드를 사용한다.

다음표는 아스키 코드표이다. 맨오른쪽 줄 char 항목을 살펴보면 빨간줄로 알파벳이 있다. 현실세계에서 표현하는 문자를 의미하는 것이고 표에서 문자 ‘a’를 찾아 보면 컴퓨터에 저장할 때는 10진수 97을 2진수로 바꾸어 저장 된다. 숫자 97과 ‘a’는 컴퓨터에 저장할 때 같은 숫자 97의 2진수로 저장되기 때문에 저장된 모양만 가지고는 숫자인지 문자인지 알수 없고, 자료형이 무엇인지 알아야 명확하게 구분할 수 있다.

왼쪽 hex(헥사)는 16진수, Dec(데시밀) 10진수, oct(액터) 8진수, bin(바이너리) 2진수를 의미한다.

문자를 숫자로 표현하고 있어서 컴퓨터 프로그램에서 ‘a’+1 하면 98이 리턴된다.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0 000	000	MUL (null)	32	20 040	000	 	Space	64	40 100	000	@	Ø	96	60 140	000	`	`
1	1 001	001	SOH (start of heading)	33	21 041	001	!	!	65	41 101	001	A	A	97	61 141	001	a	a
2	2 002	002	STX (start of text)	34	22 042	002	"	"	66	42 102	002	B	B	98	62 142	002	b	b
3	3 003	003	ETX (end of text)	35	23 043	003	#	#	67	43 103	003	C	C	99	63 143	003	c	c
4	4 004	004	EOT (end of transmission)	36	24 044	004	$	\$	68	44 104	004	D	D	100	64 144	004	d	d
5	5 005	005	ENQ (enquiry)	37	25 045	005	%	%	69	45 105	005	E	E	101	65 145	005	e	e
6	6 006	006	ACK (acknowledge)	38	26 046	006	&	&	70	46 106	006	F	F	102	66 146	006	f	f
7	7 007	007	BEL (bell)	39	27 047	007	'	'	71	47 107	007	G	G	103	67 147	007	g	g
8	8 010	010	BS (backspace)	40	28 050	010	((72	48 110	010	H	H	104	68 150	010	h	h
9	9 011	011	TAB (horizontal tab)	41	29 051	011))	73	49 111	011	I	I	105	69 151	011	i	i
10	A 012	012	LF (NL line feed, new line)	42	2A 052	012	*	*	74	4A 112	012	J	J	106	6A 152	012	j	j
11	B 013	013	VT (vertical tab)	43	2B 053	013	+	+	75	4B 113	013	K	K	107	6B 153	013	k	k
12	C 014	014	FF (NP form feed, new page)	44	2C 054	014	,	,	76	4C 114	014	L	L	108	6C 154	014	l	l
13	D 015	015	CR (carriage return)	45	2D 055	015	-	-	77	4D 115	015	M	M	109	6D 155	015	m	m
14	E 016	016	SU (shift out)	46	2E 056	016	.	.	78	4E 116	016	N	N	110	6E 156	016	n	n
15	F 017	017	SI (shift in)	47	2F 057	017	/	/	79	4F 117	017	O	O	111	6F 157	017	o	o
16	10 020	020	DLE (data link escape)	48	30 060	020	0	0	80	50 120	020	P	P	112	70 160	020	p	p
17	11 021	021	DC1 (device control 1)	49	31 061	021	1	1	81	51 121	021	Q	Q	113	71 161	021	q	q
18	12 022	022	DC2 (device control 2)	50	32 062	022	2	2	82	52 122	022	R	R	114	72 162	022	r	r
19	13 023	023	DC3 (device control 3)	51	33 063	023	3	3	83	53 123	023	S	S	115	73 163	023	s	s
20	14 024	024	DC4 (device control 4)	52	34 064	024	4	4	84	54 124	024	T	T	116	74 164	024	t	t
21	15 025	025	NAK (negative acknowledge)	53	35 065	025	5	5	85	55 125	025	U	U	117	75 165	025	u	u
22	16 026	026	SYN (synchronous idle)	54	36 066	026	6	6	86	56 126	026	V	V	118	76 166	026	v	v
23	17 027	027	ETB (end of trans. block)	55	37 067	027	7	7	87	57 127	027	W	W	119	77 167	027	w	w
24	18 030	030	CAN (cancel)	56	38 070	030	8	8	88	58 130	030	X	X	120	78 170	030	x	x
25	19 031	031	EM (end of medium)	57	39 071	031	9	9	89	59 131	031	Y	Y	121	79 171	031	y	y
26	1A 032	032	SUB (substitute)	58	3A 072	032	:	:	90	5A 132	032	Z	Z	122	7A 172	032	z	z
27	1B 033	033	ESC (escape)	59	3B 073	033	;	:	91	5B 133	033	[[123	7B 173	033	{	{
28	1C 034	034	FS (file separator)	60	3C 074	034	<	<	92	5C 134	034	\	\	124	7C 174	034	|	
29	1D 035	035	GS (group separator)	61	3D 075	035	=	=	93	5D 135	035]]	125	7D 175	035	}	}
30	1E 036	036	RS (record separator)	62	3E 076	036	>	>	94	5E 136	036	^	^	126	7E 176	036	~	~
31	1F 037	037	US (unit separator)	63	3F 077	037	?	?	95	5F 137	037	_	_	127	7F 177	037		DEL

Source: www.LookupTables.com

종종 프로그램 하다 보면 한글이 깨지는 경우가 종종 있다. 문자 ‘a’를 A코드 표에서는 97, B코드표에서는 198로 표시한다고 생각해보자. A코드표를 가지고 숫자로 만들 었다면 다시 화면에 보여줘야 할 때 숫자를 A코드 표를 이용해서 문자로 보여 줘야 이전 문자와 동일하게 나올 것이다. B코드 표를 이용해서 보여주면 처음에 같은 문자가 다른 숫자로 정의되어 있을 것이다. 이런 이유에서 글자가 깨져보이는 일이 종종 있고 이런 숫자에서 문자로 바꾸는 과정을 encoding(인코딩), 반대의 경우 decoding(디코딩)이라고 한다. 종종 인코딩 문제로 화면의 한글이 깨지는 일이 종종 있는데 해결하는 방법은 쉽지 않다. 그때 그때 증상을 웹에 검색해서 해결하는 것이 가장 쉽고 현명한 방법이다. 과거에는 아스키코드를 사용 하였지만 최근에는 UTF-8(유니코드)를 사용 하고 종종 euc-kr로 되어 있는 문서도 있는데 과거에 사용하던 코드방식이다.

CJK Radicals Supplement
CJK Strokes
Ideographic Description Characters
Hangul Jamo
Hangul Jamo Extended-A
Hangul Jamo Extended-B
Hangul Compatibility Jamo
Halfwidth Jamo
Hangul Syllables
Hiragana
Kana Extended-A

다음 유니코드를 살펴보자.

과거에는 컴퓨터를 사용할 때 영어만 사용하였기 때문에 아스키코드로 충분히 처리할 수 있었지만 지금은 수많은 나라의 언어를 컴퓨터에서 다 처리 해야 하기 때문에 유니코드를 사용한다. 유니코드는 한글자를 표현할 때 2byte가 필요하다. 우리가 배울 자바도 아스키코드를 사용하지 않고 유니코드를 사용한다.

<https://www.unicode.org/charts/index.html> 다음사이트에 들어가서 중간쯤에 한글문자관련 유니코드값 정보를 얻을 수있는 pdf파일을 얻을 수있다. 아래그림에서 빨간줄

부분을 클릭해보자. 링크가 변경되었다면 검색해서 잘 찾아보자.

AC00 Hangul Syllables												
	AC0	AC1	AC2	AC3	AC4	AC5	AC6	AC7	AC8	AC9	ACA	AC
0	가	감	갠	갰	갈	깎	꺊	거	검	겐	겠	긱
	AC00	AC10	AC20	AC30	AC40	AC50	AC60	AC70	AC80	AC90	ACA0	AC10
1	각	갑	갰	ڱ	꺃	꺃	꺃	걱	겁	겠	ڱ	긱
	AC01	AC11	AC21	AC31	AC41	AC51	AC61	AC71	AC81	AC91	ACA1	AC11
2	깎	꺃	꺃	꺃	꺃	꺃	꺃	꺾	겁	꺃	꺃	긱
	AC02	AC12	AC22	AC32	AC42	AC52	AC62	AC72	AC82	AC92	ACA2	AC12
3	꺃	갓	깬	꺃	꺃	꺃	꺃	겼	것	겐	꺃	긱
	AC03	AC13	AC23	AC33	AC43	AC53	AC63	AC73	AC83	AC93	ACA3	AC13
4	간	갓	깰	깰	꺃	꺃	꺃	꺃	건	겄	겔	꺃
	AC04	AC14	AC24	AC34	AC44	AC54	AC64	AC74	AC84	AC94	ACA4	AC14
5	꺃	강	꺃	꺃	꺃	꺃	꺃	겼	경	꺃	꺃	긱
	AC05	AC15	AC25	AC35	AC45	AC55	AC65	AC75	AC85	AC95	ACA5	AC15
6	꺃	갓	꺃	꺃	꺃	꺃	꺃	겼	것	꺃	꺃	긱
	AC06	AC16	AC26	AC36	AC46	AC56	AC66	AC76	AC86	AC96	ACA6	AC16
7	간	갓	꺃	꺃	꺃	꺃	꺃	꺃	겟	꺃	꺃	긱
	AC07	AC17	AC27	AC37	AC47	AC57	AC67	AC77	AC87	AC97	ACA7	AC17
.	가	가	가	가	가	가	가	거	거	거	거	가
	AC08	AC18	AC28	AC38	AC48	AC58	AC68	AC78	AC88	AC98	ACA8	AC18

왼쪽 표를 가지고 해당 문자의 유니코드값을 얻을 수 있다.
AC00은 ‘가’에 해당하며 ‘걸’은 AC77이다. 2진수로 표현하면 길어져서 보기 쉽게 16진수로 표현한 것이다. 16진수 8진수를 따로 공부하는 이유도 이런 이유에서이다.

1010 1100 0111 0111 이렇게 2진수를 사용해서 기술하는 것보다는 16진수 AC00이 짧고 명확하게 보일 것이다. 그리고 2진수 4자리는 명확하게 16진수 1자리와 매핑 된다. 여기서 매핑이란? 1:1로 연결을 의미한다. 0000 = 0, 0001 = 1, 0010 = 2, 0011 = 3, ..., 0111=7, ..., 1010=A, 1011=B, 1100=C, ... 1111=F 다음과 같이 1:1 매핑된다. 따라서 AC77만 보고도 특별한

과정을 거치지 않고도 A=1010 C=1100 7=0111 7=0111과 같이 2진수로 결과를 얻을 수 있다.

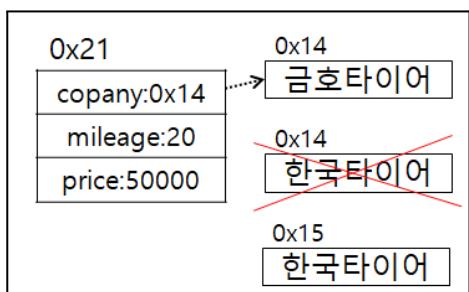
따라서, String 클래스안에 주소가 같은 값이 들어 있어도 변경이 불가능 하므로 참조 데이터의 값을 변경해서 생기는 문제는 발생하지 않는다.

결론은 String 클래스는 내부 문자열 변경이 불가능 하여 문자열 변경을 원하면 문자열을 새로 생성 하여 새로운 주소를 받아 변수에 넣는 방법만 가능하다. 기존 참조 데이터에서

발생하는 데이터 주소를 공유할때 데이터를 변경 하여 생기는 문제가 발생하지 않는다.

car 인스턴스를 통해 문자열 데이터를 변경하여도 문자열 주소 자체를 변경해야 하므로

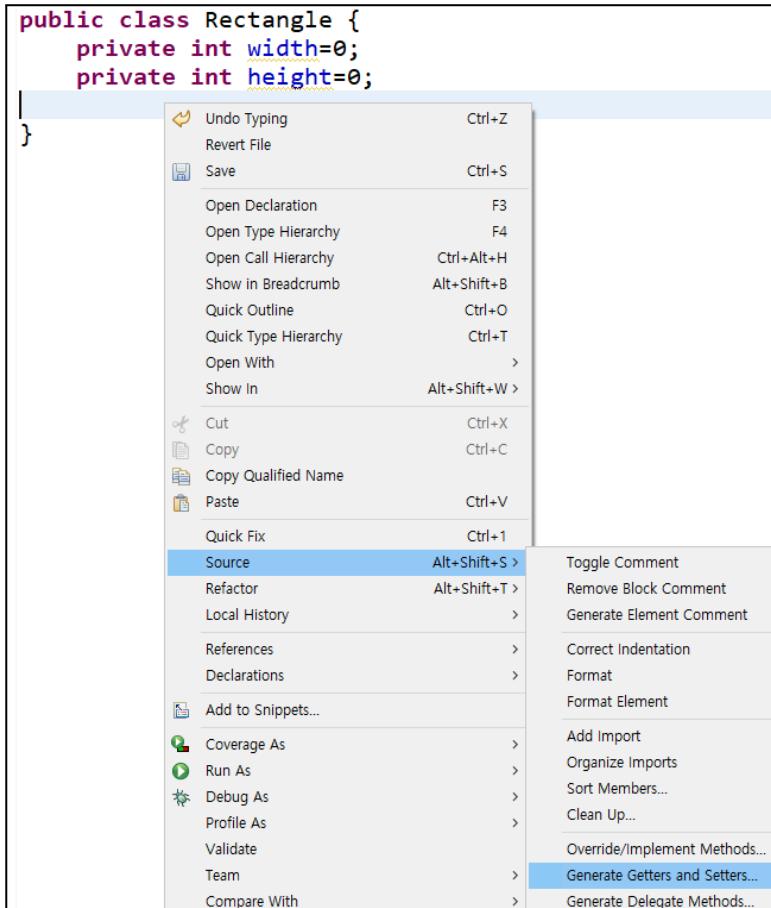
newCar 인스턴스에 데이터로 사용되고 있는 문자열에는 영향을 주지 않는다.



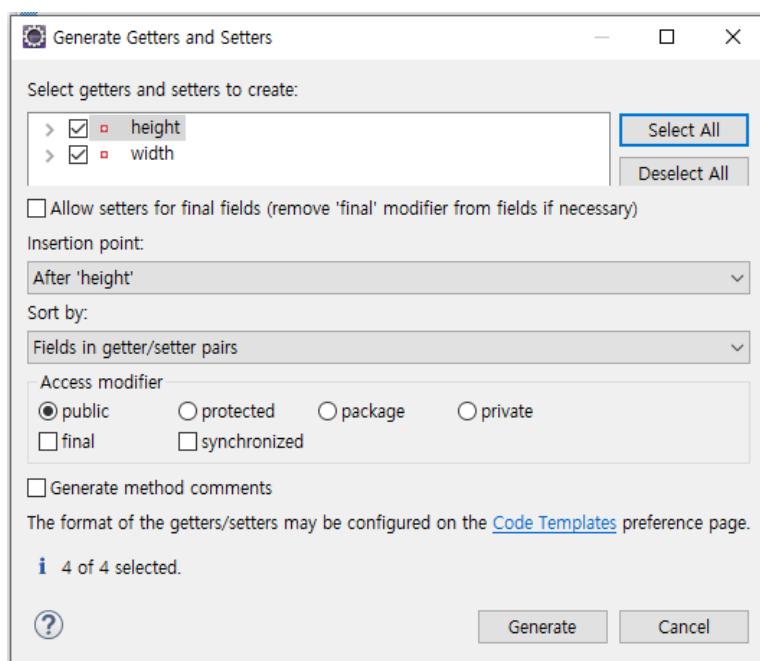
왼쪽 그림 처럼 0x14의 금호타이어라는 문자열 상수를 한국타이어라는 문자열로 변경하면 아래의 0x15의 한국타이어라는 문자열을 새로 만들어서 0x14 대신 0x15로 변경해 주어야 한다. 따라서, 다른 쪽에서 0x14를 쓰고 있더라도 0x14의 데이터가 변경된 것이 아니므로 영향을 받지 않는다. 0x14위치의 금호타이어를 0x14위치의 한국타이어로 변경하는 것은 불가능하다. 참조 자료형에서 발생하는 데이터를 변경해서 발생하는 문제는 `String` 클래스에서는 데이터 변경이 불가능 하므로 발생하지 않는다.

```
public class Person {
    private String name;
    private int age;
    // Getter 메소드
    public String getName() {
        return name;
    }
    // Setter 메소드
    public void setName(String name) {
        this.name = name;
    }
    // Getter 메소드
    public int getAge() {
        return age;
    }
    // Setter 메소드
    public void setAge(int age) {
        this.age = age;
    }
}
public class Main {
    public static void main(String[] args) {
        // Person 클래스의 객체 생성
        Person person = new Person();
        // Setter를 이용한 값 설정
        person.setName("홍길동");
        person.setAge(30);
        // Getter를 이용한 값 조회
        System.out.println("이름: " + person.getName());
        System.out.println("나이: " + person.getAge());
        // 다음과 같은 형태는 private 필드여서 접근할 수 없다.
        person.name="홍길동";
        person.age=30;
```

```
}
```



필드의 값을 변경 할 수 있다. 많이 사용되는 메소드여서



오버로딩은 같은 함수이름으로
매개변수가 다른 생성자와
메소드를 만드는 것을 이야기
한다. 다음 예제를 따라해서
hashCode, equals, 생성자,
toString 메소드, getter,
setter등을 추가해 보자.

getter와 setter 만들기

클래스의 인스턴스 필드를
private로 선언 하면 외부에서
직접 접근 할 수 없다고 하였다.
하지만 왼쪽 코드에서 사각형의
높이와 넓이를 변경하고 싶다면
어떻게 하여야 하는지 생각해
보자. 이렇게 해서 생겨난 것이
바로 getter, setter
메소드이다 getter는 해당
클래스의 private로 선언된
필드를 읽어 오고, setter
메소드는 private로 선언된

eclipse 툴에서 자동으로 만들어
주는 방법을 지원해준다. 상위
이미지 처럼 필드들을 기술한
다음에 private 필드가 있는
Rectangle 클래스안에서
오른쪽마우스클릭>>
source선택>>generate Getters
and Setter 항목을 선택 하자.
그러면 아래와 같은 이미지가
나올 것이다. select All버튼을
클릭하면 왼쪽 상단에 보이는
체크박스를 사용해 보여주고 있는
필드 목록에 모든 필드들이
선택되는 것을 볼수 있고 하단의
Generate 를 클릭하면 자동으로
추가된 코드를 확인 할 수 있다.
과거에 r1.height=30;

r1.width=40; 형태로 필드를 변경하였다면 이제는 읽어올때는 r1.getWidth()
r1.getHeight()로 읽어오고 값을 변경할 때에는 r1.setWidth(3)r1.setHeight(4)를

```
public class Rectangle {  
    private int width=0;  
    private int height=0;  
    public int getWidth() {  
        return width;  
    }  
    public void setWidth(int width) {  
        this.width = width;  
    }  
    public int getHeight() {  
        return height;  
    }  
    public void setHeight(int height) {  
        this.height = height;  
    }  
}
```

사용해서 변경 할 수 있다. private로 선언하고 getter, setter를 이용해서 복잡하게 코드를 만드는 이유는 안전성을 위해서이다. 넓이는 음수가 들어 갈 수 없다. width를 public으로 선언한다면 r1.width=-50이라는 값이 프로그램에 들어가서 문제가 발생 할 것이다. public으로 선언하면 외부에서 r1.width=-50과 같은 접근을 막을 수 있는 방법이 없다. private로 선언하면 r1.width=-50과 같은 형태로 width값을 변경하는 것을 막을 수 있고 r1.setWidth(-50)과 같은 형태의 로직은 왼쪽처럼 들어오는 값을 변경 할 수 있어서 width 변수에 외부에서 음수가 들어 오는 것을 완전히 막을 수 있다. 이러한 이유에서 getter setter를 사용 한다.

```
public void setWidth(int width) {  
    this.width = width;  
    if(this.width<0) {  
        this.width=0;  
    }  
}
```

그동안 만든것을 종합해서 Rectangle클래스를 만들어 보자.

1.com.human.dto 패키지에 Rectangle.java파일을 만들고 Rectangle클래스를 만든다.

```
public class Rectangle {  
    private double width=100;  
    private double height=100;  
  
    public double area() {  
        return width*height;  
    }  
}
```

```
11.  public double getWidth() {  
12      return width;  
13  }  
14.  public void setWidth(double width) {  
15      this.width = width;  
16      if(this.width<0)this.width=0;  
17  }  
18.  public double getHeight() {  
19      return height;  
20  }  
21.  public void setHeight(double height) {  
22      this.height = height;  
23      if(this.height<0)this.height=0;
```

2. source>> getter setter 를 이용해 getter, setter를 추가하고 setter에 0보다 작은 수가 들어가지 않도록 로직을 추가한다.

```
public Rectangle() {}
public Rectangle(double width, double height) {
    super();
    this.width = width;
    this.height = height;
    if(this.width<0)this.width=0;
    if(this.height<0)this.height=0;
}
```

3. source>>

constructor 를 이용해 생성자를 추가 한다.
생성자를 통해 음수가 들어오지 않도록 로직을 추가 하자.

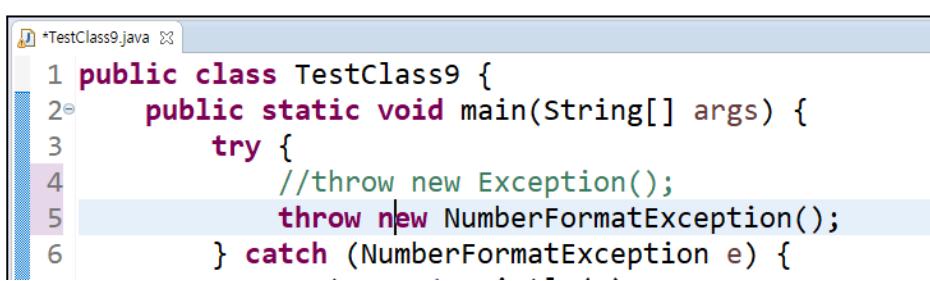
```
+ public String toString() {..}
+ public int hashCode() {..}
+ public boolean equals(Object obj) {..}
```

4. 객체비교와 객체문자열 출력에 필요한 메소드를 추가해 보자.

```
public class JavaStart {

    public static void main(String[] args) {
        Rectangle r=new Rectangle();
        r.setWidth(10);
        System.out.println(r.area());
    }
}
```

5. 메인에서 만든 클래스를 사용해 보자.



```
1 public class TestClass9 {
2     public static void main(String[] args) {
3         try {
4             //throw new Exception();
5             throw new NumberFormatException();
6         } catch (NumberFormatException e) {
7             e.printStackTrace();
8         }
9     }
10 }
```

일반적으로 예외는 프로그램이 동작중

어쩔수 없는 경우 스스로 발생시키지만 프로그래머가 throw 문을 사용하여 강제로 예외를 발생시킬 수 있다. 5번 라인처럼 기술하면 NumberFormat Exception이 발생하고 해당 예외의 처리부분의 catch 문으로 이동하므로 7,8번 라인이 실행된다.

만약 수학적 예외를 강제로 만들고 싶다면 throw new ArithmeticException ();이라고 기술하면 된다.

왼쪽에 예제는 a가 10보다 큰경우 강제로 예외를 발생하는 프로그램을 구현해 본것이다.

```
int a=15;

try {
    if(a>10) {
        //강제로 예외 발생
        throw new Exception();
    }else {
        a=a+5;
    }
}catch(Exception e) {
    System.out.println(
        "a가 10보다 커서 예외발생");
}

System.out.println(a);
```

throws는 메소드 안의 코드중 예외가 발생하였을때 예외처리를 해당 메소드에서 하지 않고 해당 메소드를 호출한 부분에서 예외처리를 넘기는 방법이다.

다음 코드를 통해서 throws를 이해해 보자.

```
public static void exceptionFunction1() throws Exception{
    throw new Exception();
}
public static void exceptionFunction2() throws Exception{
    throw new NumberFormatException();
}
public static void main(String[] args) throws Exception,NumberFormatException {
    try{
        exceptionFunction1();
    }catch(Exception e) {};
    exceptionFunction2();
}
```

exceptionFunction1() 안에서 throw new Exception();를 사용하여 예외가 발생하면 프로그램이 종료되기 때문에 예외 처리를 해주어야 한다. 하지만 상위 프로그램에서 예외처리를 하지 않았는데 프로그램이 멈추지 않고 잘 동작하는 것을 확인 할 수 있다. throws 다음에 예외처리가 필요한 예외클래스를 기술하면 메소드 안에서 해야할 예외처리를 해당 메소드를 호출한 부분에서 할 수 있다.

메인 안에서 try catch문으로 exceptionFunction1() 메소드 안에서 발생 할 수 있는 예외를 메소드를 호출한 곳에서 처리 한 것을 확인 할 수 있다.

main 메소드의 throws 다음에 기술한 예외 클래스도 마찬 가지로 main를 호출한 부분에서 처리 한다. main은 운영체제가 호출한 것이니 운영체제가 알아서 처리한다고 생각하면 된다.

main 메소드의 throws 의 기술로 exceptionFunction2()에서 발생한 예외를 main을 호출한 운영체제가 알아서 처리한다.

```
3 public class Java534 {  
4     public static int test1(int a) {  
5         try {  
6             if(a>10) {  
7                 throw new Exception();  
8             }else {  
9                 a=a+5;  
10            }  
11        }catch(Exception e) {  
12            System.out.println("a가 10보다 커서 예외발생");  
13            a=0;  
14        }  
15        return a;  
16    }  
17}
```

다음 이미지들은 이전 예제를 메소드로 만들어 throw와 throws 예제를 만든 것이다. test1 메소드의 경우 메소드 내부에서 예외처리가 되어서 호출만 하면 되는 상태이고 test2는 메소드 안에서 예외처리를 하지 않고 throws를

이용해서 호출한 곳에서 예외처리를 하도록 한 경우이다. 다음 코드를 확인해 보자.

```
17 public static int test2(int a) throws Exception {  
18     if(a>10) {  
19         throw new Exception();  
20     }else {  
21         a=a+5;  
22     }  
23     return a;  
24 }  
25 public static void main(String[] args) throws Exception {  
26     System.out.println(test1(5));  
27     try {  
28         System.out.println(test2(5));  
29     } catch (Exception e) {  
30         e.printStackTrace();  
31     }  
32 }  
33 }
```

06. 여러가지 살펴보기

> 02. hashMap 사용하기

다음은 컬렉션 중 하나인 해쉬맵을 사용하는 방법이다.

자바(Java)에서 HashMap은 java.util 패키지에 속한 클래스로, 키(key)로 값(value)을 저장하는데 사용됩니다.

키-값 쌍 저장: HashMap은 키와 값을 쌍으로 저장합니다. 각 키는 고유해야 하며, 하나의 키에 하나의 값이 있다.

데이터 순서 보장 안됨: HashMap은 데이터의 순서를 보장하지 않습니다. 삽입 순서나 정렬된 순서와 무관하게 데이터를 저장하고 관리합니다.

HashMap의 주요 메서드:

`put(K key, V value)`: 특정 키와 값을 매핑하여 HashMap에 추가합니다.

`replace(K key, V value)`: 특정 키와 값을 매핑하여 HashMap을 수정한다.

`get(Object key)`: 주어진 키에 해당하는 값을 반환합니다.

`remove(Object key)`: 주어진 키에 해당하는 키-값 쌍을 제거합니다.

`containsKey(Object key)`: 특정 키가 HashMap에 존재하는지 확인합니다.

`containsValue(Object value)`: 특정 값이 HashMap에 존재하는지 확인합니다.

`size()`: 현재 HashMap에 저장된 키-값 쌍의 개수를 반환합니다.

```
import java.util.HashMap;
public class HashMapExample {
    public static void main(String[] args) {
        // HashMap 생성
        HashMap<String, Integer> hashMap = new HashMap<>();
        // 값 추가
        hashMap.put("A", 10);
        hashMap.put("B", 20);
        hashMap.put("C", 30);
```

```

// 현재 HashMap 출력
System.out.println("HashMap: " + hashMap);
// 존재하는 키의 값을 변경
hashMap.replace("B", 25);
// 변경된 HashMap 출력
System.out.println("변경된 HashMap: " + hashMap);
// 존재하지 않는 키의 값을 변경 (아무 작업도 수행하지 않음)
hashMap.replace("D", 40);
// 변경되지 않은 HashMap 출력
System.out.println("변경되지 않은 HashMap: " + hashMap);
// 다른 주요 메서드 사용 예제
System.out.println("특정 키에 대한 값 조회: " + hashMap.get("A"));
System.out.println("존재하지 않는 키 조회: " + hashMap.get("X"));

System.out.println("HashMap에서 C 제거 전: " + hashMap);
hashMap.remove("C");
System.out.println("HashMap에서 C 제거 후: " + hashMap);
System.out.println("HashMap에 B가 존재: " + hashMap.containsKey("B"));
System.out.println("HashMap에 값 25가 존재: " + hashMap.containsValue(25));
System.out.println("현재 HashMap의 크기: " + hashMap.size());
}

}

```

다음 코드들을 확인해 보자.

```

import java.util.HashMap;
import java.util.Map;
public class HashMapExample {
    public static void main(String[] args) {
        // HashMap 생성
        HashMap<String, Integer> hashMap = new HashMap<>();
        // 값 추가 - 한글 문자열 키
        hashMap.put("일", 1);
        hashMap.put("이", 2);
        // 값 가져오기
        int value1 = hashMap.get("일");
        System.out.println("한글 키 '일'의 값: " + value1);
        int value2 = hashMap.get("이");
        System.out.println("한글 키 '이'의 값: " + value2);
        // 모든 키 출력 - 방법 1: keySet 사용
        System.out.println("HashMap의 키 출력 - 방법 1: keySet");
        for (String key : hashMap.keySet()) {
            System.out.println("Key:" + key + ", Value: " + hashMap.get(key));
        }
        // 모든 키 출력 - 방법 2: Iterator 사용
        System.out.println("HashMap의 키 출력 - 방법 2: Iterator");
        for (Map.Entry<String, Integer> entry : hashMap.entrySet()) {
            System.out.println("K:" + entry.getKey() + ", V:" + entry.getValue());
        }
    }
}

```

```

    }

    // keySet 출력
    System.out.println("HashMap의 keySet 출력:");
    System.out.println(hashMap.keySet());
}

}

import java.util.HashMap;
import java.util.Map;

public class HashMapExample {
    public static void main(String[] args) {
        // 도시와 인구를 저장하는 HashMap 생성
        HashMap<String, Integer> cityPopulation = new HashMap<>();
        // 도시와 인구 추가
        cityPopulation.put("서울", 9900000);
        cityPopulation.put("부산", 3500000);
        cityPopulation.put("인천", 2900000);
        cityPopulation.put("대구", 2500000);
        cityPopulation.put("광주", 1500000);
        // 특정 도시의 인구 출력
        System.out.println("서울의 인구: " + cityPopulation.get("서울"));
        // 도시 목록 출력 - 방법 1: keySet 사용
        System.out.println("도시 목록 - 방법 1: keySet");
        for (String city : cityPopulation.keySet()) {
            System.out.println(city + ":" + cityPopulation.get(city) + "명");
        }

        // 도시 목록 출력 - 방법 2: entrySet 사용
        System.out.println("도시 목록 - 방법 2: entrySet");
        for (Map.Entry<String, Integer> entry : cityPopulation.entrySet()) {
            System.out.println(entry.getKey()+":"+entry.getValue() + "명");
        }
        // 특정 도시의 인구 업데이트
        cityPopulation.put("서울", 10100000);
        // 업데이트된 도시 목록 출력
        System.out.println("업데이트된 도시 목록:");
        for (String city : cityPopulation.keySet()) {
            System.out.println(city+ ":" + cityPopulation.get(city) + "명");
        }
        // 특정 도시 제거
        cityPopulation.remove("부산");
        // 도시 목록 출력 - 업데이트 및 제거 후
        System.out.println("업데이트 및 제거 후 도시 목록:");
    }
}

```

```

        for (Map.Entry<String, Integer> entry : cityPopulation.entrySet()) {
            System.out.println(entry.getKey()+":"+entry.getValue() + "명");
        }
    }
}

```

값으로 객체를 담을 수 있다.

```

public class HashMapObjectValueExample {
    public static void main(String[] args) {
        // HashMap 생성
        HashMap<String, Person> personMap = new HashMap<>();
        // Person 객체 추가
        personMap.put("Java", new Person("Alice", 25));
        personMap.put("Python", new Person("Bob", 30));
        personMap.put("JavaScript", new Person("Charlie", 22));
        // 특정 키에 대한 Person 객체 가져오기
        Person javaPerson = personMap.get("Java");
        System.out.println("Java의 정보: " + javaPerson);
        // 특정 키 제거
        personMap.remove("Python");
        // replace 메서드 사용
        personMap.replace("JavaScript", new Person("David", 26));
        // 모든 키 출력 - 방법 1: keySet 사용
        System.out.println("HashMap의 키 출력 - 방법 1: keySet");
        for (String key : personMap.keySet()) {
            System.out.println("Key:"+key+", Value: " + personMap.get(key));
        }
        // 모든 키 출력 - 방법 2: Iterator 사용
        System.out.println("HashMap의 키 출력 - 방법 2: Iterator");
        Iterator<Map.Entry<String, Person>> iterator =
        personMap.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry<String, Person> entry = iterator.next();
            System.out.println("K:"+entry.getKey()+" , V:"+ entry.getValue());
        }
        // keySet 출력
        System.out.println("HashMap의 keySet 출력:");
        System.out.println(personMap.keySet());
    }
}

```

> 07. 이중 for문 사용하기

1. HashSet 개요

HashSet은 Java의 Set 인터페이스를 구현한 **중복을 허용하지 않는** 컬렉션입니다. 내부적으로 해시 테이블(Hash Table)을 사용하여 요소를 저장하며, 저장 순서를 보장하지 않습니다.

HashSet은 다음과 같은 특징을 가집니다:

중복된 요소를 허용하지 않음

빠른 검색($O(1)$ 성능)

순서 보장되지 않음

내부적으로 **HashMap<K, V>**를 사용하여 동작하며, **value** 부분에는 항상 동일한 **dummy** 값이 들어갑니다.

2. HashSet 선언 및 기본 사용법

HashSet을 사용하려면 **java.util.HashSet** 패키지를 import해야 합니다.

java

복사편집

```
import java.util.HashSet;  
  
import java.util.Set;  
  
  
public class HashSetExample {  
  
    public static void main(String[] args) {  
  
        // HashSet 선언  
  
        Set<String> set = new HashSet<>();
```

```

    // 요소 추가

    set.add("Apple");

    set.add("Banana");

    set.add("Cherry");

    set.add("Apple"); // 중복 요소 추가 (무시됨)

    System.out.println(set); // 출력 결과: [Banana, Cherry,
Apple] (순서는 다를 수 있음)

}

```

중복된 Apple은 저장되지 않음!

set은 순서를 보장하지 않으므로 출력할 때 순서가 달라질 수 있음!

3. HashSet의 내부 동작 원리

❖ HashSet은 어떻게 중복을 제거할까?

- 1 객체를 추가할 때 hashCode()와 equals()를 사용하여 중복 여부를 판단
- 2 같은 hashCode() 값을 가지더라도 equals() 메서드가 true이면 중복으로 간주하고 추가하지 않음
- 3 내부적으로 HashMap<K, V>를 사용하여 데이터를 저장함

- key → HashSet의 요소
- value → 고정된 dummy 값 (PRESENT 사용)

◆ HashSet에 데이터를 저장하면 내부적으로 put(key, dummyValue) 방식으로 저장됩니다.

```
private static final Object PRESENT = new Object();
```

```
private transient HashMap<E, Object> map;
```

즉, **HashSet**은 **HashMap**을 활용한 일종의 **key** 집합입니다.

4. **HashSet** 주요 메서드

HashSet은 **Set** 인터페이스를 구현하며, 다양한 메서드를 제공합니다.

메서드	설명
-----	----

<code>add(E e)</code>	요소 추가 (중복 요소는 무시됨)
-----------------------	--------------------

<code>remove(Object o)</code>	특정 요소 제거
-------------------------------	----------

<code>contains(Object o)</code>	특정 요소가 있는지 확인 (<code>true</code> / <code>false</code>)
---------------------------------	--

<code>size()</code>	저장된 요소 개수 반환
---------------------	--------------

<code>isEmpty()</code>	비어있는지 확인
------------------------	----------

<code>clear()</code>	모든 요소 제거
----------------------	----------

<code>iterator()</code>	Iterator 를 반환하여 요소를 순회 가능
-------------------------	----------------------------------

예제

```
java
```

복사편집

```
import java.util.HashSet;
```

```
public class HashSetMethods {  
    public static void main(String[] args) {  
        HashSet<Integer> set = new HashSet<>();  
  
        set.add(10);  
        set.add(20);  
        set.add(30);  
  
        System.out.println("HashSet: " + set);  
        System.out.println("20이 포함되어 있나요? " +  
set.contains(20));  
  
        set.remove(20);  
        System.out.println("20 제거 후: " + set);  
  
        System.out.println("현재 크기: " + set.size());  
  
        set.clear();  
        System.out.println("모든 요소 삭제 후: " + set.isEmpty());  
    }  
}
```

출력 결과:

HashSet: [20, 10, 30]

20이 포함되어 있나요? true

20 제거 후: [10, 30]

현재 크기: 2

모든 요소 삭제 후: true

5. HashSet의 반복(Iteration)

- ◆ HashSet은 순서를 유지하지 않기 때문에 순회할 때마다 출력 순서가 다를 수 있습니다.

1) for-each 문 사용

java

복사편집

```
for (String item : set) {  
    System.out.println(item);  
}
```

2) Iterator 사용

java

복사편집

```
import java.util.HashSet;  
  
import java.util.Iterator;  
  
  
public class HashSetIterator {  
    public static void main(String[] args) {
```

6. HashSet과 TreeSet, LinkedHashSet 비교

컬렉션 특징

HashSet 종복 불가, 순서 없음, 빠른
 검색($O(1)$)

**LinkedHash
Set** 입력 순서 유지

TreeSet

```
import java.util.HashSet;  
  
import java.util.LinkedHashSet;
```

```
import java.util.TreeSet;

public class SetComparison {
    public static void main(String[] args) {
        HashSet<Integer> hashSet = new HashSet<>();
        LinkedHashSet<Integer> linkedHashSet = new
        LinkedHashSet<>();
        TreeSet<Integer> treeSet = new TreeSet<>();

        int[] numbers = {30, 10, 20, 50, 40};

        for (int num : numbers) {
            hashSet.add(num);
            linkedHashSet.add(num);
            treeSet.add(num);
        }

        System.out.println("HashSet: " + hashSet); // 순서 없음
        System.out.println("LinkedHashSet: " + linkedHashSet); //
        입력 순서 유지
        System.out.println("TreeSet: " + treeSet); // 오름차순 정렬
    }
}
```

출력 결과 (순서는 다를 수 있음)

HashSet: [50, 20, 40, 10, 30]

LinkedHashSet: [30, 10, 20, 50, 40]

TreeSet: [10, 20, 30, 40, 50]

7. HashSet의 성능 (Big-O)

연산	평균 시간	최악의 경우 (해시 충돌 발생)
----	-------	----------------------

add() O(1) O(n)

remove(O(1)
)

contains O(1) O(n)
s()

☞ 일반적으로 O(1)의 성능을 보장하지만, 해시 충돌이 많을 경우 O(n)까지 느려질 수 있음!

8. HashSet을 활용한 중복 제거

- ◆ 리스트에서 중복 요소를 제거할 때 유용하게 사용할 수 있습니다.

```
import java.util.*;
```

```
public class RemoveDuplicates {
```

```
public static void main(String[] args) {  
  
    List<String> list = Arrays.asList("Apple", "Banana",  
"Apple", "Orange", "Banana");  
  
    Set<String> uniqueSet = new HashSet<>(list);  
  
    System.out.println("중복 제거 결과: " + uniqueSet);  
  
}  
  
}
```

출력 결과:

중복 제거 결과: [Banana, Orange, Apple]

9. 결론

- HashSet은 중복 없는 데이터 저장을 위해 사용
 - 해시 테이블을 이용해 빠른 검색 속도($O(1)$) 제공
 - 순서를 보장하지 않음
 - equals()와 hashCode()를 재정의하여 사용자 정의 객체의 중복을 처리 가능
- ☞ 중복을 허용하지 않으면서 빠른 데이터 처리를 원한다면 HashSet을 사용하세요!
- 

> 07. 이중 for문 사용하기

반복문안에 반복문을 넣을 수 있다. for문 안에 for문을 중복해서 사용 할 수 있다.

012345678901234567890123456789012345678901234567890123456789...

0123456789를 출력하는 이작업을 10번 반복하여 출력하고 싶으면 다음과 같이 기술 하면 될 것이다. for문 을 이용해서 0123456789를 10번 출력하는 코드로 다음과 같이 기술하면 될 것이다.

```
for(int i=0;i<10;i++) {  
    System.out.print("0123456789");  
}
```

잘 생각해 보면 for문 사이에 있는
프린트 문을 for문을 이용해서 0부터
9까지 출력해서 화면에 0123456789가
출력 되도록 할 수 있다.

print문으로 0123456789를 출력하는

대신에 다음과 같은 for문을 이용하여 같은 결과를 만들 수 있을 것이다.

```
for(int j=0;j<10;j++) {  
    System.out.print(j); // 출력결과 0123456789  
}
```

이 2가지 결과를 합친 결과는 다음과 같다.

```
for(int i=0;i<10;i++) {  
    for(int j=0;j<10;j++){  
        System.out.print(j);  
    }  
}
```

이중for 문을 사용할 때 조심해야 할
점은 초기식으로 선언되는 i,j 같은
변수들의 이름이 겹치거나 잘못 사용하면
문제가 발생 한다는 점이다.

바깥쪽 for문의 반복 코드 부분이 새로
시작하게 되면 내부에 선언한 변수와
for문의 초기문도 새로 실행되어 변수의

값들이 초기화 되어 이전에 가지고 있던 값이 사라진다. 코드에서 j 값은 i가 0부터
변환식에 도착해서 하나씩 증가 할때 마다 0으로 초기화 되어 다시 하나씩 증가한다.

변수는 지역 변수와 전역변수로 구성되어 있는데 지역변수는 특정 지역에서만 사용할 수
있는 변수이고 전역변수는 모든 지역에서 사용할 수 있는 변수이다. 지금 까지 사용한
변수는 모두 지역 변수이고 전역 변수 앞에는 static이 붙는다. 나중에 좀더 자세히 살펴볼
예정이고 일단 지역변수의 특성만 좀 기억해 보자.

지역변수는 기본적으로 중괄호 블럭안에 선언한 변수로 선언되면 해당 중괄호 블럭이나
해당 중괄호 블럭 안쪽의 중괄호 블럭들에서만 사용할 수 있다.

지역변수가 선언된 범위에서 다시 같은 이름의 지역 변수를 선언하면 동일한 지역 변수를
식별 할 수 없어서 문제가 발생 하니 주의하자.

메소드의 매개변수는 해당 메소드 중괄호 블록 안에서 지역변수로 사용된다.

for문의 초기화 식은 해당 for문 중괄호 블록 안에서 지역변수로 사용된다.

```
① int l1=1;  
{  
    ② //이 위치에서는 l1값만 접근할수있다.  
    int l2=2;  
    //l1,l2값을 접근 할 수 있다.  
    for(int j=1;j<3;j++) {  
        ③ //l1,l2,j값을 접근 할 수 있다.  
        for(int i=1;i<3;i++) {  
            ④ //l1,l2,j,i값을 접근 할 수 있다.  
        }  
        //l1,l2,j값을 접근 할 수 있다.  
    }  
    //l1,l2값을 접근 할 수 있다.  
}  
//이 위치에서는 l1값만 접근할수있다.
```

왼쪽 이미지의 1번 부분은 메인
메소드 선언부에 중괄호안
이여서 l1은 main 메소드에
선언된 지역변수이다.

이 중괄호 부분에서 선언된
11은 중괄호 내부인 왼쪽
이미지에서 표시된 1, 2, 3, 4
중괄호 안쪽 부분에서 사용할 수
있다.

2번 부분에 선언된 12의 경우
1번 부분은 2번 중괄호 부분의
바깥쪽 중괄호 이므로 1번을
제외한 2,3,4 중괄호 부분에서
사용할 수 있다.

3번에 선언된 변수 j같은 경우 3번 중괄호 블록에서 사용하려고 선언한 변수이므로
3,4에서만 사용할 수 있다.

4번 블록에서 선언된 i값은 4번 블록에서만 사용할 수 있다.

상위 모든 주석 부분에 l1,l2,i,j 변수를 찍을수 있는지 코드를 통해서 직접 확인해보자.

다음과 같이 아래로 출력하는 방법에 대해서 생각해 보자.

0123456789 //01234567890123456789012345... 옆으로 출력되는 것과

0123456789 //아래로 출력되는 것의 차이점을 생각해보자.

0123456789 //옆으로 출력되는 것은 엔터가 없는것이고

0123456789 //아래로 출력되는 것은 엔터가 있는것이여서

0123456789 // '0123456789'가 반복되는 것이아니라.

0123456789 // '0123456789엔터'가 반복되는 것이다.

```
for(int i=0;i<10;i++) {  
    //System.out.print(i+"\n");  
    System.out.print(i);  
}  
System.out.print("\n");
```

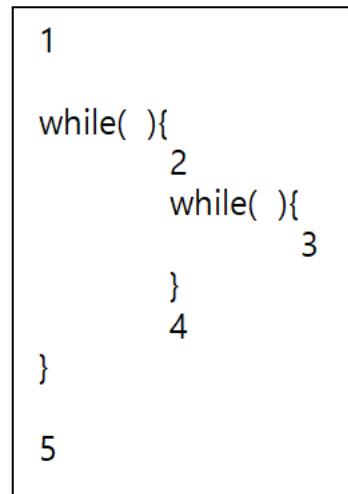
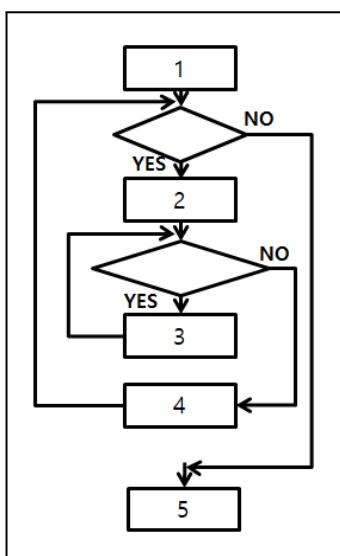
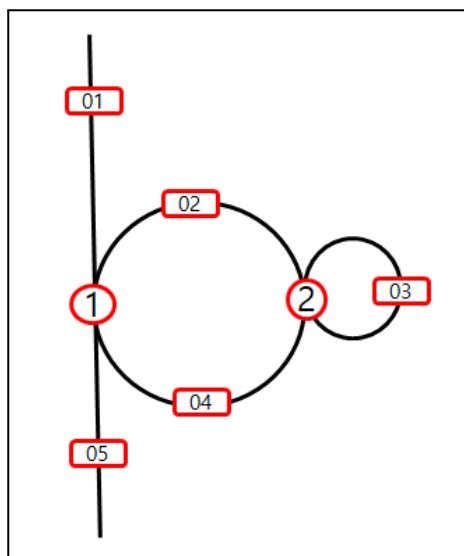
하나의 for문으로 “0123456789엔터”를
출력하는 프로그램을 구현하려면 왼쪽과
같이 하면 된다. 간혹 주석 부분처럼
잘못 구현하는 경우가 있는데 잘 생각해
보면 주석 한 부분은 다음과 같이 잘못
출력 되는 걸 알 수 있다.

“0엔터1엔터2엔터3엔터4엔터5엔터6엔터7엔터8엔터9엔터엔터”

올바르게 구현하려면 상위 코드처럼을 이용해서 “0123456789엔터”가 여섯번 반복되도록 코드를 구현하면 우리가 원하는 출력 결과를 얻을 수 있다.

```
for(int j=0;j<10;j++) {  
    for(int i=0;i<10;i++) {  
        //System.out.print(i+"\n");  
        System.out.print(i);  
  
    }  
    System.out.print("\n");  
}
```

“0123456789엔터”이 아래로 열개 출력되도록 구현해 보자. 다음과 같이 구현 가능할 것이다.



상위 맨왼쪽 이미지를 보고 순서도와 의사 코드를 작성 하시오.

상위 순서도에서 출력 되는 모든 경우를 기술해 보자. 반복되는 부분을 소괄호로 묶었다.

1번 부부의 반복문이 거짓인 경우 출력 결과는 01, 05가 된다.

1번 부분이 반복문이 참이고 2 번 부분이 반복무늬 거짓이면 출력 결과는 01, 04, 02, 05가된다.

1번 부분이 반복문이 참이고 2 번 부분이 반복무늬 거짓인 상태에서 1번 부분이 여러번 반복되면 출력 결과는 01, (04, 02), 04, 02, 04, 02, ..., 05가 된다.

1번 부분이 반복문이 참이고 2 번 부분이 참 일때 출력 결과는 01, 04, 03, 02, 05가 된다.

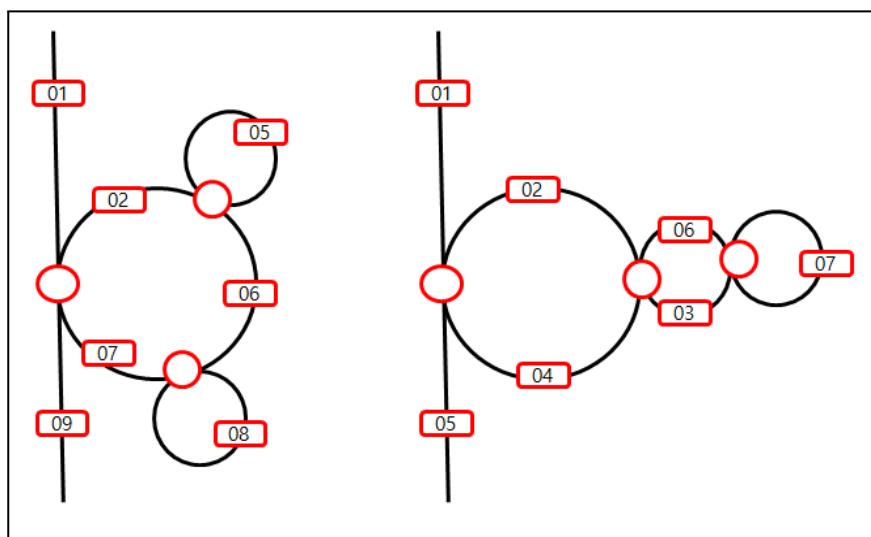
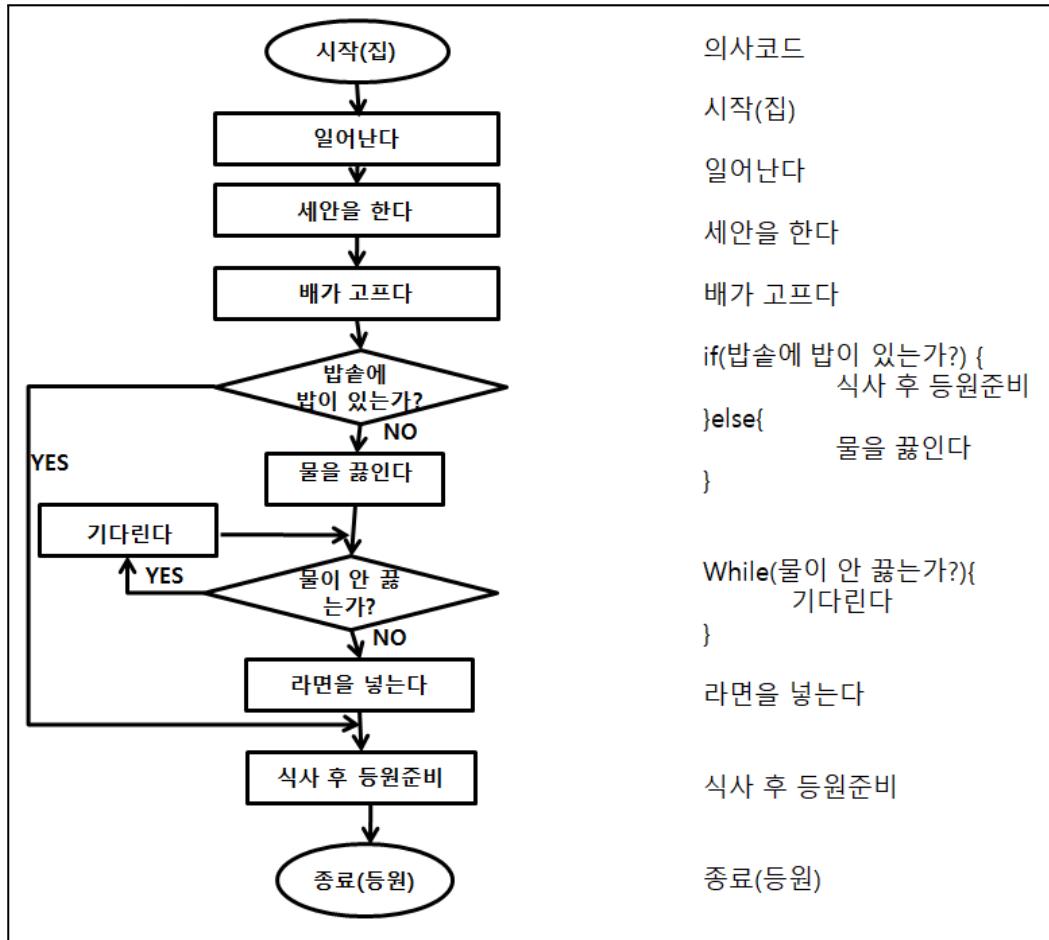
1번 부분이 반복문이 참이고 2 번 부분이 참 일때 반복문 1번은 1번 반복문 2번은 여러번 반복된다면 출력 결과는 01, 04, (03), 03, 03, ..., 02, 05가 된다.

1번 부분이 반복문이 참이고 2 번 부분이 참인 일때 반복문 1번은 여러번 반복문 2번은 한번 반복된다면 출력 결과는 01, (04, 03, 02), 04, 03, 02, 04, 03, 02, ..., 05, 가

된다.

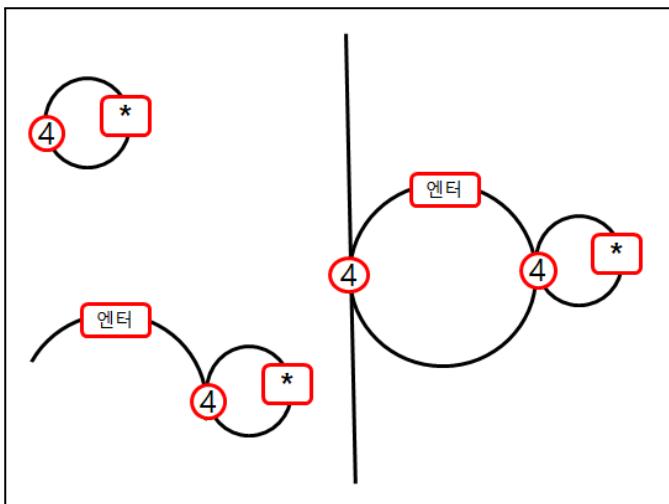
1번 부분이 반복문이 참이고 2 번 부분이 참인 일때 반복문 1번은 여러번 반복문 2번도 여러번 반복된다면 출력 결과는 01, (04, (03), 03, 03, ... 02), 04, 03, 02, 04, 03, 02, ..., 05가 된다.

문제1) 아래의 순서도 의사코드에서 뭐가 잘못되었는지 확인해서 올바르게 고쳐 보자.



문제 2) 왼쪽 기찻길 이미지 두개를 순서도와 의사 코드로 구현하여 보자.

다음을 생각해 보자. 별을 4번찍다가 엔터를 한번치고 별을 4번찍다가 엔터를 한번치고 이 작업을 4번 반복하면 ‘ ****엔터 ****엔터 ****엔터 ****엔터 ‘와 같이 출력이 될 것이고 실제로 엔터가 줄을 바꾼다면 왼쪽 처럼 직사각형 모양이 될 것이다. 이것을 기찻길로 만든다고 생각해보자. ****은 별한개 출력하는 작업을 4번 반복하는것과 같다. 이는 아래 왼쪽 상단의 기찻길 모양과 같이 될것이다. 그 다음에 엔터를 쳐야 할 것이다.

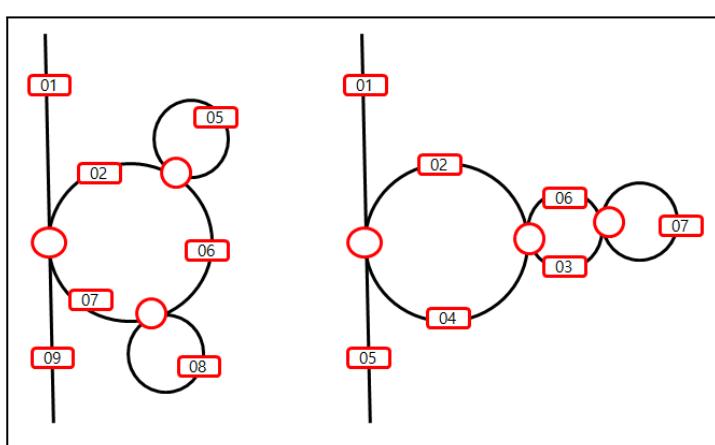


이것은 왼쪽 아래 이미지처럼 별 4 번 찍은 것에 엔터 한번 친 것이 될 것이다. 별 4번 찍고 엔터 1 번 치는 것을 총네 번 반복해야 된다. 빨간색 동그라미 안에 숫자가 반복 횟수 라면 최종 결과가 왼쪽 이미지 처럼 될 것이고 이것은 사각형 모양을 화면에 출력하는 기찻길이 될 것이다. 왼쪽 이미지 기찻길을 순서도와 의사 코드로 만들어 보자.

문제 1) 아래처럼 출력될 수 있도록 기찻길을 만들고 순서도를 만든 다음 의사 코드를 작성해보자.

1. 1*****1*****1*****1*****1
2. *****1*****1*****1*****1
3. 2*****2*****2*****2*****
4. 21*****21*****21*****21*****1
5. 1****21****21****21****2
6. 1111****21111****21111****21111****2
7. 1111****22221111****22221111****2222
8. 111122223333444411112222333344441111222233334444

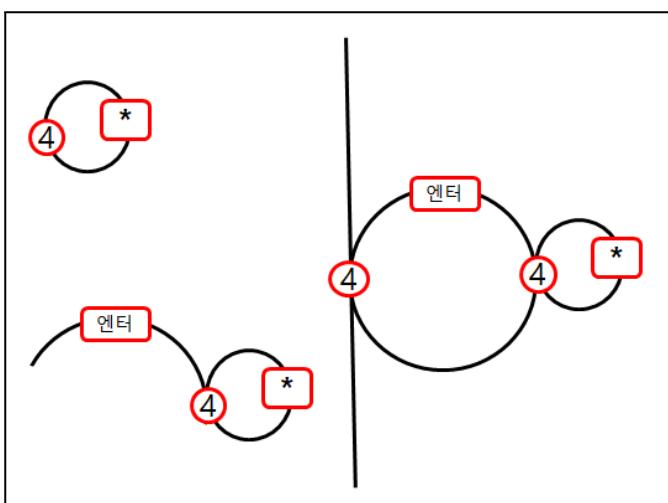
문제 1) 왼쪽 기찻길 이미지 두개를 순서도와 의사 코드로 구현하여 보자.



이중 반복문을 이용해서 아래 이미지 처럼 정사각형 별을 찍는 프로그램을 작성하는 방법을 생각해보자.



다음을 생각해보자. 별을 4번찍다가 엔터를 한번치고 별을 4번찍다가 엔터를 한번치고 이 작업을 4번 반복하면 ‘ ****엔터 ****엔터 ****엔터 ****엔터 ‘와 같이 출력이 될 것이고 실제로 엔터가 줄을 바꾼다면 왼쪽 처럼 직사각형 모양이 될 것이다. 이것을 기찻길로 만든다고 생각해보자. ****은 별한개 출력하는 작업을 4번 반복하는것과 같다. 이는 아래 왼쪽 상단의 기찻길 모양과 같이 될것이다. 그 다음에 엔터를 쳐야 할 것이다.



이것은 왼쪽 아래 이미지처럼 별 4 번
찍은 것에 엔터 한번 친 것이 될 것이다.
별 4번 찍고 엔터 1 번 치는 것을 총네 번
반복해야 된다. 빨간색 동그라미 안에
숫자가 반복 횟수 라면 최종 결과가 왼쪽
이미지 처럼 될 것이고 이것은 사각형
모양을 화면에 출력하는 기찻길이 될
것이다. 코드로 구현해 보자.

별을 한개 찍는다. `System.out.print("*");`

상위 왼쪽상단 이미지 처럼 별을 4번 찍는다.

```
for(int i=0;i<4;i++) {  
    System.out.print("*");  
}
```

상위 왼쪽하단 이미지처럼 별을 4번 찍고 엔터를 한번 친다.

```
for(int i=0;i<4;i++) {  
    System.out.print("*");  
}  
System.out.println();
```

상위 오른쪽 이미지 처럼 별을 4번 찍고 엔터를 한번 치는 작업을 4번 반복한다.

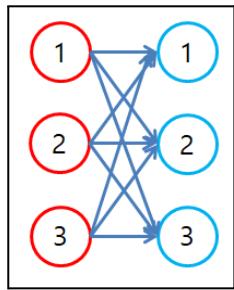
```
for(int j=0;j<4;j++) {  
    for(int i=0;i<4;i++) {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```

문제 1) 아래처럼 출력될 수 있도록 기찻길을 만들고 순서도를 만든 다음 코드를 작성해보자. 구현이 어려우면 기찻길을 그려보고 하자. 그래도 어려우면 순서도를 그려보고

하자.

1. 1*****1*****1
2. *****1*****1*****1
3. 2*****2*****2*****2*****
4. 21*****1*****1*****1
5. 1****21****21****21****2
6. 1111****21111****21111****21111****2
7. 1111****22221111****22221111****2222
8. 111122223333444411112222333344441111222233334444

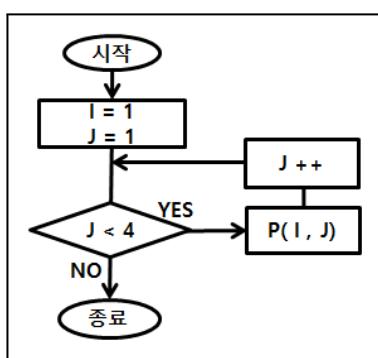
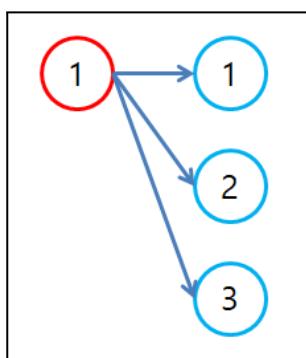
빨간공과 파란공을 모두 짹지어 찍는 프로그램을 구현해 보자.



빨간공 1, 2, 3 3개와 파란공 1, 2, 3 3개를 모두 짹지어 보자. 모든 결과를 출력해 보면 다음과 같은 결과를 가진다. (1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)

다음 3가지를 확인해 보자. 빨간색 1번 공을 가지고 파란공들과 짹을 지어 보면 (1,1), (1,2), (1,3) 과 같다. 빨간색 2번 공을 가지고 파란공들과 짹을 지어 보면 (2,1), (2,2), (2,3) 과 같다. 빨간색 3번 공을 가지고 파란공들과 짹을 지어 보면 (3,1), (3,2), (3,3)과 같다.

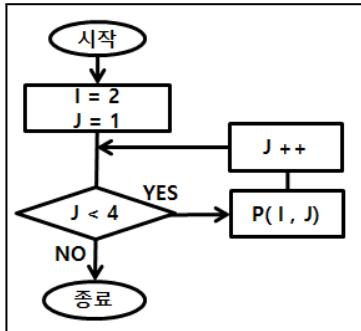
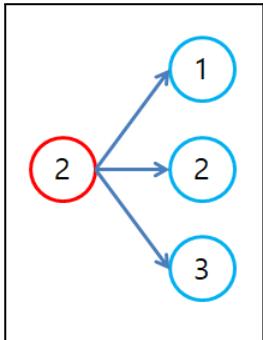
이 3가지를 하나씩 나눠서 순서도로 만들어 보면 다음과 같고 결국 3가지 결과를 합치면 빨간색 공과 파란색 공의 모든 짹이 된다.



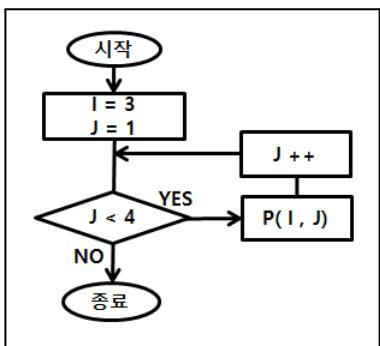
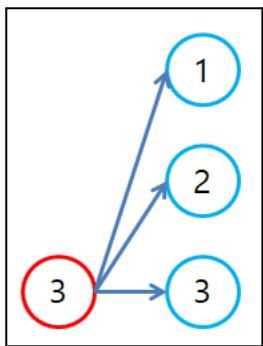
맨 왼쪽 이미지처럼 빨간색 공 1개는 파란색공 3개와 짹을 지어보면 다음과 같이 될 것이다. 앞이 빨간공 뒤가 파란공이라면 (1,1), (1,2), (1,3) 와 같은 짹이 만들어 질 것이다. 자세히 보면 빨간 공은 1인 상태로 고정하고 파란 공을 1부터 3까지 하나씩 증가 하면서 짹을 지었다. 빨간색 공을 i 라 하고 파란색

공을 j 라 한다면 반복문을 이용해서 j 값을 하나씩 증가시 키는 방법은 이미 공부 하여서 충분히 생각해 볼 수 있을 것이다. 고정된 값 i 는 반복문 밖에 선언하고 반복문 안에서는 j 값을 1부터 하나씩 증가 시키며 i 와 j 값을 출력하면 원하는 결과를 얻을 수 있을 것이다. 상위 순서도를 확인하고 기찻길과 의사코드를 만들어 보자. 순서도를 잘 따라 가며

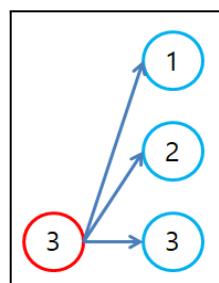
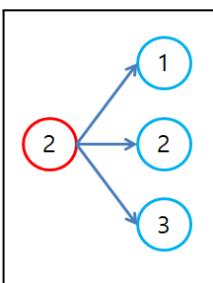
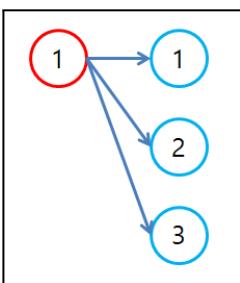
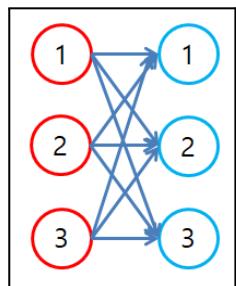
출력값을 확인해 보자.



왼쪽 첫 번째 이미지 처럼 빨간 공이 2라면 $(2, 1), (2, 2), (2, 3)$ 와 같은 짝이 만들어 질 것이다. 이전 상황과 잘 비교해 보면 i 값이 2로 바뀐 것 말고는 변한게 없다는 것을 알 수 있다.

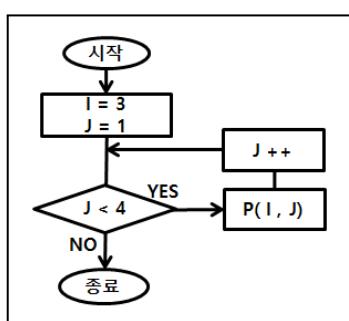
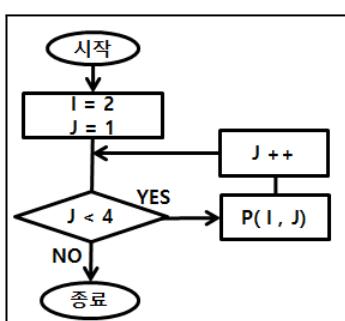
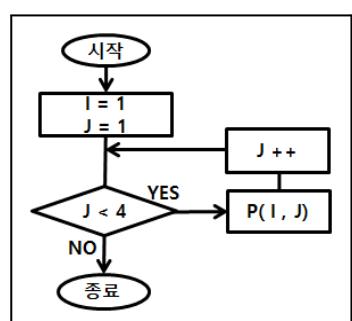


왼쪽 첫 번째 이미지 처럼 빨간 공이 3이라면 $(3, 1), (3, 2), (3, 3)$ 와 같은 짝이 만들어 질 것이다. 이전 상황과 잘 비교해 보면 i 값이 3로 바뀐 것 말고는 변한게 없다는 것을 알 수 있다.



왼쪽 3개의 이미지를 합치면 맨 왼쪽 이미지 처럼 된다. 짹을 짓는다면 다음과 같은 결과가 나올 것이다. $(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)$, 상위와 같이 출력 되려면 어떻게 순서도와 코드를 구현 할 것인지 생각해 보자.

잘 살펴보면 맨 왼쪽 이미지는 나머지 세 개 이미지를 합친 것과 같다. 따라서 오른쪽 세 개 이미지를 순서대로 그리고 순서대로 실행 한다면 같은 결과가 나올 것이다.



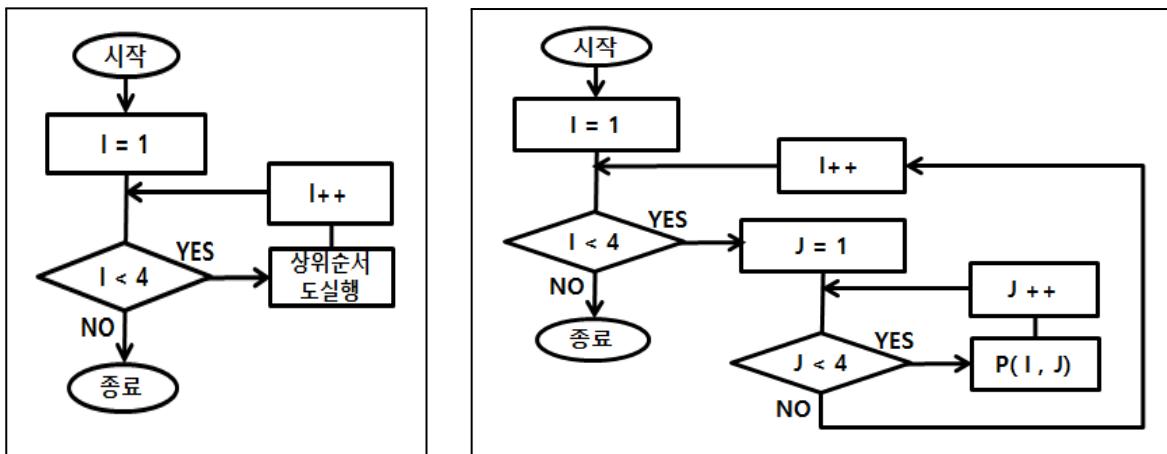
출력 결과를 잘 살펴보면 상위 첫 번째 순서도 출력결과는 $(1, 1), (1, 2), (1, 3)$ 이다. 두 번째 순서도 출력결과는 $(2, 1), (2, 2), (2, 3)$ 이다. 세 번째 순서도 출력결과는 $(3, 1), (3, 2), (3, 3)$ 이다. 세 가지 순서도를 원쪽부터 순서대로 실행한다면 우리가 원하는 결과를 얻을 수 있고, 결국 $(i, 1), (i, 2), (i, 3)$ 과 같음을 알 수 있다.

```
int i=1;
for(int j=1;j<4;j++) {
    System.out.println(i+":"+j);
}
```

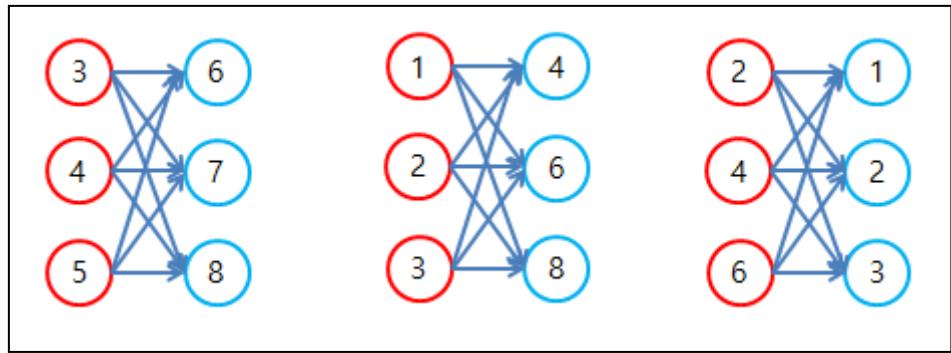
다음을 자바코드로 변경하면 다음과 같다. i 가 무엇이냐에 따라서 빨간색 공의 번호가 결정되는 것과 같다.

잘 생각해 보면 세 개의 순서도가 상당히 유사하다는 것을 알 수 있고 유일한 차이점을 찾는다면 첫 번째 순서도는 i 는 1이고 두 번째 순서도는 i 가 2이고 세 번째 순서도는 i 가 3이라는 것이다. i 값이 하나씩 1, 2, 3 증가 하고 있다는 것만 빼고는 달라진 것이 하나도 없다. 결국에 i 값이 하나씩 증가하면서 반복 되고 있다는 것이다. 숫자 하나씩 증가되면서 반복되는 것은 반복문으로 바꿀수 있다는 사실은 무수히 경험해 왔을 것이다. 반복문으로 만들어 보면 i 값이 증가하는 형태의 반복문안에 상위 순서도 1개가 들어가 있으면 될 것이다.

상위 왼쪽 순서도를 확인해 보면 i 가 1부터 3까지 3번 반복하면서 상위 순서도가 3번 실행된다. i 가 1일때 i 가 1인 상태에서 상위순서도가 실행이 되고 i 가 2일때 i 가 2인 상태에서 상위순서도가 실행이 되고 i 가 3일때 i 가 3인 상태에서 상위순서도가 실행이 되면 우리가 원하던 $(1, 1), (1, 2), (1, 3)$ $(2, 1), (2, 2), (2, 3)$ $(3, 1), (3, 2), (3, 3)$ 결과를 얻을 수 있다. 다음 이미지는 상위 3개의 이미지를 합쳐서 하나의 순서도로 만들었다. i 가 1씩 증가하면서 상위 순서도를 하나씩 반복되는 형태이다. 천천히 확인해 보자.



오른쪽 순서도는 왼쪽 순서도에서 상위 순서도 실행 부분을 실제로 붙여서 만든 순서도로 하나하나 따라 가며 출력값을 확인해 보면 원하는 결과가 출력 된다는 것을 알 수 있다. 다음에 코드가 있는데 보지말고 for문과 while문으로 만들어 보자. 반복문으로 바꾸어 보자.



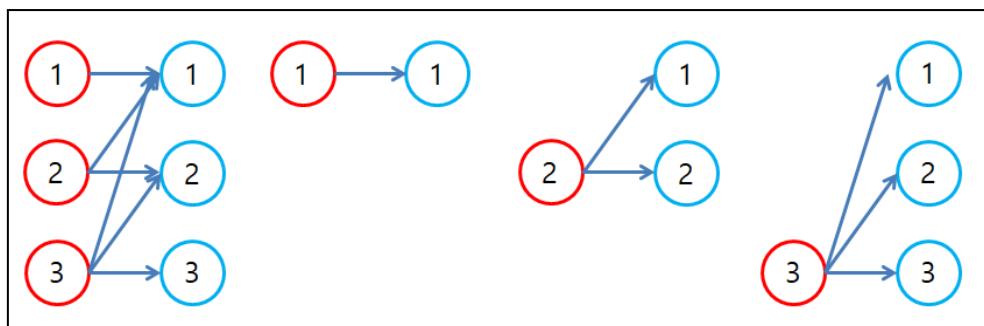
문제 1) 상위 3개의 이미지를 보고 어떤 결과가 출력되는지 기술 해 보고, 기찻길을 그려보고, 순서도를 그려보고, 의사 코드를 만들어 보자.

$1 \cdot 1 = 1$	$1 \cdot 2 = 2$	$1 \cdot 3 = 3$	$1 \cdot 4 = 4$	$1 \cdot 5 = 5$	$1 \cdot 6 = 6$	$1 \cdot 7 = 7$	$1 \cdot 8 = 8$	$1 \cdot 9 = 9$
$2 \cdot 1 = 2$	$2 \cdot 2 = 4$	$2 \cdot 3 = 6$	$2 \cdot 4 = 8$	$2 \cdot 5 = 10$	$2 \cdot 6 = 12$	$2 \cdot 7 = 14$	$2 \cdot 8 = 16$	$2 \cdot 9 = 18$
$3 \cdot 1 = 3$	$3 \cdot 2 = 6$	$3 \cdot 3 = 9$	$3 \cdot 4 = 12$	$3 \cdot 5 = 15$	$3 \cdot 6 = 18$	$3 \cdot 7 = 21$	$3 \cdot 8 = 24$	$3 \cdot 9 = 27$
$4 \cdot 1 = 4$	$4 \cdot 2 = 8$	$4 \cdot 3 = 12$	$4 \cdot 4 = 16$	$4 \cdot 5 = 20$	$4 \cdot 6 = 24$	$4 \cdot 7 = 28$	$4 \cdot 8 = 32$	$4 \cdot 9 = 36$
$5 \cdot 1 = 5$	$5 \cdot 2 = 10$	$5 \cdot 3 = 15$	$5 \cdot 4 = 20$	$5 \cdot 5 = 25$	$5 \cdot 6 = 30$	$5 \cdot 7 = 35$	$5 \cdot 8 = 40$	$5 \cdot 9 = 45$
$6 \cdot 1 = 6$	$6 \cdot 2 = 12$	$6 \cdot 3 = 18$	$6 \cdot 4 = 24$	$6 \cdot 5 = 30$	$6 \cdot 6 = 36$	$6 \cdot 7 = 42$	$6 \cdot 8 = 48$	$6 \cdot 9 = 54$
$7 \cdot 1 = 7$	$7 \cdot 2 = 14$	$7 \cdot 3 = 21$	$7 \cdot 4 = 28$	$7 \cdot 5 = 35$	$7 \cdot 6 = 42$	$7 \cdot 7 = 49$	$7 \cdot 8 = 56$	$7 \cdot 9 = 63$
$8 \cdot 1 = 8$	$8 \cdot 2 = 16$	$8 \cdot 3 = 24$	$8 \cdot 4 = 32$	$8 \cdot 5 = 40$	$8 \cdot 6 = 48$	$8 \cdot 7 = 56$	$8 \cdot 8 = 64$	$8 \cdot 9 = 72$
$9 \cdot 1 = 9$	$9 \cdot 2 = 18$	$9 \cdot 3 = 27$	$9 \cdot 4 = 36$	$9 \cdot 5 = 45$	$9 \cdot 6 = 54$	$9 \cdot 7 = 63$	$9 \cdot 8 = 72$	$9 \cdot 9 = 81$

1단	1*1=1	2*1=2	3*1=3
1*1=1	1*2=2	2*2=4	3*2=6
1*2=2	1*3=3	2*3=6	3*3=9
1*3=3	1*4=4	2*4=8	3*4=12
1*4=4	1*5=5	2*5=10	3*5=15
1*5=5	1*6=6	2*6=12	3*6=18
1*6=6	1*7=7	2*7=14	3*7=21
1*7=7	1*8=8	2*8=16	3*8=24
1*8=8	1*9=9	2*9=18	3*9=27
1*9=9			
2단	4*1=4	5*1=5	6*1=6
2*1=2	4*2=8	5*2=10	6*2=12
2*2=4	4*3=12	5*3=15	6*3=18
2*3=6	4*4=16	5*4=20	6*4=24
2*4=8	4*5=20	5*5=25	6*5=30
2*5=10	4*6=24	5*6=30	6*6=36
2*6=12	4*7=28	5*7=35	6*7=42
2*7=14	4*8=32	5*8=40	6*8=48
2*8=16	4*9=36	5*9=45	6*9=54
2*9=18			
3단	7*1=7	8*1=8	9*1=9
3*1=3	7*2=14	8*2=16	9*2=18
3*2=6	7*3=21	8*3=24	9*3=27
3*3=9	7*4=28	8*4=32	9*4=36
3*4=12	7*5=35	8*5=40	9*5=45
3*5=15	7*6=42	8*6=48	9*6=54
3*6=18	7*7=49	8*7=56	9*7=63
3*7=21	7*8=56	8*8=64	9*8=72
3*8=24	7*9=63	8*9=72	9*9=81

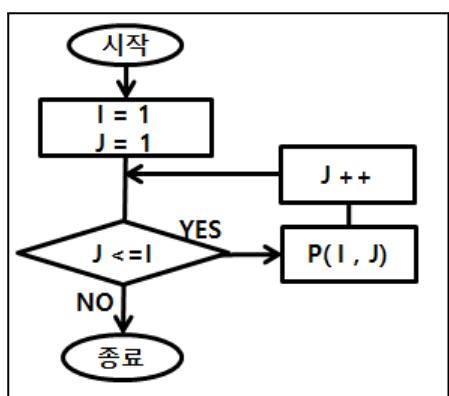
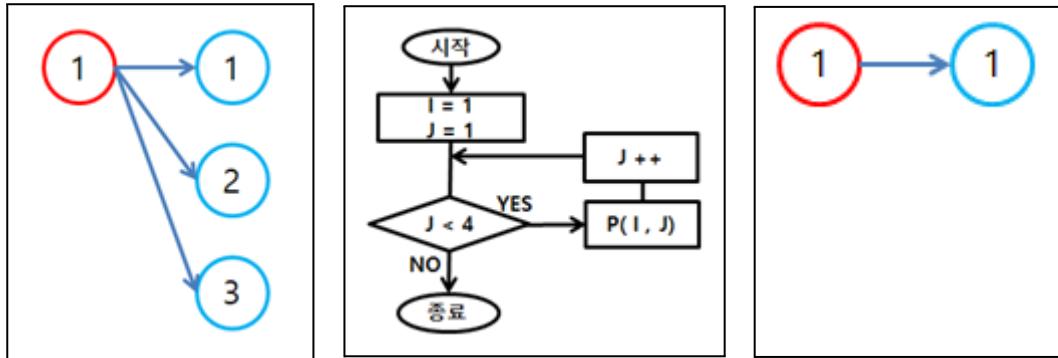
문제 2) 이미지와 같은 형태의 구구단을 출력 할 수 있는 코드를 만들어 보자.

다음은 빨간색 i값이 파란색 j값에 영향을 주는 경우이다.

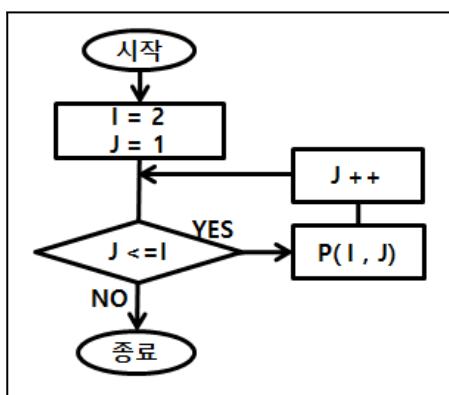
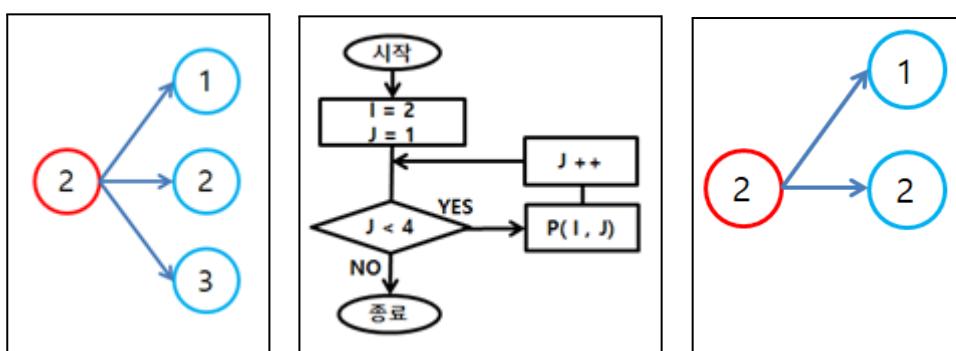


상위를 실행시키면 다음과 같은 결과가 나올 것이다. (1, 1) (2, 1), (2, 2) (3, 1), (3, 2), (3, 3) 빨간색 공을 i라 하고 파란색 공을 j라 한다면 i값은 j값을 찍는데 영향을 준다. i값이 1일때 j값은 1 한개 i값이 2일때는 j값 1, 2 두 개 i값이 3일 때는 j값은 1, 2, 3 세개가 출력 된다. 잘 생각해 보면 파란공 j가 빨간공 i보다 같거나 작을때만 실행이

되는 것을 확인할 수 있다. i 값에 따라 출력되는 결과가 규칙성 있게 달라지고 있고, 이렇게 i 값이 j 값 반복에 영향을 주려면 어떻게 해야 하는지 생각해 보자. i 값이 j 값 반복에 영향을 주려면 j 값 반복문 안의 조건식으로 i 값을 사용해야 한다.

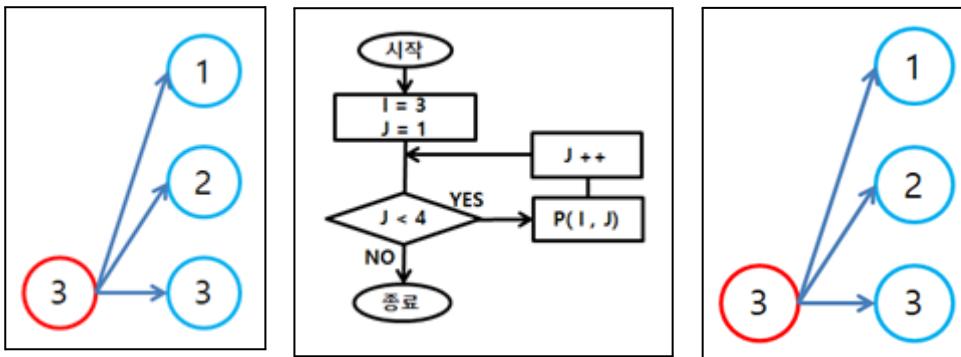


상위 첫번째 이미지와 두번째 이미지에서 반복문 밖의 i 값이 반복문의 안쪽에서 출력 할때 영향을 주었다. 두번째 순서도 구조를 유지하면서 세번째 이미지처럼 (1,1) 만 출력되도록 하려면 i 값을 가지고 반복문의 조건식에 영향을 주어야 한다. 무조건 3번 반복되는 상태에서 i 값이 j 값보다 같거나 작을때만 반복 되도록 해야 한다. 그래서 조건식을 왼쪽 처럼 $J \leq i$ 과 같이 바꾸거나 $J < i+1$ 로 바꾸어 주면 된다. 이렇게 하면 조건식을 첫번째 돌때 j 값이 1이고 i 값은 1이므로 $1 \leq 1$ 비교가 true여서 반복 부분을 반복하고 두번째 돌때 j 값이 2이고 i 값은 여전히 1이여서 $2 \leq 1$ 비교가 false여서 반복문을 빠져나와 반복문이 종료 된다. 출력 결과는 (1, 1) 이다.



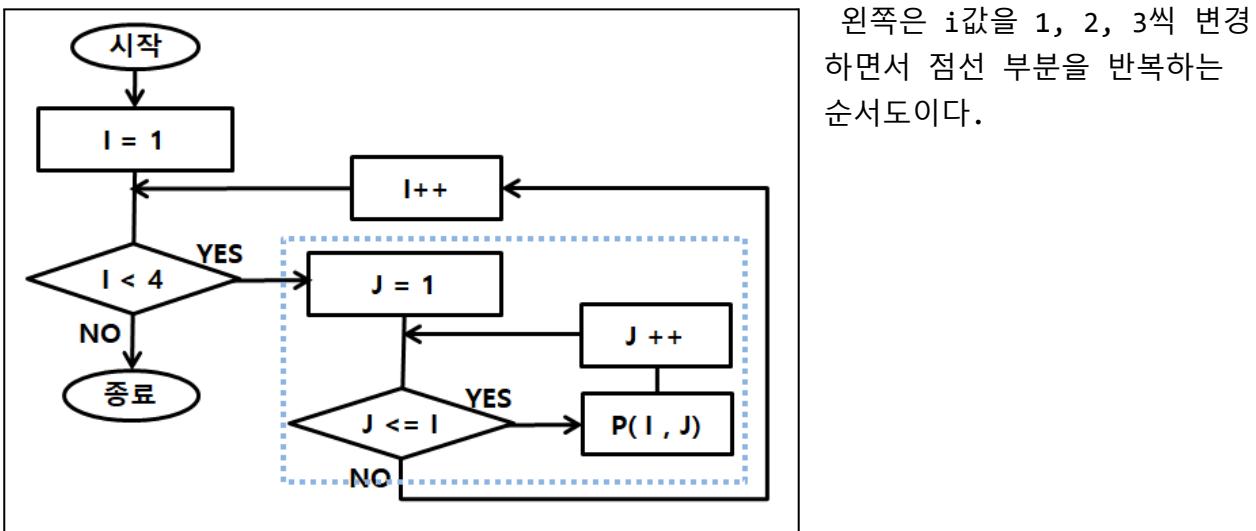
상위 첫번째 이미지와 두번째 이미지에서 반복문 밖의 i 값이 반복문의 안쪽에서 출력 할때 영향을 주었다. 두번째 이미지 순서도 구조를 유지하면서 세번째 형태로 만들려면 i 값을 가지고 반복문의 조건식에 영향을 주어야 한다. 무조건 3번 반복한 것을 i 값에 따라서 반복 횟수를 줄여야 한다. 그래서 조건식을 왼쪽 처럼 $J < i+1$ 나 $J < i$ 로 바꾸어 주면 이전 빨간공이 1일때와 동일한 구조를 가지며 원하는 결과를 출력 할 수 있게 된다. 이렇게 하면 조건식을 첫번째 돌때 j 값이 1이고 i 값은 2이므로 $1 < 2$ 비교가 true여서 반복 부분을 반복하고 두번째 돌때 j 값이 2이고 i 값은 여전히 2이여서

$2 <= 2$ 비교가 true여서 반복 부분을 반복하고 세번째 돌때 j 값이 3이고 i 값은 여전히 2여서
 $3 <= 2$ 비교가 false이므로 반복문을 빠져나와 반복문이 종료 된다.



상위 첫번째 이미지와 세번째 이미지를 확인해 보면 동일하지만 네번째 이미지 순서도를 이용 해서 같은 결과를 얻을 수 있다.. 이렇게 하면 조건식을 첫번째 돌때 j 값이 1이고 i 값은 3이므로 $1 <= 3$ 비교가 true여서 반복 부분을 반복하고 두번째 돌때 j 값이 2이고 i 값은 여전히 3이여서 $2 <= 3$ 비교가 true여서 반복 부분을 반복하고 세번째 돌때 j 값이 3이고 i 값은 여전히 3이여서 비교식 $3 <= 3$ 의 결과가 true여서 반복 부분을 반복하고 네번째 돌때 j 값이 4이고 i 값은 여전히 3이여서 $4 <= 3$ 비교 결과가 false이므로 반복문을 빠져나와 반복문이 종료 된다.

잘 생각해보면 상위 3개의 순서도는 i 값이 1에서 하나씩 증가하여 3이 될 때까지 반복하는 것과 동일하여 다음과 같이 순서도를 변경할 수 있다.



1) 다음 문자열을 출력 할 수 있는 기찻길 순서도 의사 코드를 만들어 보자.

1. *엔터**엔터***엔터****엔터*****엔터*****엔터*****엔터

2. *****엔터*****엔터*****엔터****엔터***엔터**엔터*엔터

3. 공간공간공간공간공간*엔터공간공간공간공간**엔터공간공간공간공간**
*엔터공간공간공간****엔터공간공간*****엔터공간*****엔터*****엔터

공간을 진짜 공간 키보드의 스페이스 ‘ ’로 찍고 *은 *로 찍고 엔터는 실제 엔터
줄바꿈으로 표시하면 아래와 같은 삼각형 모양이 나올 것이다.

```
*      **** * * *
**      *** * * *
***      ** * *
****      * *
*****      *
***** *      **** * *
***** * *      * *
***** * * *      *
***** * * * *      *
```

2) 웹에서 자바 별찍기로 검색을 해서 다양함 별을 찍어 보자.

11111	12345	*****	*	*****	*****	
22222	23456		**	****	****	*
33333	34567	*	***	***	***	***
44444	45678	*	****	**	**	*****
55555	56789	*	*****	*	*	*****
		*			**	*****
12345	56789		*****	*	***	*****
12345	45678	*****	***	**	****	*****
12345	34567	*****	**	***	*****	***
12345	23456	*****	**	****		*
12345	12345	*****	*	*****		

다음 6,5,1,8,7,4,2,3을 값으로 가지는 배열이 아래와 같이 있을 때 배열의 내용을
1,2,3,4,5,6,7,8순으로 오름차 순으로 배열안의 내용을 바꾸어 정렬해 보자.

다음 버블정렬을 이해해 보자.

1. 배열의 인덱스 0과 1에 들어 있는 수를 비교하여 큰수를 배열의 인덱스 1쪽으로 교환하여 이동 시킨다.
2. 0이 작으면 교환되지 않는다.

```
int arr[]={6,5,1,8,7,4,2,3};  
if(arr[0]>arr[1]) {  
    int temp;  
    temp=arr[0];  
    arr[0]=arr[1];  
    arr[1]=temp;  
}
```

3. 배열의 인덱스 1과 2에 들어 있는 수를 비교하여 큰수를 배열의 인덱스 2쪽으로 교환하여 이동 시킨다.

```
//{5,6,1,8,7,4,2,3};  
if(arr[1]>arr[2]) {  
    int temp;  
    temp=arr[1];  
    arr[1]=arr[2];  
    arr[2]=temp;  
}  
//{5,1,6,8,7,4,2,3};
```

4. 이작업을 배열이 끝날때 까지 반복한다.
5. 이 작업을 배열이 끝날때 까지 반복하면 배열안에 가장 큰 수가 맨 마지막에 정렬 된다.

```
for(int i=0;i<arr.length-1;i++){  
    if(arr[i]>arr[i+1]) {  
        int temp;  
        temp=arr[i];  
        arr[i]=arr[i+1];  
        arr[i+1]=temp;  
    }  
}
```

6. 이 작업을 2번 하면 1번째에 가장큰 수가 배열의 맨 마지막으로 이동하고 2번째에 그다음 큰수가 배열의 뒤에서 2번째 위치로 이동 한다.

```
for(int i=0;i<arr.length-1;i++){  
    if(arr[i]>arr[i+1]) {  
        int temp;  
        temp=arr[i];  
        arr[i]=arr[i+1];  
        arr[i+1]=temp;  
    }  
}
```

```

    }
    for(int i=0;i<arr.length-1;i++){
        if(arr[i]>arr[i+1]){
            int temp;
            temp=arr[i];
            arr[i]=arr[i+1];
            arr[i]=temp;
        }
    }
}

```

7. 반복문을 1번 돌때마다 1개의 데이터가 배열의 뒤쪽부터 정렬되므로 배열의 개수만큼 반복하면 배열의 모든 데이터가 정렬 된다.

```

for(int j=0;j<arr.length-1;j++){
    for(int i=0;i<arr.length-1;i++){
        if(arr[i]>arr[i+1]){
            int temp;
            temp=arr[i];
            arr[i]=arr[i+1];
            arr[i]=temp;
        }
    }
}

```

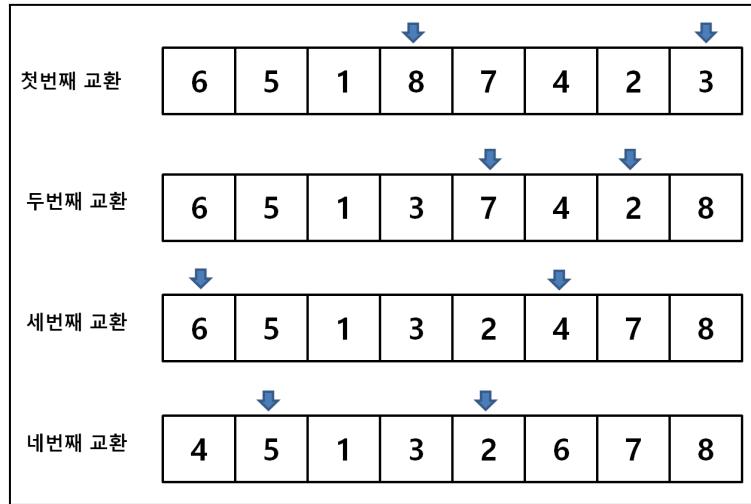
8. 내림차수로 정렬하고 싶다면 작은수를 뒷쪽으로 보내면 된다.

```

int [] arr= {7,5,9,0,3,1,6,2,4,8};
for(int i=0;i<arr.length;i++) {
    System.out.print(arr[i]+" ");
}
System.out.println();
for(int j=0;j<arr.length-1;j++) {
    for(int i=0;i<arr.length-2;i++) {
        if(arr[i] <arr[i+1]) {
            int temp=arr[i];
            arr[i]=arr[i+1];
            arr[i+1]=temp;
        }
    }
}

```

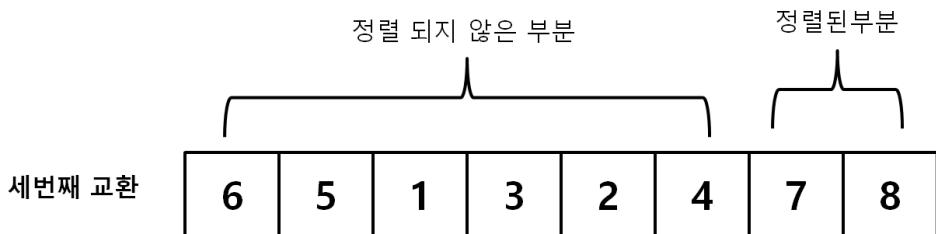
다음 선택정렬을 이해해 보자.



1. 배열안에서 가장 큰 수(첫번째 교환 부분의 첫 번째 화살표)를 찾고 배열의 마지막 위치(첫번째 교환 부분의 두 번째 화살표)와 두 값을 교환한다. 다음을 진행하고 나면 배열 안에 어떤 수가 들어 있더라도 가장 큰 수가 배열의 맨 마지막에 들어 간다.
 2. 첫번째 교환이 이루어 지고 난 다음에는 배열의 맨마지막에 가장 큰 수가 들어가고 배열 안의 내용은 두번째 교환 부분과 같은 모양이 된다. 여기서 오름차순으로 배열 내용을 만든다고 생각해 보자. 맨 마지막 부분은 이미 정렬이 끝난 상태이므로 더 이상 손을 댈 필요가 없다. 다음에 어떤 작업을 해할지 고민해보자.



3. 두 번째 교환 부분에서 정렬 된 부분과 정렬되지 않은 부분을 구분한 다음 정렬된 부분은 더 이상 손대지 않으면 되고, 정렬되지 않은 부분중에서 가장 큰수 (두 번째 교환 부분의 첫 번째 화살표)를 찾아 정렬되지 않은 부분의 마지막 배열 부분(두 번째 교환 부분의 두 번째 화살표)의 값과 교환하면 세 번째 교환 부분 배열처럼 정렬이 된다.



4. 세 번째 교환 부분에서 정렬 된 부분과 정렬되지 않은 부분을 구분한 다음 정렬된 부분은 더 이상 손대지 않으면 되고, 정렬되지 않은 부분중에서 가장 큰수 (세 번째 교환 부분의 첫 번째 화살표)를 찾아 정렬되지 않은 부분의 마지막 배열 부분(세 번째 교환 부분의 두 번째

화살표)의 값과 교환하면 네 번째 교환 부분 배열처럼 정렬이 된다. 이런 방법으로 계속 반복하다 보면 배열안의 모든 내용이 정렬될 것이다.

총 몇번 반복해야 정렬이 되는가?

3개라면 2개만 정렬되면 나머지 하나는 저절로 정렬이 된다. 따라서, 배열크기-1 한 만큼 반복하면 배열은 정렬이 된다. 이후 부터 배열의 크기를 배열이름.length로 표기 한다.

상위 내용을 정리해 보면 정렬하기 위해서 다음과 같은 과정을 거치면 된다.

0~7 까지 8개 중에 가장 큰수를 찾아 배열 인덱스 7에들어 있는 수와 교환해야 한다.

0~6 까지 7개 중에 가장 큰수를 찾아 배열 인덱스 6에들어 있는 수와 교환해야 한다.

0~5 까지 6개 중에 가장 큰수를 찾아 배열 인덱스 5에들어 있는 수와 교환해야 한다.

0~4 까지 5개 중에 가장 큰수를 찾아 배열 인덱스 4에들어 있는 수와 교환해야 한다.

0~3 까지 4개 중에 가장 큰수를 찾아 배열 인덱스 3에들어 있는 수와 교환해야 한다.

0~2 까지 3개 중에 가장 큰수를 찾아 배열 인덱스 2에들어 있는 수와 교환해야 한다.

0~1 까지 2개 중에 가장 큰수를 찾아 배열 인덱스 1에들어 있는 수와 교환해야 한다.

반복되는 숫자를 확인해 보면 반복문으로 바꿀수 있을 것이다.

> 08. 다양한 예제

문제 1) 1~50까지의 짝수를 출력하는 코드를 만들어 보자.

문제 2) 1~100사이의 10의 배수를 출력하는 코드를 만들어 보자. 어떤 숫자가 배수 인지 아닌지 알고 싶으면 7로 나눈 나머지가 0이면 7의 배수이고 아니면 7의 배수가 아니다.

문제 3) 30~300까지의 6의 배수의 합을 출력하는 코드를 만들어 보자.

문제 4) 숫자를 하나 입력 받아 1부터 입력한 수까지 순서대로 화면에 출력 되도록 코드를 만들어 보자.

문제 5) 사용자에게 두 수를 입력받아 두 수의 사이에 있는 모든 수를 오름차순으로 출력하는 순서도와 프로그램을 만들어 보자. 예) 5 9 입력시 6 7 8 더한 결과를 얻음

문제 6) 두수를 입력 받아 사이에 있는 짝수를 화면에 오름차순으로 출력 되도록 순서도와 프로그램을 만들어 보자.

문제 7) $1-2+3-4+5-6\dots+99-100$ 의 결과를 구하는 프로그램을 작성해 보자.

문제 8) $1/2+2/3+3/4+4/5+\dots+99/100$ 의 결과를 구하는 프로그램을 작성해 보자.

문제 9) 피보나치 수열을 10개를 순서대로 출력하는 프로그램을 작성해 보자. 피보나치 수열이 무엇인지는 웹사이트를 검색해서 스스로 알아보자.

문제 10) 원하는 색의 전구와 밝기를 입력 받아 처리하는 프로그램을 만들어 보자. 사용자가 원하는 밝기와 색상을 입력받아 원하는 결과를 출력 받자. 아래 설명한 내용대로 운영되도록 기술 하자.

초기 변수값 : `color="빨강" brightness =50`

값의 범위 : `color=빨강, 노랑, 파랑` `brightness는 0~100`

제안사항 : `brightness`의 숫자 변경은 1씩 가능하다. `brightness`값이 10이라면 다음 `brightness`값은 11이나 9만 가능하다. 10을 50으로 변경하려면 반복문을 사용해야 한다.

사용자입력변수 : `colorInput, brightnessInput`

최종결과값출력 : `p("현재 색상은"+color+"밝기는"+brightness+"이다");`

문제 11) $1-2+3-4+5-6\dots+99-100$ 의 결과를 구하는 순서도와 프로그램을 작성해 보자.

문제 12) $1/2+2/3+3/4+4/5+\dots+99/100$ 의 결과를 구하는 순서도와 프로그램을 작성해 보자.

문제 13) 다음과 같이 출력 되도록 프로그램을 완성해 보자.

1 2 3 4 5

10 9 8 7 6

11 12 13 14 15

20 19 18 17 16

21 22 23 24 25

문제 14) 해당 달의 시작 요일과 일수를 입력 받아 달력을 출력해 보자.\t 탭을 이용해서 만들어 보자.

문제 15) 배열 1,2,3,4,5,6,7,8,9에서 이동방향, 이동칸수, 채울수자를 입력 받아 배열의 내용을 변경후 출력해보자.

ex) 입력 왼쪽 3 2 결과 4,5,6,7,8,9,2,2,2

ex) 입력 오른쪽 3 4 결과 4,4,4,1,2,3,4,5,6

문제 16) 배열 1,2,3,4,5,6,7,8,9에서 회전방향과 회수를 입력받아 배열 내용을 회전시키고 출력해보자.

ex) 입력 왼쪽 2 결과 3,4,5,6,7,8,9,1,2

ex) 입력 오른쪽 3 결과 7,8,9,1,2,3,4,5,6

문제 17) 배열을 100개 선언하여 0~99까지 넣은 다음 i=2 부터 50까지 i를 제외한 i의 배수와 같은 인덱스에 0를 넣은 다음 배열에 0이 아닌 수를 출력해 보자. 출력 결과가 모두 소수인데 이유를 생각해 보자.

ex) i가 2이면 2를 제외한 2의 배수는 4, 6, 8, 10, 12, 14, 16... 등이 있고 해당 인덱스에 0을 넣으면된다.

ex) i가 3이면 3를 제외한 3의 배수는 6,9,12,15,18... 등이 있고 해당 인덱스에 0을 넣으면된다.

문제 18) 17번에서 배열의 값이 0이 아닌 수를 출력해보면 나오는 결과물이 소수이다. 소수란 1과 자기 자신만으로 나눠지는 수를 의미한다.

```
double sum=0;
for(int i=1;i<100;i++) {
    sum+=(double)i/(i+1);
}
System.out.println(sum);

String color="빨강";
int brightness=50;

String colorInput="노랑";
int brightnessInput=70;

color=colorInput;
while(brightness!=brightnessInput) {
    if(brightness>brightnessInput) {
        brightness--;
    }else {
        brightness++;
    }
}
System.out.println("현재 색상은"+color+"밝기는"+brightness+"이다");

//1 1 2 3 5 8
int preValue=1;
int curValue=1;
System.out.println(preValue);
for(int i=0;i<9;i++) {
    System.out.println(curValue);
    int temp=curValue;
    curValue=preValue+curValue;
    preValue=temp;
}
```

> 04. 상속

상속은 기존 클래스의 속성과 메서드를 그대로 사용하면서 새로운 기능이 추가된 클래스를 만들어 재사용하고 싶을 때 사용한다.

기존 클래스를 부모 클래스, 새로 만든 클래스를 자식클래스라 한다. 자식 클래스에서는 부모 클래스에 정의된 `public`으로 선언된 속성과 메서드 뿐 아니라 추가로 자식 클래스에 기술된 속성과 메소드를 사용 할 수 있다.

```
class 부모클래스 {
    // 부모 클래스의 속성과 메서드들
}

class 자식클래스 extends 부모클래스 {
    // 자식 클래스의 새로운 속성과 메서드들
    // 새로 만든 자식 클래스는 기존부모클래스가 가지고 있던 기능과 본인이 가지고 있는 메소드 모드를 그대로 사용할 수 있다.
}

class RectanglePillar extends Rectangle{
    //추가 내용
}
```

RectanglePillar 클래스는 Rectangle클래스의 기능과 추가된 기능을 모두 사용할 수 있는 클래스가 된다. 상속하여 만든 RectanglePillar를 사용하는 방법은 일반 클래스를 생성하는 것과 동일하다. RectanglePillar rp=new RectanglePillar();

다음 예제를 확인해 보자. Child 클래스는 Parent클래스를 extends로 상속 받아 만든 클래스여서 Child클래스로 생성된 인스턴스는 Parent클래스에 있는 print메소드를 사용할 수 있다.

```
class Parent {  
    public void print() {  
        System.out.println("I am the parent.");  
    }  
}  
  
class Child extends Parent {  
    public void study() {  
        System.out.println("I am the child.");  
    }  
}  
class Child2 {  
    public void print() {  
        System.out.println("I am the parent.");  
    }  
    public void study() {  
        System.out.println("I am the child.");  
    }  
}  
public class Main {  
    public static void main(String[] args) {  
        Child child = new Child();  
        child.print(); //상속을 이용해서 print메소드 사용  
        child.study();  
        Child2 child2 = new Child2();  
        child2.print(); //상속 없이 print메소드 사용  
        child2.study();  
    }  
}
```

2.

누군가가 만들어 놓은 클래스에 본인이 새로운 기능을 추가하게 되면 기존 클래스의 내용과 본인이 만든 코드가 섞여 유지 보수하기 어렵다. 이런 문제를 해결하기 위해 기존 클래스의 기능을 그대로 유지 한 상태에서 새로운 클래스를 만들어 원하는 기능을 추가할 수 있는 방법을 지원하는데 이때 사용한 방법이 상속이다.

기존클래스를 부모클래스로 새로운 기능이 추가된 클래스를 자식클래스라고 한다.

부모에 있는 메소드를 자식이 제정의 하면 자식 클래스의 인스턴스는 제정의 된 메소드가 출력이 된다. 제정의 전문용어로 override 오버라이드 라한다.

```
class Animal {
    private String name;

    public Animal(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void speak() {
        System.out.println("The animal speaks");
    }
}

class Dog extends Animal {
    public Dog(String name) {
        //부모의 생성자를 선택해서 호출 기술하지 않으면 기본 생성자super()호출
        super(name);
    }

    @Override
    public void speak() {
        System.out.println("The dog barks");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal("Animal");
        System.out.println(animal.getName());
        animal.speak();

        Dog dog = new Dog("Dog");
        System.out.println(dog.getName()); //부모에만 있는 메소드
        dog.speak(); //부모 자식에 다있는 메소드
    }
}
```

3. supper를 이용해서 부모의 생성자에 접근할 수 있다.

```
class Person {
    private String name;
```

```
private int age;
public Person(String name, int age) {
    this.name = name;          this.age = age;
}
public String getName() {
    return name;
}
public int getAge() {
    return age;
}
public void introduce() {
    System.out.println("name :" + name + " age : " + age);
}
}
class Student extends Person {
    private int grade;
    public Student(String name, int age, int grade) {
        super(name, age);
        this.grade = grade;
    }
    public int getGrade() {      return grade;    }
    public void study() {
        System.out.println("I am studying.");
    }
}
class Teacher extends Person {
    private String subject;
    public Teacher(String name, int age, String subject) {
        super(name, age);
        this.subject = subject;
    }
    public String getSubject() {
        return subject;
    }
    public void teach() {
        System.out.println("I am teaching " + subject + ".");
    }
}
public class Main {
    public static void main(String[] args) {
        Person person = new Person("John", 30);
        person.introduce();

        Student student = new Student("Jane", 20, 3);
        student.introduce();
        student.study();
    }
}
```

```

Teacher teacher = new Teacher("Smith", 40, "Math");
teacher.introduce();
teacher.teach();
}
}

```

사각형 넓이 구하는 Ractangle클래스는 width, height필드와 넓이를 구하는 area라는 메소드를 가지고 있다. 이를 업그레이드해서 사각기둥을 구하는 RectanglePillar클래스는 기존 Ractangle클래스가 가지고 있는 내용에 추가로 사각기둥의 높이 필드와 부피를 구하는 volum 메소드를 가지고 있으면 된다.

Rectangle의 기능을 RectanglePillar가 그대로 가지고 있으므로 Rectangle 클래스 내부의 코드를 그대로 복사해서 RectanglePillar안에 붙여 넣은 다음 추가 기능들을 기술하는 방법이 있는데, 이런 작업을 반복 한다면 상당히 귀찮을 것이다. 그래서, 생각해 낸 방법이 상속으로 기존 클래스의 기능을 그대로 가지고 와서 새로운 기능을 추가하는 방법이다.

기존 코드를 붙여 넣기 하는것 보다 상속을 이용해서 RectanglePillar클래스를 만드는 방법 상속을 이용해서 Ractangle 클래스의 코드에 영향을 주지 않고 동일한 기능을 가지는 RectanglePillar클래스를 만들수 있다.

Rectangle클래스를 부모 클래스 RectanglePillar클래스를 자식클래스라고 한다. 생성하는 방법은 다음과 같다.

다음 코드를 확인해 보자. 기존 클래스를 상속 받아 새로운 클래스를 만들고 싶다면 19번 라인 처럼 extends를 사용하면 된다.

RectanglePillar 클래스는 Rectangle클래스의 기능과 추가된 기능을 모두 사용할 수 있는 클래스가 된다. 상속하여 만든 RectanglePillar를 사용하는 방법은 일반 클래스를 생성하는 것과 동일하다. RectanglePillar rp=new RectanglePillar();

```

5 class Rectangle{
6     public double width=1;  public double depth=1;
7     public double area() {
8         return width*depth;
9     }
10    public Rectangle() {}
11    public Rectangle(double width) {
12        this(width,width);
13    }
14    }
15    public Rectangle(double width,double depth) {
16        super();      this.width=width;      this.depth=depth;
17    }
18 }
19 class RectanglePillar extends Rectangle{
20     public double height=0;
21     public double volume() {
22         return area()*height;
23     }
24     public RectanglePillar(){}
25     public RectanglePillar(double width,double depth,
26                           double height){
27         super(width,depth);  this.height=height;
28         //this.width=width;this.depth=depth;this.height=height;
29         //super.width=width;super.depth=depth;this.height=height;
30     }
31 }
```

```

33 public class Java532 {
34     public static void main(String[] args) {
35         Rectangle r=new Rectangle(5);
36         System.out.println("사각형의 넓이:"+r.area());
37
38         RectanglePillar rp=new RectanglePillar(3,4,5);
39         System.out.println("사각형의 넓이:"+rp.area());
40         System.out.println("사각형의 부피:"+rp.volume());
41 }
```

39번처럼 rp를 통해서 부모에서 상속받은 area 메소드를 실행 할 수 있고 40번 volume 메소드처럼 새로 추가한 메소드도 사용할 수 있다. 이처럼 기존 클래스 Rectangle를 상속해서 RectanglePillar클래스를 만들었다면 Rectangle를 부모클래스, RectanglePillar클래스를 자식클래스라 부른다.

부모에서 접근 제한자를 private로 선언한 것은 자식에서 사용할 수 없으니 자식에서 사용하고 싶다면 접근 제한자는 public이나 protected로 선언 하여야 한다. protected는 자식에서만 접근할 수 있는 접근제한자인데 복잡하게 사용하는 것보다는 private와 public 2개만 사용하는 것이 처음에는 이해가 쉽다.

혹시 자식클래스에서 부모클래스의 필드나 메소드를 접근하고 싶을때는 super를 사용한다.

`super`.필드 `super`.메소드로 기술하면 된다. 식별이 가능할 때는 생략 할 수 있다.

16,27번에 기술한 `super()`는 부모의 생성자를 호출하는 것이다. 27번 처럼 원하는 부모의 생성자를 호출 할 수 있다. 매개변수가 있는 부모의 생성자를 호출 하려면 `super`다음에 소괄호 안에 매개변수를 넣어서 매개변수가 있는 부모의 생성자를 실행 시킬수 있다.

10번 라인을 주석 한다면 16번 라인에서 에러가 발생 한다. 자식 인스턴스가 생성되면 반드시 자신의 생성자 뿐아니라 부모의 생성자도 실행되어야 한다. 16번 라인이 주석을 하고 사용자가 부모 생성자를 지정하지 않으면 부모의 디폴트 생성자 `super()`가 호출 된다. 10번 라인이 없다면 기본 생성자 `super()`가 자동 호출 되면서 부모에 디폴트 생성자 `super()`를 호출할 수 없어서 에러가 발생 한다. `super()`를 이용한 보모 생성자 호출은 `this()`처럼 생성자 상단에 위치해야 하며 본인 생성자 보다 먼저 호출되므로 반드시 `super()` 다음에 `this()`를 기술하여야 한다. 사용자가 명시하지 않으면 컴파일러가 자동으로 기본 생성자를 추가한다.

22번 라인에서 부모의 메소드를 호출하였다. 자신의 클래스에 `area()`가 없어도 부모를 상속받았기 때문에 부모에 있는 `area()`가 호출된다. 부모의 메소드여서 `super.area()`라 기술해야 하지만 식별 가능하여 자식클래스에서 `super`.은 생략 되었다. 되도록 생략하지 않는 것이 좋다.

부모의 필드와 메소드를 접근할때 `super.` 를 사용하고 자기 자신의 필드나 메소드를 접근할 때 `this.`를 사용하면 되고 지역변수는 변수명만 사용하면 되고 사용 할 때 같은 이름이 존재하면 지역, `this`, `super` 순으로 변수를 검색한다. 식별이 가능하면 `this.`이나 `super.`은 생략 가능 하지만 명확성을 위해서 생략하지 말자.

25~30라인을 살펴보자. `RectanglePillar`클래스의 메소드에서 변수를 찾는 순서는 해당 메소드의 지역변수(메소드에서 선언한 변수, 매개변수)를 처음으로 찾고 `RectanglePillar` 클래스의 필드(`this`)에서 찾고, 부모(`super`)에서 찾는다. 같은 이름이 있으면 지역변수, 자식클래스의 필드, 보모클래스의 필드 순으로 찾는다는 이야기이다. 만약 영역별로 같은 이름의 변수가 있고 특정 영역에 변수에 접근하고 싶다면 지역 변수는 변수명, 본인의 클래스의 필드는 `this.`변수명 부모의 필드는 `super.`변수명을 사용하면 된다.

`super.height`를 기술하면 부모에 `height`필드가 없으므로 에러이다.

`this.width`는 `width`가 자기 자신에 없고 부모에 있으나 상속되면 자기 자신에서 접근 가능하다 보니 에러없이 동작된다. 원래 부모에 위치하고 있으니 `super.width`로 기술하는 것이 더 나은 방법이다.

상위 코드를 확인해 보고 원기둥, 삼각기둥 클래스를 만들어 보자. 추가 가능한 필드와 메소드들을 더 추가해보자.

> 05. 오버라이드

오버라이드는 부모의 메소드를 자식 클래스에서 같은 이름으로 재정의 하는 것을 의미한다. 다음 코드를 확인해 보자.

```

public class Animal {
    public void move() {
        System.out.println("동물이 움직입니다.");
    }
}

public class Dog extends Animal {
    @Override // 없어도 되는 부분인데 부모의 메소드를 재정의 했다는 주석 같은 것임
    public void move() {//부모에 있는 메소드를 새로운 기능으로 재정의 하였다.
        System.out.println("강아지가 네 발로 움직입니다.");
    }
}

public class Cat extends Animal {
    @Override //어노테이션 이라 부르는 컴파일 주석이다.
    public void move() {//부모에 있는 메소드를 새로운 기능으로 재정의 하였다.
        System.out.println("고양이가 네 발로 움직입니다.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal1 = new Animal();
        Dog dog2 = new Dog();
        Cat cat3 = new Cat();
        animal1.move();
        dog2.move();
        cat3.move();
    }
}

```

다음은 메소드 오버라이드(override) 방법에 대해서 설명할 예정이다. 오버로딩과 오버라이딩을 혼갈려 하지 말자. 오버로딩(overload)이란? 메소드는 기본적으로 이름이 모두 달라야 하지만, 매개 변수 개수나 자료형이 다를 때 식별 가능하므로 같은 이름으로 여러 개의 메소드를 정의해서 사용할 수 있는데 이런 방법을 오버로딩 이라한다.

오버라이드란? 결국 자식 클래스에서 부모 클래스의 메소드를 같은 이름으로 재정의하여 사용하는 것을 의미한다.

부모 클래스의 메소드를 자식 클래스의 메소드로 이름을 같게 하여 재정의하면 자식 클래스의 인스턴스에서는 제정의한 메소드가 실행되는데 이를 오버라이드라 한다.

다음 코드를 확인해 보자. 부모 클래스 Pet에 있는 메소드인 eat 메소드를 자식 클래스인 Cat, Dog 클래스에서 재정의 해서 선언 하였다.

```

3 class Pet {
4     public void eat() {//자식에도 있는 메소드
5         System.out.println("Pet이 밥을먹는다.");
6     }
7     public void move() {//자식에는 없는 메소드
8         System.out.println("Pet이 움직인다.");
9     }
10 }
11 class Cat extends Pet {
12     @Override
13     public void eat() {
14         System.out.println("Cat가 생선을 먹는다.");
15     }
16     public void work() {//부모에 없고 자식에만 있는 메소드
17         System.out.println("고양이가 줄을 잡는다.");
18     }
19 }
20 class Dog extends Pet {
21     @Override
22     public void eat() {
23         System.out.println("Dog가 뼈다리를 먹는다. 육식");
24     }
25     public void hunting() {//부모에 없고 자식에만 있는 메소드
26         System.out.println("Dog가 꿩을 사냥한다.");
27     }
28 }

```

4,13,22번 라인에서 부모 Pet클래스의 eat메소드를 자식클래스 Cat,Dog 에서 재정의 하여 사용하고 있다.

다음코드를 메인에 기술하고 결과를 확인해 보자.

Pet a = new Pet();	a.eat();	//Pet이 밥을먹는다.
Cat c = new Cat();	c.eat();	//Cat가 생선을 먹는다.

Pet클래스에 eat메소드가 있지만 Pet클래스를 상속 받은 Cat클래스에서 eat메소드를 호출하면 부모에 있는 eat메소드가 실행되는 것이 아니고 Cat클래스에서 재정의된 eat메소드가 출력된다.

부모의 메소드를 자식에서 재정의해서 사용하는것을 오버라이드라고 한다. Dog d=new Dog(); d.eat();를 실행시키면 어떤 결과가 나올지 생각해 보자.

12, 21라인의 @Override는 어노테이션 이라는 것인데 컴파일러 주석이다 여기서는 부모가 가지고 있는 메소드를 자식이 재정의 하였다는 것을 의미 한다. 본인이 부모에 있는 eat이라는 메소드를 재정의 할때 실수로 eat1이라는 메소드로 이름을 붙였다면 Cat클래스에 eat1이라는 메소드를 재정의 한것이 아니고 추가한 것이 된다. 의도는 다르지만 에러는 아니다. 하지만, 메소드 상단에 @Override 어노테이션을 추가하면 eat1은 재정의 한 메소드가 아니므로 에러가 발생한다. 사전에 예상치 못한 에러를 방지하기 위해서 사용한다.

```
class Cat extends Pet {  
    @Override  
    public void eat1() {  
        System.out.println("Cat가 생선을 먹는다.");  
    }  
}
```

다음은 각 객체에서 출력할 수 있는 메소드 종류이다. eat는 재정의되어 출력 결과가 모두 다르고 move는 부모에 있으니 모든 인스턴스에서 접근할 수 있다. work와 hunting은 특정 자식 인스턴스에서만 접근할 수 있다. 출력 결과를 확인해 보고 어떤 메소드가 실행되었는지 본인이 생각한 것과 같은지 확인해 보자.

```
Pet a = new Pet(); a.eat(); a.move();  
Cat c = new Cat(); c.eat(); c.move(); c.work();  
Dog d = new Dog(); d.eat(); d.move(); d.hunting();
```

상위 코드와 같은 경우의 부모자식 관계 클래스들을 생각해보고 만들어 보자.

비슷한 형태의 오버라이드된 클래스를 만들어 보자.

> 06. 다형성

다형성은 부모 클래스 변수에 자식 클래스의 객체를 할당할 수 있는 능력을 의미합니다.

상속을 통해서 부모 자식 클래스가 만들어 지면 부모 클래스 변수에 자식 클래스를 넣을 수 있고, 이런 것을 다형성이라 한다.

다형성을 통해 부모 클래스 변수에 자식 클래스의 인스턴스를 할당하면, 해당 변수를 통해 메서드 호출 시 실제 객체의 타입에 따라 오버라이딩된 자식 클래스의 메서드가 호출됩니다.

Animal은 부모클래스이고 Dog는 자식 클래스이고 둘다 move메소드를 가지고 있다면, 부모 클래스 인스턴스에 자식 클래스를 할당하고 move메소드를 실행하면 자식 메소드 move()가 실행 된다. Animal animal2 = new Dog(); animal2.move(); Dog 클래스의 move메소드가 실행 된다.

부모 클래스 인스턴스에 부모 클래스 인스턴스를 넣고 move메소드를 실행하면 부모클래스의 move메소드가 실행된다. Animal animal2 = new Animal(); animal2.move();

자식 클래스 인스턴스에 자식 클래스 할당하고 move메소드를 실행하면 자식 클래스의 move메소드가 실행된다. Dog d=new Dog(); d.move();

자식 클래스 인스턴스에 부모클래스 할당하면 에러가 난다. Dog d= new Animal(); //error 설명이 복잡하니 아래 예제를 실행해보고 상위 글들을 이해해 보자.

```
//Animal 클래스:  
public class Animal {  
    public void makeSound() {  
        System.out.println("동물이 소리를 냅니다.");  
    }  
}  
//Dog 클래스:  
public class Dog extends Animal {  
    @Override  
    public void makeSound() {  
        System.out.println("멍멍!");  
    }  
}  
//Cat 클래스:  
public class Cat extends Animal {  
    @Override  
    public void makeSound() {  
        System.out.println("야옹~");  
    }  
}  
//Main 클래스:  
public class Main {
```

```

public static void main(String[] args) {
    //변수를 사용한 다형성
    Animal animal1 = new Dog();
    Animal animal2 = new Cat();
    Animal animal3 = new Animal();

    //실제로 할당된 객체의 메소드가 실행된다.
    animal1.makeSound(); //Dog의 메소드
    animal2.makeSound(); //Cat의 메소드
    animal3.makeSound(); //Animal의 메소드

    //array를 이용한 다형성
    Animal[] animals = new Animal[3];
        animals[0] = new Dog();
        animals[1] = new Cat();
        animals[2] = new Animal();

        for (Animal animal : animals) {
            animal.makeSound();
        }

    //arrayList를 이용한 다형성
    Animal a1 = new Dog();
    Animal a2 = new Cat();
    Animal a3 = new Animal();
    ArrayList<Animal> animals = new ArrayList<>();
    animals.add(a1);
    animals.add(a2);
    animals.add(a3);

    for (Animal animal : animals) {
        animal.makeSound();
    }
}
}

```

일반적으로 클래스 변수에는 다음과 같은 형태의 클래스 인스턴스만 넣을 수 있다.

```
A a=new A(); B b=new B(); Dog d=new Dog(); Cat c=new Cat();
```

다형성은 다양하다는 의미이다. 필통에 연필만 꽂을 수 있다면 다형성이 없는 것이고 필통에 연필, 볼펜, 지우개등 다양하게 넣을 수 있다면 다형성이 있는 것이다.

프로그램에서 다형성은 하나의 클래스 변수에 다양한 형태의 인스턴스를 넣을 수 있다는 의미이다.

일반적인 형태로는 불가능 하지만 상속 관계에 있을때 부모클래스 변수에 자식 인스턴스는 넣을 수 있다. 자식 클래스 변수에 부모 클래스 인스턴스는 넣을 수 없다. 부모가 같은 자식끼리도 넣을 수 없다. 오직 부모에만 담을 수 있다.

```
class A{}, class B extends A{}, class C extends A{}
```

A a=new A(); A a=new B(), A a=new C()은 되고 B b=new C(); B b=new A();은 안된다.

다음 클래스는 Shape라는 도형 클래스를 상속받아 Triangle이라는 삼각형 클래스를 만들었다.

```
1 package com.human.ex3;
2 class Shape{
3     void drow1() {
4         System.out.println("Shape:drow1");
5     }
6     void drow2() {
7         System.out.println("Shape:drow2");
8     }
9 }
10 class Triangle extends Shape{
11     @Override
12     void drow2() {
13         System.out.println("Triangle:drow2");
14     }
15     void drow3() {
16         System.out.println("Triangle:drow3");
17     }
18 }
```

Shape클래스의 변수에
Shape인스턴스, Shape클래스의
변수에 Triangle인스턴스, Triangle
클래스 변수에 Triangle인스턴스를
넣을 때 어떤 메소드가 실행되는지
다음 코드를 확인해 보자.

```
20 public class Java540 {
21     public static void main(String[] args) throws Exception {
22         //Triangle은 Shape를 상속하여 만들었다.
23         //Shape에는 drow1,2메소드가 있고 Triangle에는 drow2,3메소드가 있다.
24         System.out.println("Shape에 Shape를 넣어 출력-----");
25         Shape s1=new Shape();
26         s1.drow1();      s1.drow2();
27         System.out.println("s1.drow3() Shape에 없는 메소드여서 접근불가");
28
29         System.out.println("Triangle에 Triangle를 넣어 출력----");
30         Triangle t1=new Triangle();
31         t1.drow1(); //Triangle에는 없지만 부모(Shape)에 있어서 출력이 된다.
32         t1.drow2(); //부모자식에 다있지만 재정의된 자식의 drow2()가 실행된다.
33         t1.drow3();
34
35         //Triangle t2=new Shape(); 자식에 부모를 넣을수 없다.
36         System.out.println("Shape(부모)에 Triangle(자식)를 넣어 출력----");
37         Shape s2=new Triangle(); //부모에 자식을 넣을수 있다.(다형성)
38         s2.drow1();
39         //s2는 Shape(부모)이지만 들어있는것이 Triangle(자식)이여서
40         //다형성에 위해서 Triangle(자식)에 있는 drow2()가 출력된다.
41         s2.drow2();
42         //s2.drow3(); //drow3은 Triangle에만 있는 메소드
43         //s2에들어 있는것이 Triangle이지만 s2자체는 Shape여서 접근 불가"
44
45         //s2.drow3()를 출력하고 싶다면 원래의 모양으로 강제형변환 해야한다.
46         Triangle t2=(Triangle)s2;          t2.drow3();
47         //((Triangle)s2).drow3();
48     }
49 }
```

```

3 class Pet {
4     public void eat() { //자식에도 있는 메소드
5         System.out.println("Pet이 밥을먹는다.");
6     }
7     public void move() { //자식에는 없는 메소드
8         System.out.println("Pet이 움직인다.");
9     }
10 }
11 class Cat extends Pet {
12     @Override
13     public void eat() {
14         System.out.println("Cat가 생선을 먹는다.");
15     }
16     public void work() { //부모에 없고 자식에만 있는 메소드
17         System.out.println("고양이가 쥐를 잡는다.");
18     }
19 }
20 class Dog extends Pet {
21     @Override
22     public void eat() {
23         System.out.println("Dog가 빠다귀를 먹는다.육식");
24     }
25     public void hunting() { //부모에 없고 자식에만 있는 메소드
26         System.out.println("Dog가 펑을 사냥한다.");
27     }
28 }

```

이전에 구현하였던 Pet 관련 클래스들을 가지고 다형성을 확인해보자.

```
Pet a0 = new Pet(); Pet a1 = new Cat(); Pet a2 = new Dog();
```

```
a0.eat(); //Pet이 밥을먹는다.
a1.eat(); //Cat가 생선을 먹는다.
a2.eat(); //Dog가 빠다귀를 먹는다.
```

Pet,Cat,Dog 3개의 인스턴스를 부모 클래스인 Pet의 변수에 넣어서 eat메소드를 실행해 보면 Pet클래스의 eat메소드가 출력 되어 모두 pet이 밥을 먹는다가 찍히는 것이 아니고, Pet클래스

변수에 들어 있는 실제 인스턴스의 eat메소드가 출력된다.

자식 인스턴스가 부모 클래스 변수에 들어 가면 재정의된 자식 메소드는 접근 할 수 있고 재정의 하지 않은 자식 메소드들은 접근 할 수 없다.

다음 코드를 확인해 보자.

```
a1.work(); a2.hunting(); ((Cat) a1).work(); ((Dog) a2).hunting();
((Cat) a0).work(); //에러가 없는것 처럼 보이지만 실행시켜 보면 문제가 발생한다.
```

a1에 들어 있는 내용이 Cat 인스턴스이지만 a1이 Pet으로 선언되어 있으므로 work() 메소드는 재 정의된 메소드가 아니여서 접근할 수 없다. hunting()메소드도 마찬가지다. 만약 접근하고자 한다면 형변환을 통해서 본래의 형태로 형변환 해야 접근할 수 있다. 실제 들어 있던 클래스와 동일한 클래스로 형변환 해야 한다. 잘못 형변환을 해도 에러가 없는 것처럼 보이지만 실행시 에러가 발생하니 형변환시 주의하자.

다형성을 이용해서 부모 클래스 변수에 자식 인스턴스가 들어가면 부모 멤버는 모두 접근 가능하고 오버라이드된 자식의 메소드는 접근, 가능하고 자식에서 추가된 멤버들은 접근

불가능하고, 만약 접근하고 싶다면 원래의 클래스로 형변환하여 접근 해야 한다.

```
Pet arr[] = new Pet[3];
arr[0] = a0;
arr[1] = a1;
arr[2] = a2;

for (Pet p : arr) {
    p.eat();
}
```

상속관계가 있는 여러 객체의 특정 메소드를 한번에 실행 시키고 싶다면 최상위 부모 객체를 배열로 선언한 다음 관련이 있는 여러 클래스의 인스턴스를 배열에 넣고 다형성을 이용하여 재정의된 메소드를 출력 하면된다.

왼쪽 코드를 확인해 보자.

```
3 class Pet {
4     public static void petEattting(Pet a) {
5         a.eat();
6     }
7     public static void petEattting(Cat a) {
8         a.eat();
9     }
10    public static void petEattting(Dog a) {
11        a.eat();
12    }
}
```

다음 코드에서 Pet 클래스에 애완 동물에게 밥을 먹이는 petEattting 클래스 메소드를 Pet클래스에 추가 하였다. 다형성이 존재하지 않는다면 매개 변수로 받아 eat메소드를 실행하는 메소드를 다음 코드 처럼 클래스 개수 3개 만큼 만들어야 한다. 다형성을

이용한다면 4번 메소드 한개만 있어도 3개의 메소드가 다 있을때와 동일하게 동작한다.
7~12라인 까지 주석을 해보고 주석하기 전과 후가 동일하게 동작하는지 확인해 보자.

```
2 class Student{
3     //학생관련 여러 메소드 생략
4     public void information() {
5         System.out.println("학생의 정보를 출력");
6     };
7 }
8 class HSchoolStudent extends Student{
9     public void information() {
10        System.out.println("고등학생의 정보를 출력");
11        System.out.println("원하는 진로정보 출력");
12    };
13 }
14 class UniversityStudent extends Student{
15     public void information() {
16         System.out.println("대학생의 정보를 출력");
17         System.out.println("원하는 취업처 정보 출력");
18    };
19 }
```

왼쪽의 코드는 학생 클래스를 상속받은 고등학생, 대학생 클래스를 만들어 information() 메소드를 재정의 한 것이다.

다음 코드는 Student 클래스 배열에 Student 클래스를 상속받은 클래스 인스턴스를 배열에 넣고 다형성을 이용해서 배열에 들어 있는 모든 인스턴스의 infomation 메소드를 출력하는 방법을 보여준 코드이다.

```

21 public class Java607 {
22     public static void main(String[] args) {
23         //부모인 Student클래스에 다형성을 이용해서 Student와
24         //관련된 클래스들을 넣을 수 있다.
25         Student []sts= {
26             new Student(),new HSchoolStudent(),
27             new UniversityStudent()
28         };
29         //학생 클래스와 관련된 모든 인스턴스의 information()메소드를
30         //실행해보자.
31         for(Student st:sts) {
32             System.out.println("-----");
33             st.information();
34         }
35     }
36 }
37 }
```

다음 코드는 Human 클래스에 클래스 메소드와 인스턴스 메소드를 다형성을 이용하여 Student관련 인스턴스를 출력하는 메소드를 선언한 것이다.

이렇게 메소드를 선언하면 Student와 Student클래스의 자식 클래스 모두를 매개변수로 받아 처리 할 수 있다.

```

2 class Human{
3     public static void humanStudentInformation(Student st) {
4         st.information();
5     }
6     public void studentInformation(Student st) {
7         st.information();
8     }
9 }
```

```

10 public class Java608 {
11     public static void main(String[] args) {
12         //클래스메소드는 하나인데 다형성을 이용해서 Student와 관련된 모든 클래스의
13         //information메소드를 출력할 수 있다.
14         Human.humanStudentInformation(new Student());
15         Human.humanStudentInformation(new HSchoolStudent());
16         Human.humanStudentInformation(new UniversityStudent());
17
18         Human h=new Human();
19         Student s1=new Student();
20         HSchoolStudent hs1=new HSchoolStudent();
21         UniversityStudent us1=new UniversityStudent();
22         h.studentInformation(s1);
23         h.studentInformation(hs1);
24         h.studentInformation(us1);
25
26
27         System.out.println(s1 instanceof Student);
28         System.out.println(s1 instanceof HSchoolStudent);
29         System.out.println(s1 instanceof UniversityStudent);
30
31         System.out.println(hs1 instanceof Student);
32         System.out.println(hs1 instanceof HSchoolStudent);
33
34         Student s5=new HSchoolStudent();
35         System.out.println(s5 instanceof Student);
36         System.out.println(s5 instanceof HSchoolStudent);
37         System.out.println(s5 instanceof UniversityStudent);
38 }
```

왼쪽 코드는 다형성을 이용해서 하나의 메소드로 관련있는 클래스를 출력하는 방법이다.

instanceof는 해당 클래스와 인스턴스가 관계가 있으면 true 관계가 없으면 false를 리턴하는 연산자이다.

다음 문제를 풀어보자.

제품 종류로 TV와 컴퓨터가 있는데 제품마다 5% 10% 할인을 해주고 있다. TV에는 인치, 컴퓨터에는 RAM 크기와 같은 특별한 속성이 있고 두 제품다 최종가격을 리턴하는 getPrice 메소드가 있는데 TV의 경우 10인치가 넘어가면 가격에 배송료 만원을 추가하고 Ram 같은 경우 1기가당 2만원을 가격에 추가한다.

1. 부모 클래스로 Product 클래스를 만들어서 모든 제품들이 가지고 있는 필드와 메소드를 구현하였다.

```
class Product {  
    public String id="";    public double price=0;  
    public double discountRate=0;  
    public Product(String id, double price, double discountRate) {  
        this.id = id;      this.price = price;  
        this.discountRate = discountRate;  
    }  
    public double getPrice() {  
        return price-price*(discountRate/100);  
    }  
}
```

2. TV제품에 만있는 필드와 TV에만 적용되는 계산 방법을 이용해서 TV클래스를 만들었다. 가격 계산 방법이 달라 getPrice를 재정의 하였다.

```
class TV extends Product{  
    public double inch=0;//inch가 10이상이면 추가배송 만원 추가  
    public TV(String id,double price,double discountRate,double inch) {  
        super(id, price, discountRate);      this.inch = inch;  
    }  
    @Override  
    public double getPrice() {  
        return inch>10?super.getPrice()+10000:super.getPrice();  
    }  
}
```

3. Computer제품에 만있는 필드와 Computer에만 적용되는 계산 방법을 이용해서 Computer클래스를 만들었다. 가격 계산 방법이 달라 getPrice를 재정의 하였다.

```
class Computer extends Product{  
    public Computer(String id,double price,double discountRate,double ram){  
        super(id, price, discountRate);      Ram = ram;  
    }  
    public double Ram=0;//ram 1기가 당 2만원 추가  
    @Override  
    public double getPrice() {  
        return super.getPrice()+Ram*2;  
    }  
}
```

다음을 확인해 보자.

고객은 VIP고객과 일반 고객이 있는데 VIP고객은 구매 요금에 2%, 일반 고객은 구매 요금에 1%를 포인트 적립해 주고 VIP 고객만 특별히 제품 가격을 5%할인 해 준다. 포인트는 모든 물건 할인후 최종 결제 금액으로 포인트에 추가해 준다. 정보를 출력할 수 있도록 `toString`를 구현해 보자.

1. 일반 고객을 표현한 수 있는 `Customer`클래스를 만들어 보자. 아이디, 보유 포인트, 물건 구매시 포인트 취득 비율을 넣을 수 있는 필드와 `point`를 구하는 `setPoint` 메소드를 가지고 있다.

```
class Customer{
    public String id="";
    public int point=0;
    public double pointRate=0;
    public void setPoint(Product p) { //제품 p에 대한 point 적립
        this.point=(int)(p.getPrice()*(pointRate/100));
    }
    public Customer(String id, int point, double pointRate) {
        this.id = id;      this.point = point;
        this.pointRate = pointRate;
    }
    @Override
    public String toString() {
        return "Customer [id=" + id + ", point=" + point + ", ";
    }
}
```

2. VIP고객을 표현할 수 있는 `VIPCustomer`클래스를 만들어 보자. 아이디, 보유 포인트, 물건 구매시 포인트 취득 비율을 넣을 수 있는 필드와 `point`를 구하는 `setPoint` 메소드를 가지고 있다.

```
class VIPCustomer extends Customer{
    public double priceRate=0;
    public VIPCustomer(String id,int point,double pointRate,double priceRate){
        super(id, point, pointRate);      this.priceRate = priceRate;
    }
    public void setPoint(Product p) { //제품 p에 대한 point 적립
        double resultPrice=p.getPrice()-(p.getPrice()*(priceRate/100));
        this.point+=(int)(resultPrice*(pointRate/100));
    }
    @Override
    public String toString() {
        return "VIPCustomer [priceRate=" + priceRate + ", id=" + id + ", point=" + point + "]";
    }
}
```

3. 특정 고객이 특정 제품을 구매할때 고객의 포인트를 추가할 수 있도록 메인 클래스에 다음 프로그램을 넣어 완성해 보자.

```
Product p1=new TV("LG0015",100000,10,40);           // (id, price, discountRate, inch)
Product p2=new Computer("AMD0051",100000,20,16); // (id, price, discountRate, ram)
Customer c1=new Customer("hong",0,1);             // (id, point, pointRate)
Customer c2=new VIPCustomer("kim",0,2,5);        // (id, point, pointRate, priceRate)
c1.setPoint(p1);       c2.setPoint(p2);
System.out.println(c1);   System.out.println(c2);
```

4. 상위와 같은 형태의 관계 클래스들을 만들어 본인이 생각하는 기능을 추가하여 각각 프로그램을 구현해 보자.

- a. 부모:Vehicle(탈것) 자식:Car(차), Bus(버스)
- b. 부모:shape(도형) 자식:Rectangle(사각형), Circle(원)

> 07. JAVA API

프로그램을 만들 때 일일이 모든 프로그램을 만드는 것이 아니고 많이 사용하는 클래스를 전문 프로그래머가 미리 만들어 놓고 개발자들이 만들어진 클래스를 가져다가 원하는 프로그램을 작성한다. 이때 전문 프로그래머가 만든 클래스 사용방법을 정의해 놓은 문서를 API 혹은 라이브러리, 프레임워크라 부른다. 세 가지 의미가 조금씩 다르지만 거의 비슷한 의미로 쓰인다.

`System.out.println()`은 화면에 문자열을 출력하는 메소드인데 구현부를 몰라도 우리는 잘 사용해 왔다. 전문 프로그램이 만든 API를 사용한 것이다. API를 잘 활용하면 새로 구현 할 필요 없이 원하는 기능을 사용 할 수 있다.

자바 프로그래머가 미리 만들어 놓은 클래스들을 JAVA API라고 하고 다음과 같이 기술된 문서를 제공해 주면 일반 프로그래머들이 이를 이용해 프로그램을 한다. 다음 주소는 `java api`를 정리해 놓은 사이트이다.

<https://docs.oracle.com/javase/8/docs/api/index.html>

내용이 전문적이어서 API만으로는 이해하기 어렵지만 API를 보는 습관이 필요하다. 시간 날 때마다 아는 내용을 문서로 읽어보고 정리해 보자.

간단히 많이 사용하는 `String` 클래스 API 살펴보자.

문자열을 저장하는 클래스로 문자열과 관련된 메소드들을 가지고 있다. 스트링 관련 메소드들을 살펴보자.

```
System.out.println("equals 메소드.....");
String str=new String("안녕하세요");
if(str.equals("안녕하세요")) {
    System.out.println("같은 문자열입니다."); //true
}else {
    System.out.println("다른 문자열입니다.");
}
```

equals 메소드

객체의 주소비교가 아닌 들어 있는 문자열을 비교한다.

```
System.out.println("concat 메소드.....");
String str1="Hello";
String str2="World";
String str3=str1.concat(str2);
System.out.println(str3); //HelloWorld
System.out.println(str1); //Hello
```

concat() 메소드

두 문자열을 합친다. String st1=“hello”;
String st2= “world”;
String st3=st1.concat(st2);

`System.out.println(st3);` 출력 결과는 “hello world”이며 `st1.concat`은 `st1` 내용을 변경하는 것이 아니고 값이 유지된 상태로 새로운 문자열을 만든다는 것에 주의하자.

```
System.out.println("charAt 메소드.....");
String str4="abcdef";
System.out.println(str4.charAt(2)); //c
```

charAt()메소드 해당 문자열에서 원하는 인덱스의 문자를 출력하는 프로그램이다. String str1= “abcdef”에서 str1.charAt(2)를

사용하면 str1문자열에서 2번째 인덱스에 있는 문자가 리턴된다. 인덱스는 0부터 실행되므로 ‘c’가 출력 된다.

length()메소드

문자열의 문자 개수를 리턴한다. String str1= “abcdef”; str1.length는 6이 된다.

```
System.out.println("indexOf 메소드.....");
str4="abcdef";
System.out.println(str4.indexOf("cd")); //2
System.out.println(str4.indexOf("ac")); //-1
```

indexOf()메소드

매개변수로 넘겨진 문자열과 같은 문자열이 존재하는 인덱스를 리턴한다. 앞에서

부터 찾는다. String str1= “abcdef”; str1.indexOf(“cd”); 의 경우 2가 리턴된다. 문자열을 찾지 못하면 -1이 리턴 된다.

```
System.out.println("lastIndexOf 메소드.....");
str4="abcdef";
System.out.println(str4.lastIndexOf("bc")); //1
System.out.println(str4.lastIndexOf("ac")); //-1
```

lastIndexOf()메소드

매개변수로 넘겨진 문자열의 존재하는 인덱스를 리턴한다. 뒤에서부터 찾는다. String

str1= “abcdef”; str1.lastIndexOf(“cd”);의 경우 2가 리턴된다. 문자열을 찾지 못하면 -1이 리턴 된다.

substring()메소드

매개변수로 인덱스를 받아 해당 문자열의 특정부분을 추출한다.

String str1= “abcdef”; str1.substring(2);의 경우 “cdef”가 리턴된다.
str1.substring(2,4);의 경우 “cd”가 리턴 된다.

```
System.out.println("subString 메소드.....");
str4="abcdef";
System.out.println(str4.substring(3)); //str4 3번 인덱스부터 나머지출력
System.out.println(str4.substring(2,3)); //str4 2부터 3번 인덱스 출력
```

split()메소드

매개변수로 받은 문자열을 기준으로 문자열을 분리해서 문자열 배열을 리턴한다. String str1= “사과, 배, 바나나”; String arr[] = str1.split(“,”); 실행 결과 arr의 0번째 인덱스에 사과 1번째 인덱스에 배 2번째 인덱스에 바나나가 들어 있다.

```

System.out.println("split 메소드.....");
String str5="사과, 배%-바나나-귤";
//문자 , % - 로 문자열 분리
//String arr[] = str5.split(", |%-");
//문자열 "%-"로 분리
String arr[] = str5.split("%-"); // "사과, 배"와 "바나나-귤"로 분리
for(String st:arr) {
    System.out.println(st);
}

```

.toLowerCase()메소드, .toUpperCase()메소드

해당문자열을 소문자 혹은 대문자로 모두 변경하여 리턴 한다.

```

System.out.println(".toLowerCase, .toUpperCase 메소드.....");
String str6="ABCdef";
System.out.println(str6.toUpperCase()); //ABCDEF
System.out.println(str6.toLowerCase()); //abcdef

```

trim()메소드

문자열 앞뒤로 공백을 제거한 문자열을 리턴한다.

```

System.out.println("trim 메소드.....");
String str7=" AB cd ";
System.out.println(str7.trim()); //문자열 앞뒤로 공백 제거

```

다음 문제를 풀어보자.

주민등록을 입력받아 정규식으로 주민등록 번호가 맞는지 확인하고 생년월일과 성별을 출력해 보자. 7번째 자리가 1,3이면 남자 2,4이면 여자이다. 앞에 6자리는 생년월일을 의미한다.

정규식 관련 클래스 API를 확인해 보자.

정규식은 특정한 패턴을 가진 문자열들을 사용하는 표현식 언어이다. 정규식 패턴을 스스로 만드는 방법은 많은 내용을 공부해야 한다. 처음부터 만들어서 사용하려면 많은 어려움이 생길 것이다. 시간있을 때 틈틈이 공부해 보도록 하고 일단은 필요할 때 검색창에 “자바 정규식 이메일”처럼 필요한 정규식을 입력해 찾아낸 다음 복사해서 사용 하자. 다음은 간단한 정규식 예제이다 확인해 보자. 메인에 기술하면 되고 빨간 줄이 생기면 Pattern클래스나 Matcher클래스가 제대로 import되어 있는지 확인 하자.

```
import java.util.regex.Matcher; import java.util.regex.Pattern;
```

```

//주민등록 정규식
String num = "\d{6}[-]{1-4}\d{6}";
Pattern np = Pattern.compile(num);
String test1 = "123456-1234567";
Matcher m1 = np.matcher(test1);
if(m1.matches()) {
    System.out.println("주민 형식이 맞습니다. " + test1);
} else {
    System.out.println("주민 형식이 아닙니다. "+ test1);
}
//핸드폰번호 정규식
String phone = "^01(?:0|1|6-9])-?(?:\d{3}|\d{4})-\d{4}$";
Pattern pp = Pattern.compile(phone);
String test2 = ("010-1111-2222");
Matcher m2 = pp.matcher(test2);
if(m2.matches()) {
    System.out.println("휴대전화 형식이 맞습니다. " + test2);
} else {
    System.out.println("휴대전화 형식이 아닙니다. "+ test2);
}
//이메일 정규식
String email = "^[_a-z0-9-]+(?:[_a-z0-9-]+)*@(?:\w+\.\.)+\w+$";
Pattern ep = Pattern.compile(email);
String test3 = ("foxman12@naver.com");
Matcher m3 = ep.matcher(test3);
if(m3.matches()) {
    System.out.println("이메일 형식이 맞습니다. " + test3);
} else {
    System.out.println("이메일 형식이 아닙니다. "+ test3);
}

```

첫 라인에 "\d{6}[-]{1-4}\d{6}"은 주민등록 정규식 문자열이다. String num에 넣은 다음 Pattern.compile(num); 처럼 매개 변수를 이용해서 특정 정규식 문자열 인지 확인하는 Matcher 인스턴스를 얻어와 .matcher 메소드로 검정 할 문자열을 선택한 다음 .matches메소드로 일치여부를 판단 한다. 리턴값이 true이면 정규식과 검정할 문자열이 일치한 것이고 false이면 다른 것이다.

정규식을 따로 공부하려면 시간이 많이 들어간 웹에서 java 정규식을 검색해서 가지고와 프로그램을 구현하자.

문제1) 이메일, 핸드폰, 주민등록 번호가 ,표로 구분해서 기술되어 있는 데이터에서 해당 데이터 별로 문자열을 모아 출력 되도록 프로그램을 구현해 보자.

> 10. 추상클래스

추상 메소드란 구현부가 없는 메소드를 의미한다.

추상 메소드를 가지고 있는 클래스를 추상 클래스라 한다.

추상 클래스는 구현부가 없어서 인스턴스로 생성할 수 없다.

추상 클래스의 인스턴스를 생성하려면 추상 클래스를 상속 받은 자식 클래스에서 부모 클래스에 존재하는 추상 메소드들을 모두 재정의 한후 자식 클래스를 인스턴스로 생성하면 사용할 수 있다.

추상 메소드는 다음 설명할 인터페이스를 위해서 설명한 것이지 실제 사용할 일이 없다. 간단히 추상메소드가 무엇인지 알고 넘어가면 된다. 다음 코드를 보고 이해해 보자.

```
abstract class Shape {  
    private String color;  
    // 추상 메서드 선언 구현부가 없다  
    public abstract double getArea();  
    public String getColor() {  
        return color;  
    }  
    public void setColor(String color) {  
        this.color = color;  
    }  
}  
// 추상 클래스 Shape를 상속받은 하위 클래스 Rectangle  
class Rectangle extends Shape {  
    private double width;  
    private double height;  
    public Rectangle(double width, double height) {  
        this.width = width;  
        this.height = height;  
    }  
    // 추상 메서드 구현  
    @Override  
    public double getArea() {  
        return width * height;  
    }  
}  
// 메인 클래스에서 추상 클래스와 하위 클래스 사용 예시  
public class Main {  
    public static void main(String[] args) {  
        Rectangle rectangle = new Rectangle(5.0, 10.0);  
        rectangle.setColor("Red");  
        System.out.println("색상: " + rectangle.getColor());  
        System.out.println("넓이: " + rectangle.getArea());  
    }  
}
```

```

//Shape shape1 = new Shape(); //추상클래스는 생성할 수 없다.
    Shape shape1 = new Rectangle(5.0, 10.0); //다형성
    shape1.setColor("Red");
    System.out.println("색상: " + shape1.getColor());
    System.out.println("넓이: " + shape1.getArea());

// 다음 예제가 동작하는 서클 클래스를 만들어 보자
//Shape shape2 = new Circle(7.0); //다형성
//shape2.setColor("Blue");
//System.out.println("색상: " + shape2.getColor());
//System.out.println("넓이: " + shape2.getArea());
}

}

```

```

3 abstract class Product {
4     abstract public void getPrice();
5 }
6 class TV extends Product {
7     @Override
8     public void getPrice() {
9         System.out.println("TV가격 5십만원");
10    }
11 }
12 class Computer extends Product {
13     @Override
14     public void getPrice() {
15         System.out.println("컴퓨터가격 6십만원");
16    }
17 }

```

4번 라인을 보면 메소드의 구현부가 없고 메소드 시작 부분에 `abstract`라는 키워드를 넣었다. 이것이 추상 메소드이다. 추상 메소드를 포함한 클래스를 추상 클래스라고 한다. 추상클래스는 3번 라인처럼 `class` 앞에 `abstract`를 붙인다.

추상 클래스는 `new` 연산자를 이용해서 인스턴스를 생성 할 수 없다. 추상 클래스의 인스턴스를 생성하고 싶으면

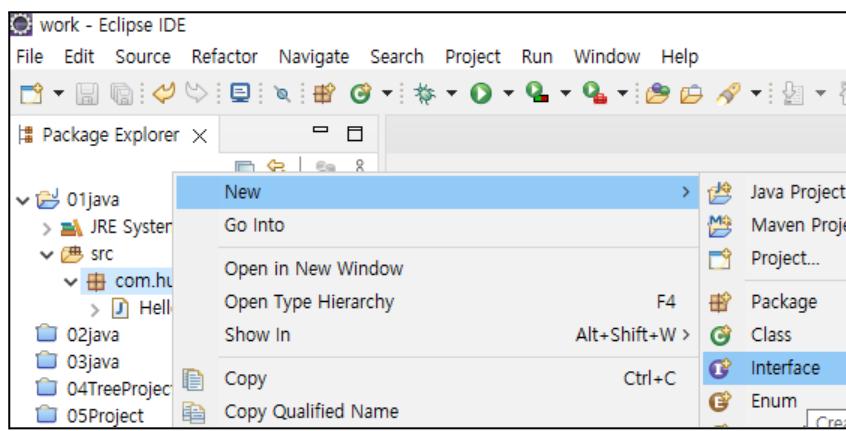
추상 클래스를 상속받은 자식 클래스에서 추상 클래스의 추상 메소드를 모두 재정의 한 다음에 자식 클래스로 인스턴스를 생성할 수 있다.

실제 코드에서 추상 메소드나 추상 클래스를 사용할 일은 거의 없고 인터페이스 설명을 위해서 사용 한다.

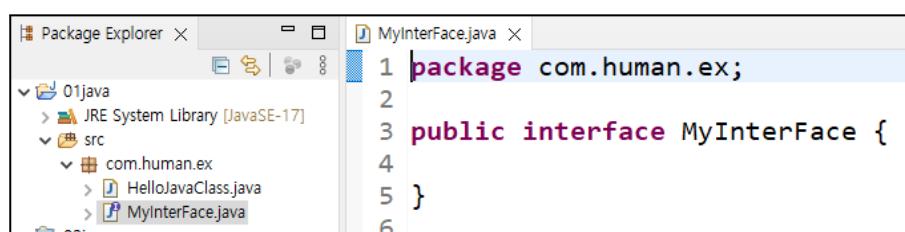
> 11. 인터페이스

인터페이스란? 구현부가 없는 추상 메소드로만 만들어진 클래스를 인터페이스라 생각하면 된다. 사용 하려면 인터페이스를 상속한 자식 클래스에서 모든 메소드를 재정의 한후 인스턴스를 생성해서 사용 한다. 다형성을 사용할 목적으로 인터페이스를 만들어 사용한다.

추상 메소드만으로 이루어져 있고 class 대신에 interface라 기술하고 extends 대신에 implements라 기술한다. 다형성을 사용하기 위해서 인터페이스를 사용한다. 다음 예제를 확인해 보자.



인터페이스를 추가하는 방법은 클래스를 추가하는 방법과 동일하다. src에서 오른쪽 마우스를 클릭하고 new >> interface를 선택한 다음에 클래스 파일처럼 패키지와 인터페이스 이름을 입력하고 완료하면 다음 이미지처럼 인터페이스 파일이 만들어 진다.



인터페이스도 클래스처럼 파일이름과 인터페이스 이름이 동일하고 패키지를 기술해야 하고, 다른 패키지에서 사용하려면 전체 경로를

기술 해야 한다. 인터페이스 이름은 대문자로 시작한다.

다음 인터페이스 파일을 기술해서 실행결과를 확인해 보자.

```
public interface Shape {//interface 키워드를 사용하여 인터페이스를 선언
    public double getArea();// 추상 메서드 선언
    public double getPerimeter();// 추상 메서드 선언
}
//implements 키워드를 사용하여 인터페이스를 구현하는 클래스를 선언
public class Circle implements Shape {
    private double radius;
    public Circle(double radius) {
        this.radius = radius;
    }
    public double getArea() {//추상 메서드는 반드시 구현
        return Math.PI * radius * radius;
```

```

}

public double getPerimeter() {//추상 메서드는 반드시 구현
    return 2 * Math.PI * radius;
}
}

public class Rectangle implements Shape {
    private double width;
    private double height;
    public Rectangle(double width, double height) {
        this.width = width;           this.height = height;
    }
    public double getArea() {
        return width * height;
    }
    public double getPerimeter() {
        return 2 * (width + height);
    }
}
public class Triangle implements Shape {
    private double side1;
    private double side2;
    private double side3;
    public Triangle(double side1, double side2, double side3) {
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }
    public double getArea() {
        double s = (side1 + side2 + side3) / 2;
        return Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }
    public double getPerimeter() {
        return side1 + side2 + side3;
    }
}
public class Main {
    public static void main(String[] args) {
        Shape[] shapes = new Shape[3];
        shapes[0] = new Circle(5);
        shapes[1] = new Rectangle(3, 4);
        shapes[2] = new Triangle(3, 4, 5);
        for (Shape shape : shapes) {
            System.out.println("A:"+shape.getArea()+",P:"+shape.getPerimeter());
        }
        Shape s1 = new Circle(2);
    }
}

```

```

        Shape s2 = new Rectangle(4, 5);
        Shape s3 = new Triangle(6, 7, 8);
        ArrayList<Shape> shapesList = new ArrayList<>();
        shapesList.add(s1);
        shapesList.add(s2);
        shapesList.add(s3);
        for (Shape shape : shapesList) {
            System.out.println("A:" + shape.getArea() + ", P:" + shape.getPerimeter());
        }
    }
}

```

다음 코드를 확인해 보자.

```

3 interface Product {
4     public void getPrice();
5 }
6 class TV implements Product {
7     @Override
8     public void getPrice() {
9         System.out.println("TV가격 5십만원");
10    }
11 }
12 class Computer implements Product {
13     @Override
14     public void getPrice() {
15         System.out.println("컴퓨터가격 6십만원");
16    }
17 }

```

인터페이스는 다형성과 상속받은 자식 클래스의 개발 방향 설정 용도로 사용 한다. 부모 클래스 변수에 자식 클래스 인스턴스를 넣을 수 있듯이 인터페이스를 상속받은 클래스는 부모 인터페이스 변수에 자식 클래스 인스턴스를 넣을 수 있다.
약속된 형태로 프로그램을 구현 할때 정해진 구조로 인터페이스를 만들어 놓으면 누구나 인터페이스를 상속받아 정해진 형태의 클래스로 프로그램을 구현할 수 있다.

정해진 형태의 클래스로 프로그램을 구현할 수 있다.

인터페이스에서 개발 방향 설정 용도에 대한 예는 다음과 같다. 3~5라인의 Product 인터페이스 정보를 통하여 Product 인터페이스를 상속받은 자식 클래스에는 반드시 getPrice 메소드를 재정의 해야 한다는 정보를 얻을 수 있다. 개발자에게 프로그램을 만들어 달라고 하면 개발자마다 프로그램 형태가 다르지만, 인터페이스를 제공해 주고 만들어 달라 요청하면 어느 정도 동일한 결과물을 보장 받을 수 있다.

접근제한자 사용방법은 클래스와 동일하다. interface 앞에 public, private 같은 접근 제한자를 기술하면 된다. 지금은 하나의 파일로 기술하기 위해서 default 접근 제한자를 사용하였다. 모든 곳에서 사용하고 싶다면 public으로 선언하여야 한다. 메인에 다음 코드를 기술해서 제대로 동작하는지 확인해 보자.

```
//Product p1=new Product(); 인터페이스는 생성할 수 없다.
Product p2=new TV();          p2.getPrice();
Product p3=new Computer();   p3.getPrice();
```

```

interface Product {
    public void getPrice();
}
interface Market {
    public void getMarket();
}
class TV implements Product,Market {
    @Override
    public void getPrice() {
        System.out.println("TV가격 5십만원");
    }
    @Override
    public void getMarket() {
        System.out.println("슈퍼에서구입");
    }
}
public class JavaStart3 {
    public static void main(String[] args) {
        Product p1=new TV(); p1.getPrice(); p1.getPrice();
    }
}

```

인터페이스는 왼쪽 처럼 다중 상속이 가능하다. 상속받은 두개의 인터페이스에 있는 추상 메소드들은 반드시 재정의되어야 사용할 수 있다.

왼쪽 코드를 확인해 보고 다음 코드도 추가해서 동작여부를 확인해 보자.

Market m=new TV();
m.getMarket();

```

2 interface Shape{
3     public void draw1();
4     public void draw2();
5 }
6 class Triangle implements Shape{
7     @Override
8     public void draw1() {
9         System.out.println("Triangle:draw1");
10    }
11    @Override
12    public void draw2() {
13        System.out.println("Triangle:draw2");
14    }
15    void draw3() {
16        System.out.println("Triangle:draw3");
17    }
18 }

```

상속 페이지에서 구현한 Shape, Triangle 클래스에서 shape 클래스를 인터페이스로 변경하여 프로그램을 완성하면 왼쪽 코드와 같다.

다음 페이지에 실행코드도 확인해 보자.

```

19 public class Java540 {
20     public static void main(String[] args) throws Exception {
21         //인터페이스는 추상메소드여서 생성할수 없다.
22         //Shape s1=new Shape();
23
24         System.out.println("Triangle에 Triangle를 넣어 출력----");
25         Triangle t1=new Triangle();
26         //t1에 제정의된 메소드와 추가된 메소드를 출력하기
27         t1.draw1();      t1.draw2();      t1.draw3();
28
29         System.out.println("Shape(부모)에 Triangle(자식)를 넣어 출력----");
30         Shape s2=new Triangle(); //부모에 자식을 넣을수 있다.(다형성)
31         //부모 인터페이스를 통해서 자식메소드 출력가능
32         s2.draw1();      s2.draw2();
33
34         //s2.draw3()를 출력하고 싶다면 원래의 모양으로 강제형변환 해야한다.
35         Triangle t2=(Triangle)s2;          t2.draw3();
36         //((Triangle)s2).draw3();
37     }
38 }
```

```

2 interface Student{
3     public void information();
4 }
5 class HSchoolStudent implements Student{
6     public void information() {
7         System.out.println("고등학생의 정보를 출력");
8         System.out.println("원하는 진로정보 출력");
9     }
10 }
11 class UniversityStudent implements Student{
12     public void information() {
13         System.out.println("대학생의 정보를 출력");
14         System.out.println("원하는 취업처 정보 출력");
15     }
16 }
```

왼쪽 코드와 이후에 나오는 코드는 이전에 다형성에서 설명한 학생, 고등학생, 대학생 관련 코드를 학생 인스턴스를 사용해서 구현한 코드이다 확인해 보자.

부모 클래스를 인터페이스로 선언하였을뿐 크게 달라진 점이 없음을 확인할 수 있다.

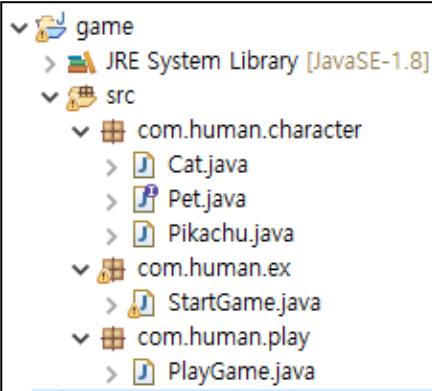
```

17 public class Java607 {
18     public static void main(String[] args) {
19         //부모인 Student인터페이스에 다형성을 이용해서 Student를
20         //상속한 클래스들을 다형성을 이용해서 넣을 수 있다.
21         //new Student()처럼 Student인터페이스여서 인스턴스를 생성할 수 없다.
22         Student []sts= {
23             new HSchoolStudent(),
24             new UniversityStudent()
25         };
26         //학생 클래스와 관련된 모든 인스턴스의 information()메소드를
27         //실행해보자.
28         //collection, 다형성, 인터페이스
29         for(Student st:sts) {
30             System.out.println("-----");
31             st.information();
32         }
33     }
34 }
```

```

2 class Human{
3     public static void humanStudentInformation(Student st) {
4         st.information();
5     }
6     public void studentInformation(Student st) {
7         st.information();
8     }
9 }
10 public class Java608 {
11     public static void main(String[] args) {
12         //클래스메소드는 하나인데 다형성을 이용해서 Student를 상속하여 만든
13         //모든 클래스의 information메소드를 출력할 수 있다.
14         Human.humanStudentInformation(new HSchoolStudent());
15         Human.humanStudentInformation(new UniversityStudent());
16
17         Human h=new Human();
18         HSchoolStudent hs1=new HSchoolStudent();
19         UniversityStudent us1=new UniversityStudent();
20         h.studentInformation(hs1);
21         h.studentInformation(us1);
22
23     }
24 }
```

인터페이스를 이용해서 애완동물 키우는 프로그램을 구현해보자. 애완동물이 죽지 않도록 먹을 것을 주면서 운동을 시켜 성장시키는 프로그램이다. 프로그램에는 다양한 애완동물 캐릭터가 있고 많은 새로운 캐릭터를 추가 할 수 있도록 하기 위해 캐릭터가 개임을 진행할때 필요한 메소드들을 미리 인터페이스로 만들어 놓고 캐릭터를 추가할 때 마다 해당 인터페이스를 상속해서 만들 예정이다.



왼쪽처럼 game이라는 자바프로젝트를 만들었고 com.human.character 패키지에는 애완동물 인터페이스 Pet과 Pet를 상속받아 실제 구현한 애완동물 Cat과 Pikachu 클래스가 있다. com.human.ex 패키지에는 실제 프로그램을 시작하는 main메소드가 들어 있다. com.human.play는 실제 게임 로직이 들어 있다.

```
package com.human.character;
public interface Pet {
    public void eat();
    public void sleep();
    public void play();
    public void train();
    public void levelUp();
    public boolean endGame();
    public void printInfo();
}
```

1. 게임에서 캐릭터 추가시 공통적으로 있어야 하는 Pet인터페이스를 만들어 보자.
2. 피카추(Pikachu)캐릭터를 추가하기 위해서 Pet클래스를 상속받아 피카추(Pikachu)클래스를 만들었다. 다음 이미지는 관련코드들이다.

```
public class Pikachu implements Pet{
    private int experience = 40;
    private int energy = 50;
    private int level = 1;
    @Override
    public void eat() { //피카추에게 먹이를 주는 메소드로 먹이를
        energy += 25; //먹을때마다 energy가 25증가 한다.
    }
    @Override
    public void sleep() {
        energy += 10; //잠을 제워 에너지 10를 증가 시킨다.
    }
    @Override
    public void play() { //놀게 만들어 경험도를 증가 시킨다.
        energy -= 30; experience += 30;
    }
    @Override
    public void train() { //훈련을 통해서 경험도를 증가시킨다.
        energy -= 20; experience += 20;
    }
}
```

```

    @Override
    public void levelUp() {//경험도가 쌓이면 렙업하게 만든다.
        if(experience > 60) {
            experience -= 40;
            level++;
            System.out.println("렙업");
        }
    }
    @Override
    public boolean endGame(){//개임이 끝났는지 확인하는 메소드
        boolean returnValue = true;
        if(level >= 4) {//레벨이 4이면 성공으로 게임을 끝낸다.
            System.out.println("피카추가 용됨");
            returnValue = false;
        }else if(energy <= 0) {//에너지가0이면 실패로 게임을 끝낸다.
            System.out.println("피카추가 굶죽음");
            returnValue = false;
        }
        return returnValue;
    }
    @Override
    public void printInfo() {
        System.out.println("=====");
        System.out.println("피카추 정보입니다.");
        System.out.println("experience: " + experience);
        System.out.println("energy: " + energy);
        System.out.println("level: " + level);
        System.out.println("=====");
    }
}

```

```

public class Cat implements Pet {
    private int experience = 50;
    private int energy = 60;
    private int level = 1;
    public void eat() {}
    public void sleep() {}
    public void play() {}
    public void train() {}
    public void levelUp() {}
    public boolean endGame() {}
    public void printInfo() {}
}

```

3. 새로운 고양이 캐릭터를 추가한다.

4. 다음 이미지에 나와 있는 실제 게임을 진행 할때 사용하는 PlayGame클래스를 만들어 보자. play메소드는 게임을 진행하는 메소드이고 selectCharacter메소드는 게임 진행을 할 캐릭터를 선택하는 메소드이다. gameStart() 메소드는 PlayGame 클래스에 추가된 각종 게임 진행 관련 메소드를 실제로 진행하는 메소드이다.

현재는 selectCharacter()로 캐릭터를 선택한 다음 play()로 게임을 진행하면 된다. 추후 게임에 새로운 기능을 추가하고 싶다면 PlayGame클래스에 새로운 메소드를 추가하고 gameStart() 메소드에서 실행하면 된다.

```

package com.human.play;
import java.util.Scanner;
import com.human.character.Pikachu;
import com.human.character.Cat;
import com.human.character.Pet;
public class PlayGame {
    private Pet character=null;
    private boolean flag = true;
    private Scanner sc = new Scanner(System.in);
    public void gameStart() {
        selectCharacter();
        play();
    }
    public void selectCharacter() {..}
    public void play() { //캐릭터 생성과 무관하게 독립됨..}
}

```

```

public void selectCharacter() {
    System.out.println("캐릭터를 입력하세요.");
    System.out.println("1.피카추 2.고양이");
    String ch = sc.nextLine();
    if(ch.equals("1")) { //캐릭터 추가
        character= new Pikachu();
    }else if(ch.equals("2")) {
        character = new Cat();
    }
}

```

다음은 캐릭터를 생성하는 selectCharacter() 메소드 관련 내용이다. 사용자 입력을 받아 피카추와 고양이중 하나를 character변수에 넣고 있다.

다음은 play()메소드로 캐릭터를 선택하고 나서 해당 게임을 진행하는 메소드이다.

선택한 캐릭터를 가지고 실제적으로 게임을 진행하는 메소드이다. 사용자에게 1.밥먹이기 2.잠재우기 3.놀아주기 4.운동 5.종료와 같은 작업을 선택하게 해서 본인이 선택한 캐릭터를 키울 수 있다. 사용자가 지속적으로 메뉴를 골라 키울 수 있도록 하다가 어느 시점에 캐릭터가 죽거나 성장 시켜 게임을 종료 시킬 수 있다.

```

public void play() {      //캐릭터 생성과 무관하게 독립됨
    while(flag) {
        character.printInfo();
        System.out.println("1.밥먹이기 2.잠자우기 3.놀아주기 4.운동 5.종료");
        System.out.print("입력>>");
        String select = sc.nextLine();
        switch(select) {
            case "1":
                character.eat();      break;
            case "2":
                character.sleep();   break;
            case "3":
                character.play();    break;
            case "4":
                character.train();   break;
            case "5":
                flag = false;       break;
            default:
                break;
        }
        character.levelUp();
        if(flag) {
            flag = character.endGame();
        }
    }
}

```

```

package com.human.ex;
import com.human.play.PlayGame;
public class StartGame {
    public static void main(String[] args) {
        PlayGame pg = new PlayGame();
        pg.gameStart();
    }
}

```

5. 실제 프로그램을 실행시키기 위한 메인메소드를 구현해 보자. 아까 만든 PlayGame클래스의 gameStart()메소드를 호출하면 된다.

> 12. 람다식

람다식은 익명함수를 간편하게 작성할 수 있는 방법이다. 다음과 같은 Calculator 인터페이스를 객체로 생성하려면 아래 처럼 일일이 상속 받아 재구현 하여야 한다. 이런 불편함을 없애기 위해서 람다식을 이용해 재구현 없이 객체를 생성할 수 있다. 다음 예제 2개를 확인해 보자.

1. 람다식을 사용하지 않은 계산기

```
interface Calculator {
    int calculate(int a, int b);
}

class Addition implements Calculator {
    public int calculate(int a, int b) {
        return a + b;
    }
}

class Subtraction implements Calculator {
    public int calculate(int a, int b) {
        return a - b;
    }
}

class Multiplication implements Calculator {
    public int calculate(int a, int b) {
        return a * b;
    }
}

class Division implements Calculator {
    public int calculate(int a, int b) {
        return a / b;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator addition = new Addition();
        Calculator subtraction = new Subtraction();
        Calculator multiplication = new Multiplication();
        Calculator division = new Division();

        System.out.println(addition.calculate(10, 5));
        System.out.println(subtraction.calculate(10, 5));
        System.out.println(multiplication.calculate(10, 5));
        System.out.println(division.calculate(10, 5));
    }
}
```

2. 람다식을 사용한 예제

```
interface Calculator {
    int calculate(int a, int b);
}

public class LambdaExample {
    public static void main(String[] args) {
        Calculator addition = (a, b) -> a + b;
        Calculator subtraction = (a, b) -> a - b;
        Calculator multiplication = (a, b) -> a * b;
        Calculator division = (a, b) -> a / b;

        int num1 = 10;
        int num2 = 5;

        System.out.println(num1+"+"+num2+"="+addition.calculate(num1, num2));
        System.out.println(num1+"-"++num2+"="+subtraction.calculate(num1, num2));
        System.out.println(num1+"*"+num2+"="+multiplication.calculate(num1, num2));
        System.out.println(num1+"/"++num2+"="+division.calculate(num1, num2));
    }
}
```

```
interface Inter1{
    public void z();
}

class I implements Inter1{
    @Override
    public void z() {
        System.out.println("Z 메소드 실행");
    }
}

public class TESTTime2 {
    public static void main(String[] args) {
        //Inter1인터페이스의 z메소드를 사용하려면 클래스로
        //재정의하여 인스턴스를 생성하여 실행해야 한다.
        I i=new I();      i.z();
    }
}
```

람다식이란? 객체지향 언어에서
클래스 선언없이 메소드를 만들어
사용하는 병법이다.

간단한 메소드를 실행하려면 왼쪽
코드 처럼 클래스의 인스턴스를
생성해서 인스턴스의 메소드를
호출하여야 한다. 람다식을
사용하면 아래 코드 처럼 클래스
생성없이 인스턴스의 메소드를 제
정의해 실행 할 수 있다.

```

2 interface Inter1 {
3     public void z();
4 }
5 interface Inter2 {
6     public void z(int a);
7 }
8 interface Inter3 {
9     public void z(int a, int b);
10}
11 public class JavaStart3 {
12    public static void main(String[] args) {
13        Inter1 a=() -> System.out.println("Inter1");
14        a.z();
15        Inter2 b=b1->System.out.println("Inter2"+b1);
16        b.z(10);
17        Inter2 b2 = (b1) -> {
18            System.out.println("Inter2" + b1);
19        };
20        b2.z(11);
21        Inter3 b3 = (a1, b1) -> {
22            int x = a1 + b1;
23            System.out.println(x);
24        };
25        b3.z(11, 21);
26    }
27 }

```

기존 메소드 작성 방법은 다음과 같다.

반환타입 메소드이름(매개변수){ 메소드 내용; return값;}

람다식은 더욱 간단한 형태로 기술 할 수 있다. 리턴값과 매개변수가 없는 타입 없는 람다식은 형태는 `()->{}`과 같고 사용 방법은 다음과 같다.

1. 2~4라인 처럼 인터페이스를 생성한다.
2. 13라인처럼 인터페이스의 인스턴스에 람다식을 넣는다. 여기서 함수 내용이 1줄이면 중괄호를 생략할 수 있다.
3. 14라인처럼 람다식으로 만든 메소드를 실행 할 수 있다.

반환값이 있을 경우 일반 메소드 처럼 `return`를 사용하여 기술하면 된다.

매개 변수가 있을 경우 람다식은 `(a,b)->a+b` 와 같은 형태로 구현한다. 상위 코드에서 `Inter2`, `Inter3` 이 어떻게 만들어 지는지 확인해 보자.

아래 3가지 문장은 같은 메소드이다 람다식으로 생략해서 만든것이다.

`int sum(int a,int b){ return a+b; }`

`(a,b)->{ return a+b; }`

`(a,b)-> a+b`