[객체배열 MVC 실습문제] 다음과 같은 조건을 만족하는 프로그램을 작성 하시오

회원을 관리하는 프로그램으로 MVC 패턴을 이용한 종합 실습문제이다. 회원 등록, 검색, 수정, 삭제, 출력, 정렬에 해당하는 프로그램을 구현하시오. 해당 구현 클래스 다이어그램과 클래스 구조를 참고하여 프로젝트를 완성하시오.

*** 프로젝트 명 : 07_ObjectArray_Homework_본인이름**

1. 구현 클래스 다이어그램 (Class Diagram)

mvc.model.vo.Member
- userId : String // 아이디 - userPwd : String // 비밀번호 - name : String // 이름 - age : int // 나이 - gender : char // 성별 - email : String // 이메일
+ Member() + Member(userId:String, userPwd:String, name:String, age:int, gender:char, email:String) + information() : String

mvc.controller.MemberController
+ SIZE : int // 최대 회원 수 = 10 - memberCount : int // 현재 회원 수 - mem : Member[] // 회원 객체 배열
+ getMemberCount() : int + getMem() : Member[] + checkId(userId:String) : Member + insertMember(m:Member) : void + searchMember(menu:int, search:String) : Member + updateMember(m:Member, menu:int, update:String) : void + deleteMember(userId:String) : void

*** Member의 setter/getter 메소드는 직접 구현한다.**

mvc.view.MemberMenu
- mc : MemberController // 객체 생성 - sc : Scanner // 객체 생성
+ mainMenu() : void + insertMember() : void + searchMember() : void + updateMember() : void + deleteMember() : void + printAllMember() : void

mvc.run.Run
+ main(args:String[]) : void

2. 구현 클래스 설명

Package명	Class명	Method	설명
mvc.run	Run	+main(args:String[]) : void	MemberMenu의 mainMenu 실행
mvc.view	MemberMenu	+mainMenu() : void	반복문을 이용하여 메인 메뉴를 반복적으로 실행 (각 메뉴 별 서브 메뉴 호출)
		+insertMember() : void	메인 메뉴에서 1번 선택시 실행되는 메소드이며 등록할 회원에 대한 정보를 입력 받고 동일한 아이디가 없는 경우 MemberController의 insertMember 메소드 실행
		+searchMember() : void	메인 메뉴에서 2번 선택시 실행되는 서브메뉴 → 반복실행 각 서브메뉴에 해당하는 검색 내용을 입력 받고 MemberController의 searchMember 메소드 실행
		+updateMember() : void	메인 메뉴에서 3번 선택시 실행되는 서브메뉴 → 반복실행 각 서브메뉴에 해당하는 수정할 내용을 입력 받고 MemberController의 updateMember 메소드 실행
		+deleteMember() : void	메인 메뉴에서 4번 선택시 실행되는 메소드이며 삭제할 회원 아이디를 입력 받고 MemberController의 deleteMember 메소드 실행
		+printAllMember() : void	메인 메뉴에서 5번 선택시 실행되는 메소드이며 MemberController의 getMem 메소드 실행의 결과값을 반복문을 통해 출력

mvc.controller	MemberController	+getMemberCount() : int	현재 회원 수를 나타내는 memberCount 변수값을 반환해주는 메소드
		+getMem() : Member[]	현재 회원 객체 배열을 반환해주는 메소드
		+checkId(userId:String) : Member	전달 받은 아이디가 mem 배열에 존재하는 경우 해당 아이디의 회원 정보를 반환해주는 메소드 (없을 경우 null 반환)
		+insertMember(m:Member) : void	전달 받은 회원 정보를 mem 배열에 추가해주는 메소드로 회원 등록 시 memberCount 1 증가
		+searchMember(menu:int, search:String) : Member	매개변수로 전달받은 menu에 따라 1인 경우 전달받은 search 문자열을 아이디로 검색, 2인 경우 이름으로 검색, 3인 경우 이메일로 검색하고 그 결과인 회원 객체를 반환
		+updateMember(m:Member, menu:int, update:String) : void	전달받은 menu에 따라 전달받은 회원 m을 setter 메소드를 이용하여 menu가 1인 경우 비밀번호 수정, 2인 경우 이름 수정, 3인 경우 이메일을 수정하는 메소드
		+deleteMember(userId:String) : void	전달받은 아이디를 이용하여 검색 후 존재하는 경우 삭제해주는 메소드로 삭제 후 다음 인덱스 객체들의 정보를 한 칸씩 앞으로 이동시킴 (memberCount 1감소)

3_1. class 구조 - MemberMenu

```
public class MemberMenu{

    // MemberController 클래스 접근을 위해 필드로 선언
    private MemberController mc = new MemberController();
    private Scanner sc = new Scanner(System.in); // 키보드로 입력 받기 위해서

    public void mainMenu() {

        // 메뉴 출력                    >> 반복 실행 처리함
        ====== 회원 관리 메뉴 ======
        1. 신규 회원 등록              >> insertMember() 실행
        2. 회원 정보 검색              >> searchMember() 실행
        3. 회원 정보 수정              >> updateMember() 실행
        4. 회원 정보 삭제              >> deleteMember() 실행
        5. 회원 정보 출력              >> printAllMember() 실행
        9. 프로그램 종료
    }

    public void insertMember() {
        // 1. 현재 회원 수(memberCount)가 최대 회원 수(SIZE)를 넘어설 경우 return 처리
        //    MemberController의 getMemberCount() 메소드를 통해 현재 회원 수 알아오고
              최대 회원 수는 상수필드여서 클래스명. 으로 직접 접근 가능

        // 2. 아이디를 입력 받아 MemberController의 checkId() 메소드로 전달 >> 결과 값 받기
        //     (아이디 중복 체크하는 과정)

        // 2_1. 결과 값이 null이 아닌 경우 즉, 동일한 아이디가 존재하는 경우
        //      "동일한 아이디가 존재합니다. 회원등록 실패" 출력
        // 2_2. 결과 값이 null인 경우 즉, 동일한 아이디가 존재하지 않는 경우
        //      나머지 회원 정보 입력 받도록 (비밀번호, 이름, 나이, 성별, 이메일)
        //      입력 받은 정보를 Member의 매개변수 생성자를 이용하여 객체 생성 후
        //      MemberController의 insertMember() 메소드로 전달
        //      "성공적으로 회원 등록이 되었습니다." 출력
    }
```

```
public void searchMember() {

    // 메뉴 출력                    >> 반복 실행 처리함
    ====== 회원 정보 검색 ======
    1. 아이디로 검색하기
    2. 이름으로 검색하기
    3. 이메일로 검색하기
    9. 이전 메뉴로
    메뉴 선택 :                   >> 키보드로 입력 받기 (menu : int)
    검색 내용 :                   >> 키보드로 입력 받기 (search : String)

    // 1. MemberController의 searchMember() 메소드로 menu와 search 문자열 전달 >> 결과 값
    // 1_1. 결과 값이 null인 경우 즉, 검색 결과가 없는 경우 >> "검색된 결과가 없습니다." 출력
    // 1_2. 결과 값이 null이 아닌 경우 즉, 검색 결과가 존재하는 경우 >> 회원 정보 출력
}

public void updateMember() {
    // 메뉴 출력                    >> 반복 실행 처리함
    ====== 회원 정보 수정 ======
    1. 비밀번호 수정
    2. 이름 수정
    3. 이메일 수정
    9. 이전 메뉴로
    메뉴 선택 :                   >> 키보드로 입력 받기 (menu : int)
    변경할 회원 아이디 :          >> 키보드로 입력 받기 (userId : String)

    // 1. MemberController의 checkId()로 userId 전달   >> 결과 값 (m : Member)
    // 1_2. 결과 값이 null인경우 >> "변경할 회원이 존재하지 않습니다" 출력
    // 1_2. 결과 값이 null이 아닌 경우 기존 정보 출력 후
    //      변경내용(update : String) 입력 받고
    //      MemberController의 updateMember()에 m, menu, update 전달
    //      "회원의 정보가 변경되었습니다." 출력
}
```

```
public void deleteMember() {

    삭제할 회원 아이디 :                    >> 키보드로 입력 받기 (userId : String)


    // 1. MemberController의 checkId()에 userId 전달   >> 결과 값 (m : Member)
    // 1_1. 결과 값이 null인 경우 "삭제할 회원이 존재하지 않습니다." 출력
    // 1_2. 결과 값이 null이 아닌 경우 기존 정보 출력
    //      "정말 삭제하시겠습니까? (y/n) : "          >> 키보드로 입력 받기
    //      대소문자 구분 없이 'Y'인 경우 MemberController의 deleteMember()에 userId 전달
    //      "회원의 정보가 삭제되었습니다." 출력
}

public void printAllMember() {
    // MemberController의 getMem() 메소드 호출      >> 결과 값 (mem : Member[])
    // 반복문을 통해 결과 값 안의 존재하는 회원들 정보 출력
}
```

3_2. class 구조 - MemberController

```
public class MemberController{

    public static final int SIZE = 10;                  // 최대 회원 수 상수필드로 10 초기화
    private int memberCount;                            // 현재 회원 수 필드
    private Member[] mem = new Member[SIZE];            // 회원들의 정보를 담는 객체 배열

    // 초기화 블럭을 이용하여 기본적인 회원 5명의 정보 초기화, memberCount 수 5 초기화
    {
        mem[0] = new Member("user01", "pass01", "김유신", 20, 'M', "kim12@naver.com");
        mem[1] = new Member("user02", "pass02", "이순신", 60, 'M', "lee2@naver.com");
        mem[2] = new Member("user03", "pass03", "유관순", 17, 'F', "yo5@hanmail.net");
        mem[3] = new Member("user04", "pass04", "연개소문", 57, 'M', "yeon@gmail.com");
        mem[4] = new Member("user05", "pass05", "신사임당", 45, 'F', "shin@naver.com");
        memberCount = 5;
    }

    public void getMemberCount() {
        // memberCount 리턴
    }

    public Member[] getMem() {
        // mem 주소 값 리턴
    }

    public Member checkId(String userId) {
        Member m = null;          // 아이디로 검색된 결과를 담을 변수 초기화

        // mem 에서 매개변수로 전달받은 userId와 동일한 아이디를 가지고 있는 회원을 m에 대입
        // m 리턴
    }

    public void insertMember(Member m) {
        // 매개변수로 전달받은 회원객체를 mem 배열에 추가
        // memberCount 1 증가
    }
```

```
public Member searchMember(int menu, String search) {

	Member searchMember = null;		// 검색된 회원 정보를 담을 변수 초기화

	// 매개변수로 전달받은 search 문자열을 menu 번호에 따라

	// 1 인 경우 아이디로 검색 후 결과를 searchMember에 대입하고
	// 2 인 경우 이름으로 검색 후 결과를 searchMember에 대입하고
	// 3 인 경우 이메일로 검색 후 결과를 searchMember에 대입하고

	// searchMember 주소 값 리턴
}

public void updateMember(Member m, int menu, String update) {

	// 매개변수로 전달받은 m 회원과 변경 내용인 update 문자열을 menu에 따라

	// 1 인 경우 setter 메소드를 이용하여 m의 비밀번호를 update 문자열로 변경
	// 2 인 경우 setter 메소드를 이용하여 m의 이름을 update 문자열로 변경
	// 3 인 경우 setter 메소드를 이용하여 m의 이메일을 update 문자열로 변경
}

public void deleteMember(String userId) {

	// 매개변수로 전달받은 userId가 mem에 존재하는 경우 해당 회원 삭제 후
	// 다음 인덱스 객체들의 정보를 한 칸씩 앞으로 이동 시킴
	// 실행 시 NullPointerException 발생할 수 있음 -> 왜 그런지 생각해보고 해결하시오

	// memberCount 1 감소

}
```