소프트웨어 공학

(과제3)

학번: b211200 이름: 장현석

학번: b411206 이름: 조현정

학번: b111047 이름: 김윤영

학번: b111250 이름: 홍유남

역할 분담

김윤영- 투표 관리 관련 sequence diagram 수정 및 design class diagram작성 및 implementation

조현정- 회원 관리 관련 sequence diagram 수정 및 design class diagram 작성 및 implementation

장현석- 그룹 관리 관련 sequence diagram 수정 및 design class diagram 작성 및 implementation

홍유남- 투표 관리& 타이머 관련 sequence diagram 수정 및 design class diagram 작성 및 implementation

실제 수행한 내용

김윤영- 타이머, 투표, 투표제안, sequence diagram 수정 및 design class diagram 작 성, implementation, use case diagram 수정. 공용 헤더 파일 생성. 코드 취합, 에러 처리, 테스팅.

조현정- 회원가입, 로그인, 로그아웃, 회원 탈퇴 sequence diagram 수정 및 design class diagram 작성, implementation, 보고서 초안 작성, 수정 보 완. 프로그램 테스팅

장현석- 그룹생성, 전체 그룹조회, 가입 그룹조회, 그룹 탈퇴, 그룹 삭제 sequence diagram 수정 및 design class diagram 작성, 전체적인 코드 틀을 만듦. 코드취합, 에러 처리, 테스팅.

홍유남- 투표 정보 조회, 현재 진행중인, 종료된, 향후진행 예정 투표 리스트 조회 sequence diagram 수정 및 design class diagram 작성, implementa tion, 보고서 수정 및 보완

저희 팀은 2주일 동안 7번의 모임을 가졌으며, 첫 번째 모임에는 과제 2의 올바르지 못한 부분에 대해 어떤 부분으로 수정해 나갈지 토의하였으며 앞으로의 계획을 세우고 역할 분담을 하였습니다.

그 후 각자 맡은 부분의 use case에서 필요한 collection diagram을 생각하였고, 함께 사용되는 class의 attribute와 operation에 대해 논의하였습니다.

세 번째 모임에서는 수정된 sequence diagram과 analysis diagram을 기반으로 desgin class diagram을 작성하였고, 기본적인 헤더 파일을 생성하였습니다.

그 후 모임에서는 계속 각자 구현한 코드를 합치고, 에러를 처리하였으며 모르는 부분에 대해 협력하여 해결하였습니다. 또한 실제 구현하다 보니, 설계할 때와는 다르게 생각되는 부분들에 대해 토의하며 sequence diagram을 수정하였고 그에 따라 implementation하였습니다.

마지막 모임에서는 전체적으로 테스팅 해 보았으며, 보고서를 수정, 보완 하였습니다.

**<1>Requirement list**

**1. functional**

비회원

|  |  |  |
| --- | --- | --- |
| No | Requirement | Use Case |
| 1 | 비회원은 온라인 투표 시스템을 사용하기 위한 권한을 얻기 위해 기본적인 정보(이름, 주민번호, 주소, 이메일 등)와 ID/password를 등록한다. | 회원가입 |

회원

|  |  |  |
| --- | --- | --- |
| No | Requirement | Use Case |
| 1 | 회원은 온라인 투표 시스템을 이용하기 위해 ID와 password를 입력하여 서버에 로그인한다. | 로그인 |
| 2 | 회원이 온라인 투표 시스템을 로그아웃하면 회원가입이나 로그인을 할 수 있다. | 로그아웃 |
| 3 | 회원은 온라인 투표 시스템에서 탈퇴할 수 있다. 탈퇴와 동시에 시스템의 사용권한은 소멸된다. | 회원 탈퇴 |

비그룹원

|  |  |  |
| --- | --- | --- |
| No | Requirement | Use Case |
| 1 | 회원은 전체 그룹 리스트를 조회할 수 있다. | 전체 그룹 조회 |
| 2 | 회원은 자신이 원하는 그룹을 생성하기 위해 그룹 이름을 입력 한 후 그룹을 생성한다. | 그룹 생성 |
| 3 | 회원은 출력된 그룹 리스트를 바탕으로 그룹 가입을 수행할 수 있다. | 그룹 가입 |

그룹원

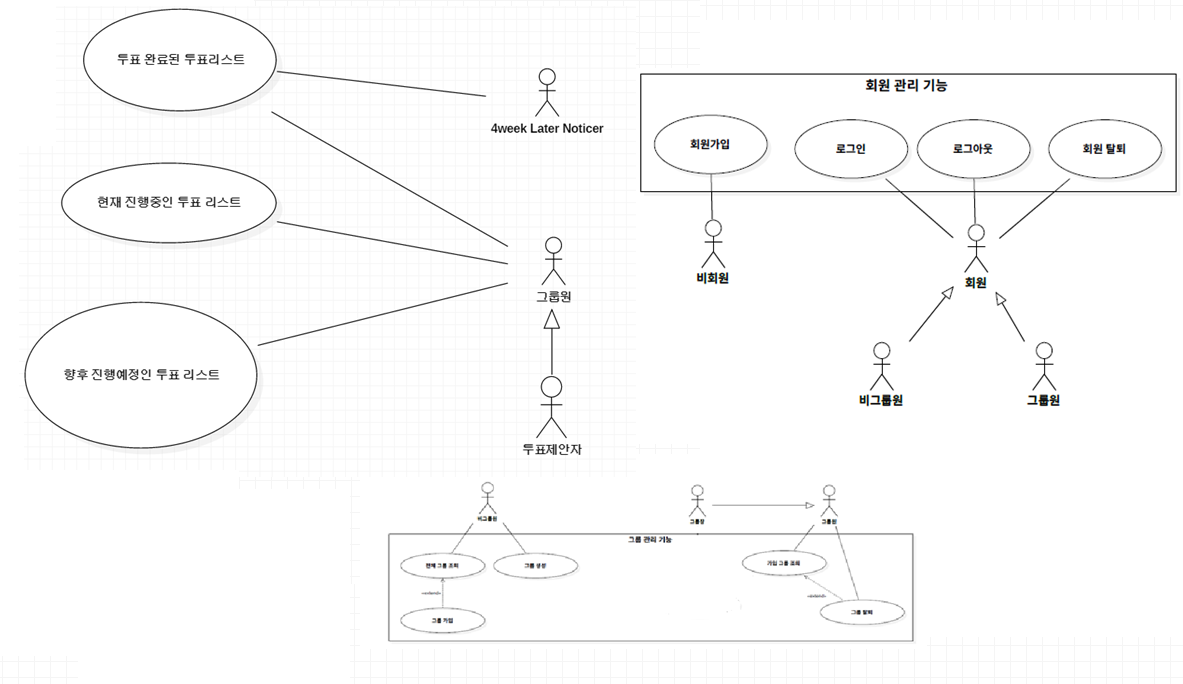
|  |  |  |
| --- | --- | --- |
| No | Requirement | Use Case |
| 1 | 회원은 자신이 가입한 그룹을 조회할 수 있다. | 가입 그룹 조회 기능 |
| 2 | 현재 진행 중인 투표 리스트를 보여준다. 내용은 투표 주제, 투표항목, 남은 시간 이다. | 현재 진행 중인 투표 리스트 |
| 3 | 향후 진행 예정인 투표 리스트를 보여준다. 내용은 투표 주제, 투표항목, 투표 시작 및 마감 시각이다. | 향후 진행 예정인 투표 리스트 |
| 4 | 종료된 투표 리스트를 보여준다. 내용은 투표 주제, 투표 항목, 투표결과이다. 종료된 투표는 4주후에 삭제 된다. | 종료된 투표 리스트 |
| 5 | 현재 진행 중인 투표에 대해서는 투표자가 해당 항목을 클릭함으로써 투표권을 행사 할 수 있다. | 투표 |
| 6 | 투표는 특정 회원의 제안에 의해 생성된다.  투표 제안자는 다음 부분을 입력해야 한다.  (투표 주제, 투표 항목, 투표 시작 및 마감 시각 명시) | 투표 제안 |
| 7 | 회원은 자신이 속한 그룹에서 탈퇴할 수 있다. | 그룹 탈퇴 |

**2.nonfunctional**

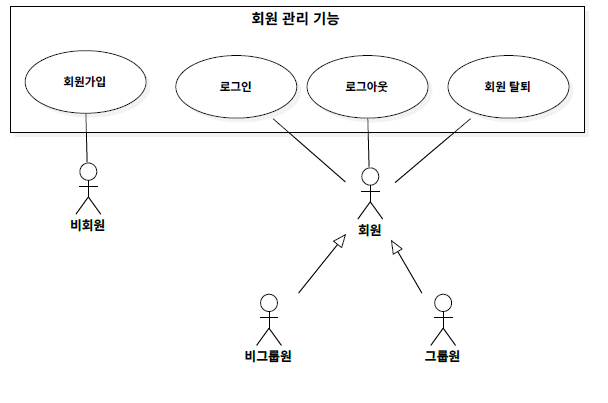
|  |  |  |
| --- | --- | --- |
| No | Requirement | Use Case |
| 1 | 동시 접속자가 몰려도 원활한 처리가 가능해야 한다. | not applicable |
| 2 | 다양한 언어를 지원해야 한다. | not applicable |
| 3 | 시차를 고려하여 투표시간을 동기화 시켜야 한다. | not applicable |
| 4 | 새로운 투표 제안과 그룹 생성 요청을 동시에 원활한 처리가 가능해야 한다. (관리자 입장) | not applicable |
| 5 | 작업들은 빠른 시간 내에 처리되어야 한다. | not applicable |
| 6 | 외부 오류(지진 정전 등)에 의해 요청들이 누락 되는 것을 방지되어야 한다. | not applicable |

**<2>Use Case Diagram**

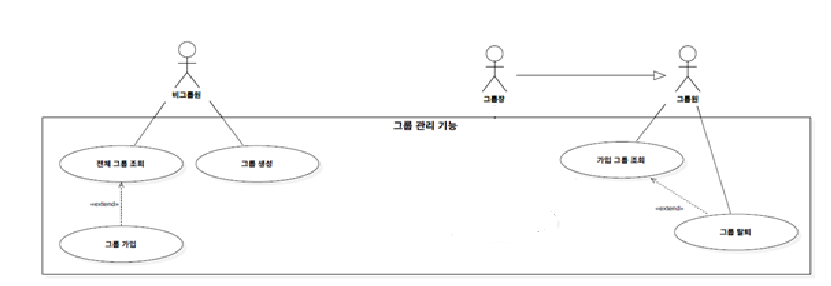
**전체 system Diagram**

****

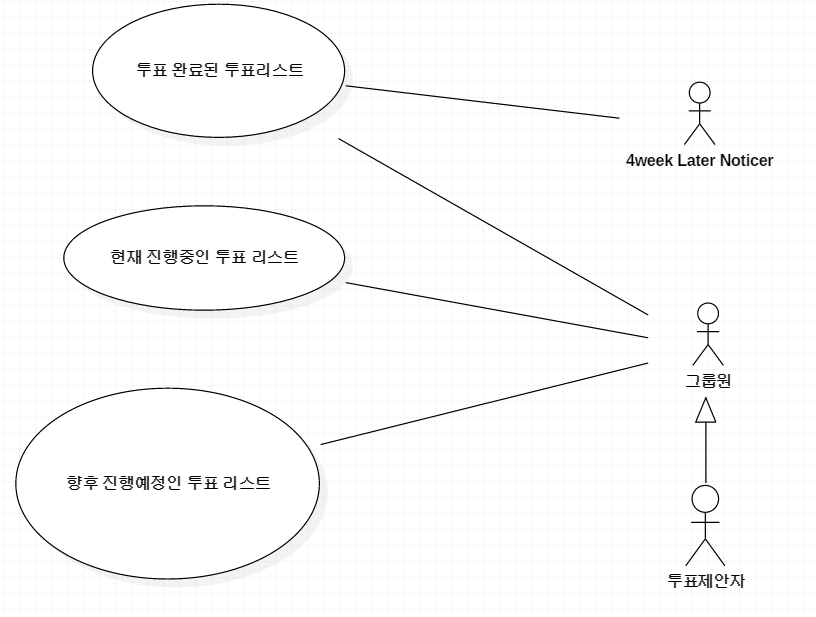
**회원 관리 Diagram**

****

**그룹관리 diagram**

****

**투표 관리diagram**

****

**<3>Use Case Descriptions (step by step breakdown)**

**1.비회원 description**

Use case description: 회원 가입

|  |  |
| --- | --- |
| Actor Action | System Response |
| 2. 회원가입 번호(1 1)과 이름, 주민번호, 주소, ID, 비밀번호를 입력한다. | 1. 입력 받을 창을 출력한다.   3.완료 메시지(1.1회원가입  이름 주민번호 주소 ID 비밀번호)를 출력한다. |
| Alternative  2a 이미 존재하는 ID가 입력 될 경우  2a1 회원 가입이 불가능 하다는 경고 메시지를 출력한다. | |

**2.회원 description**

Use case description: 로그인

|  |  |
| --- | --- |
| Actor Action | System Response |
| 2.로그인 번호(2 1)과 ID, 비밀번호를 입력한다. | 1. 입력 받을 창을 출력한다.   3. 완료 메시지(2.1 로그인  ID 비밀번호)를 출력한다. |
| Alternative  2a ID와 password가 일치 하지 않을 경우  2a1 경고 메시지를 출력한다. | |

Use case description: 로그아웃

|  |  |
| --- | --- |
| Actor Action | System Response |
| 2.로그아웃 번호(2 2)를 입력한다. | 1. 입력 받을 창을 출력한다.   3. 완료 메시지(2.2 로그아웃  ID)를 출력한다. |

Use case description: 회원탈퇴

|  |  |
| --- | --- |
| Actor Action | System Response |
| 2.회원 탈퇴 번호(1 2)를 입력한다. | 1. 입력 받을 창을 출력한다.   3. 완료 메시지(1.2. 회원탈퇴  ID)를 출력한다. |
| Alternative  2a 그룹장이 회원 탈퇴를 시도할 경우  2a1 그룹장은 회원 탈퇴가 불가능하다는 경고 메시지를 출력한다. | |

**3. 비그룹원 description**

Use case description: 그룹 생성

|  |  |
| --- | --- |
| Actor Action | System Response |
| 2.그룹 생성 번호(5 3)과 그룹 이름을 입력한다. | 1.입력 받을 창을 출력한다.  3. 완료 메시지(5.3. 그룹생성  그룹이름)을 출력한다. |
| Alternative  2a 이미 존재하는 그룹 이름이 입력 될 경우  2a1 그룹 생성이 불가능 하다는 경고 메시지를 출력한다. | |

Use case description: 전체 그룹 조회

|  |  |
| --- | --- |
| Actor Action | System Response |
| 2. 전체 그룹 조회 번호(5 1)을 입력한다.  4.그룹 가입을 선택한다.  4a 그룹 가입 번호(5 2)와 그룹이름을 입력한다.  4b None. | 1. 입력받을 창을 출력한다.   3. 완료 메시지(5.1. 전체그룹조회  그룹이름들)을 출력한다.  5. 해당하는 메시지를 출력한다.  5a 완료 메시지(5.2 그룹가입  그룹이름)을 출력한다.  5b None. |

**4. 그룹원 description**

Use case description: 가입 그룹 조회

|  |  |
| --- | --- |
| Actor Action | System Response |
| 2.가입 그룹 조회 번호(5 4)를 입력한다. | 1. 입력받을 창을 출력한다.   3.완료 메시지(5.4. 가입그룹 조회  그룹이름)을 출력한다. |
| Extension  After step 2. 일반 그룹원일 경우, 그룹 탈퇴가 가능 하다.  Alternative  2a 비그룹원이 가입그룹 조회를 시도할 경우  2a1 가입된 그룹이 없다는 메시지를 출력한다. | |

Use case description: 투표 제안

|  |  |
| --- | --- |
| Actor Action | System Response |
| 2.투표 제안 번호(3 1) 그리고 투표 주제, 항목수, 투표시작시간, 투표마감시간을 입력한다. | 1. 입력받을 창을 출력한다.   3. 완료 메시지(3.1. 투표 제안  투표주제, 항목수, 투표시작시간, 투표마감시간)를 출력한다. |
| Alternative  2a 투표 시작시간이 현재시간으로부터 다섯시간 이내인 경우  2a1 5시간 이후로 투표를 제안해야 한다는 메시지를 출력한다.  2b 투표 시작시간이 현재 시간보다 과거인 경우  2b1 투표를 제안할 수 없다는 메시지를 출력한다. | |

Use case description: 현재 진행 중인 투표 리스트

|  |  |
| --- | --- |
| Actor Action | System Response |
| 2. 현재 진행 중인 투표 리스트 번호(4 1)을 입력한다. | 1. 입력받을 창을 출력한다.   3. 완료 메시지(4.1. 현재 진행 중인 투표 리스트  투표주제 항목수 남은시간 들)을 출력한다. |
| Extension  After step 3. 투표를 할 수 있다. | |

Use case description: 향후 진행 예정인 투표 리스트

|  |  |
| --- | --- |
| Actor Action | System Response |
| 2. 향후 진행 예정인 투표 리스트 번호(4 3)을 입력한다. | 1. 입력받을 창을 출력한다.   3. 완료 메시지(4.3. 향후 진행 예정인 투표 리스트  투표주제 항목수 시작시간 종료시간 들)을 출력한다. |

Use case description: 종료된 투표 리스트

|  |  |
| --- | --- |
| Actor Action | System Response |
| 2. 종료된 투표 리스트 번호(4 4)를 입력한다. | 1. 입력받을 창을 출력한다.   3. 종료 된지 4주가 지난 투표를 삭제하고 완료 메시지(4.4. 종료된 투표 리스트  투표주제 투표항목:투표수 들)를 출력한다. |

Use case description: 투표

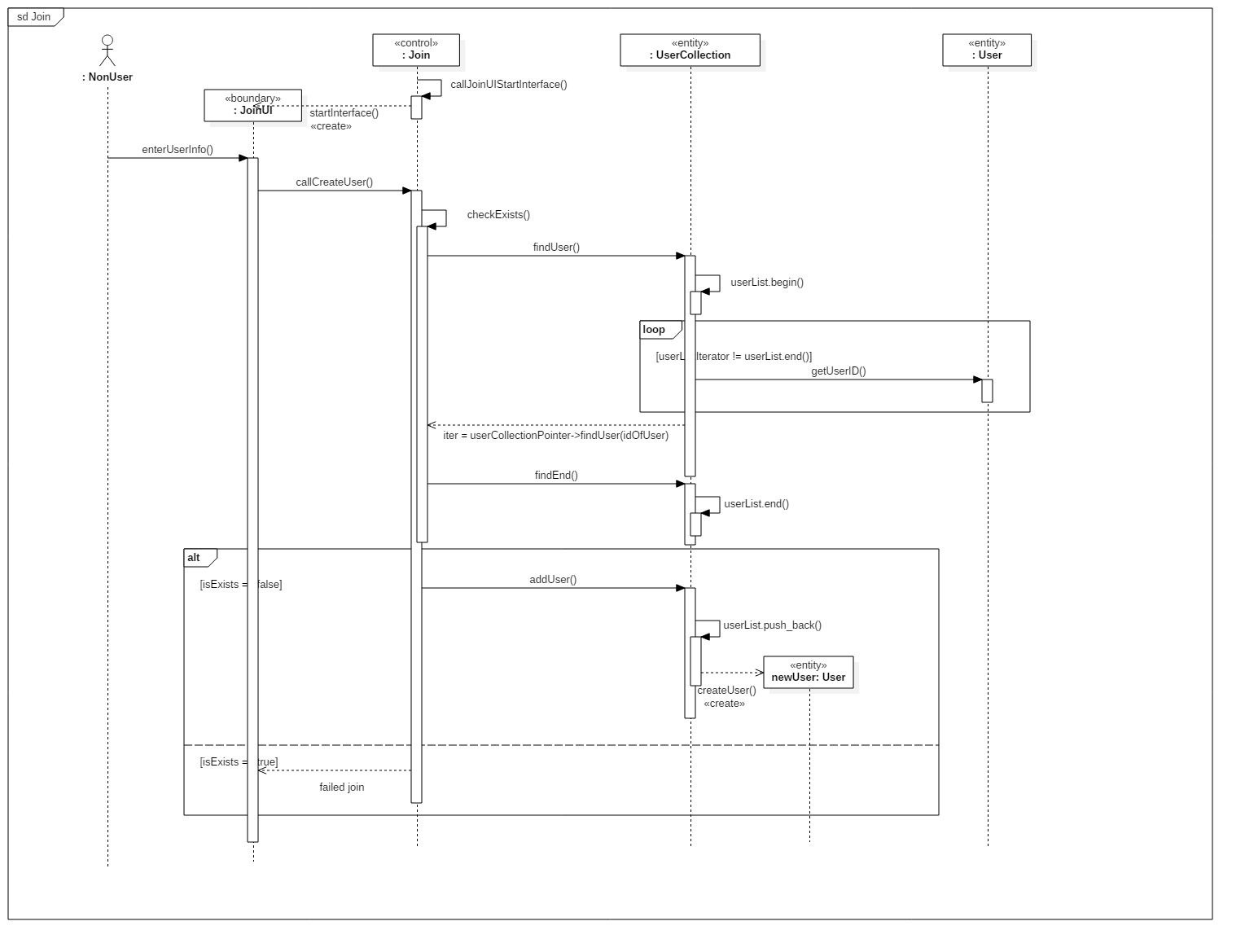
|  |  |
| --- | --- |
| Actor Action | System Response |
| 2.투표 번호(4 2) 그리고 투표주제 항목번호를 입력한다. | 1. 입력받을 창을 출력한다.   3. 완료 메시지(4.2. 투표  투표주제 항목번호)를 출력한다. |
| Alternative  2a 투표 주제 또는 항목 번호가 존재 하지 않을 경우  2a1 접근 불가능하다는 경고 메시지를 출력한다.  2b 같은 투표 주제에 대해 이미 투표한 경우  2b1 투표가 불가능 하다는 경고 메시지을 출력한다. | |

Use case description: 그룹 탈퇴

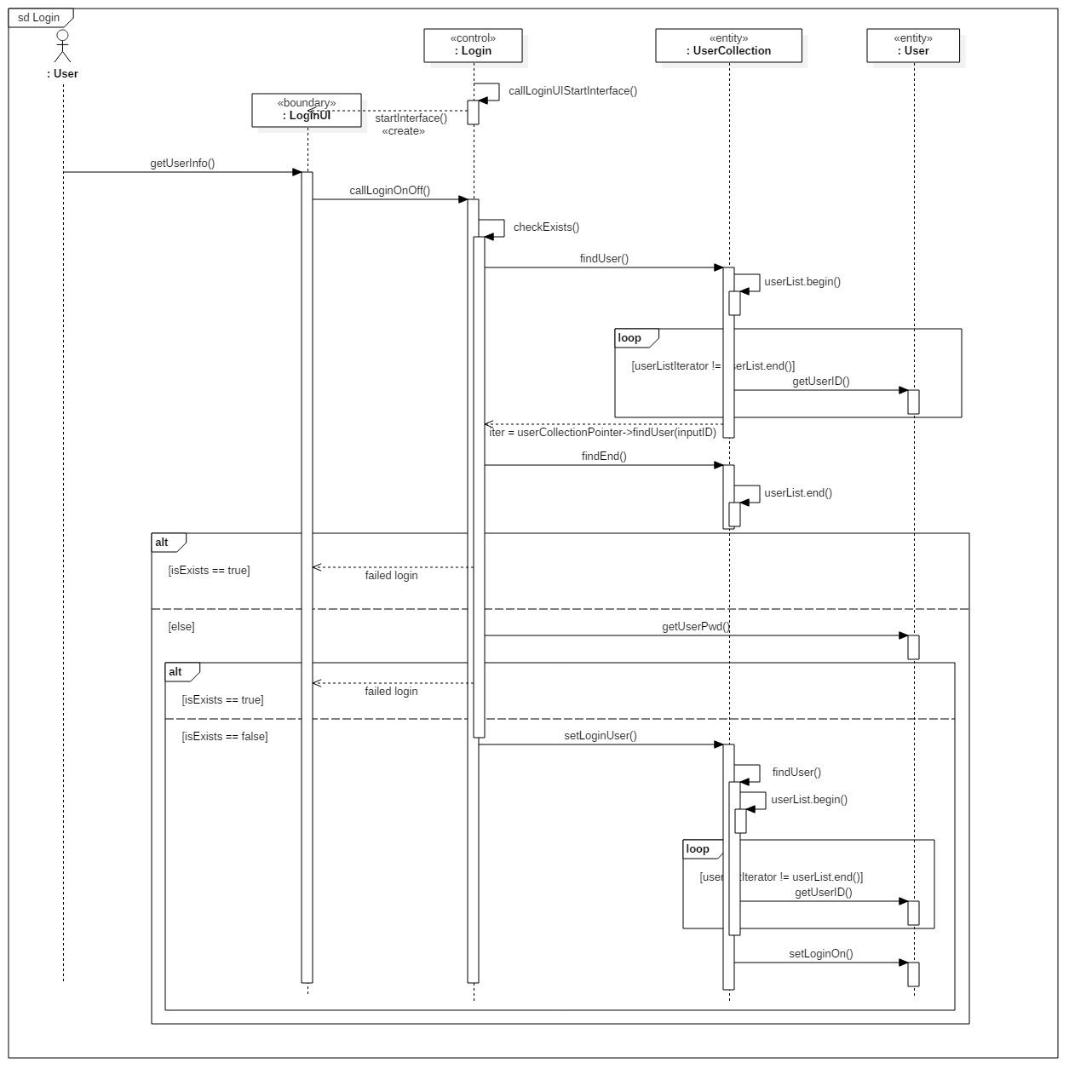
|  |  |
| --- | --- |
| Actor Action | System Response |
| 2. 그룹 탈퇴 번호(5 5)를 입력한다. | 1. 입력받을 창을 출력한다.   3. 완료 메시지(5.5 그룹 탈퇴  그룹 이름)을 출력한다. |
| Alternative  2a 그룹장이 그룹 탈퇴를 시도할 경우  2a1 그룹장은 그룹 탈퇴가 불가능하다는 경고 메시지를 출력한다. | |

**<4>Sequence Diagram**

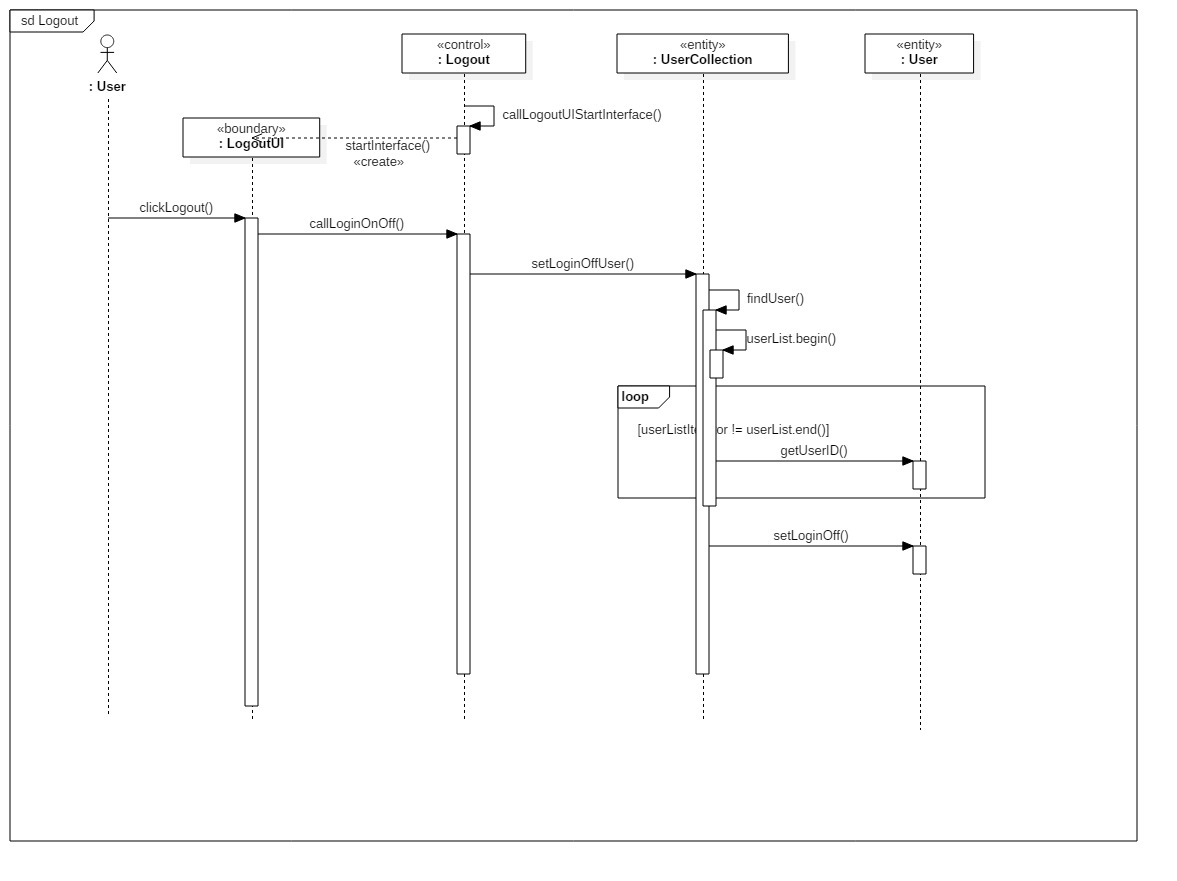
**1.회원가입**

****

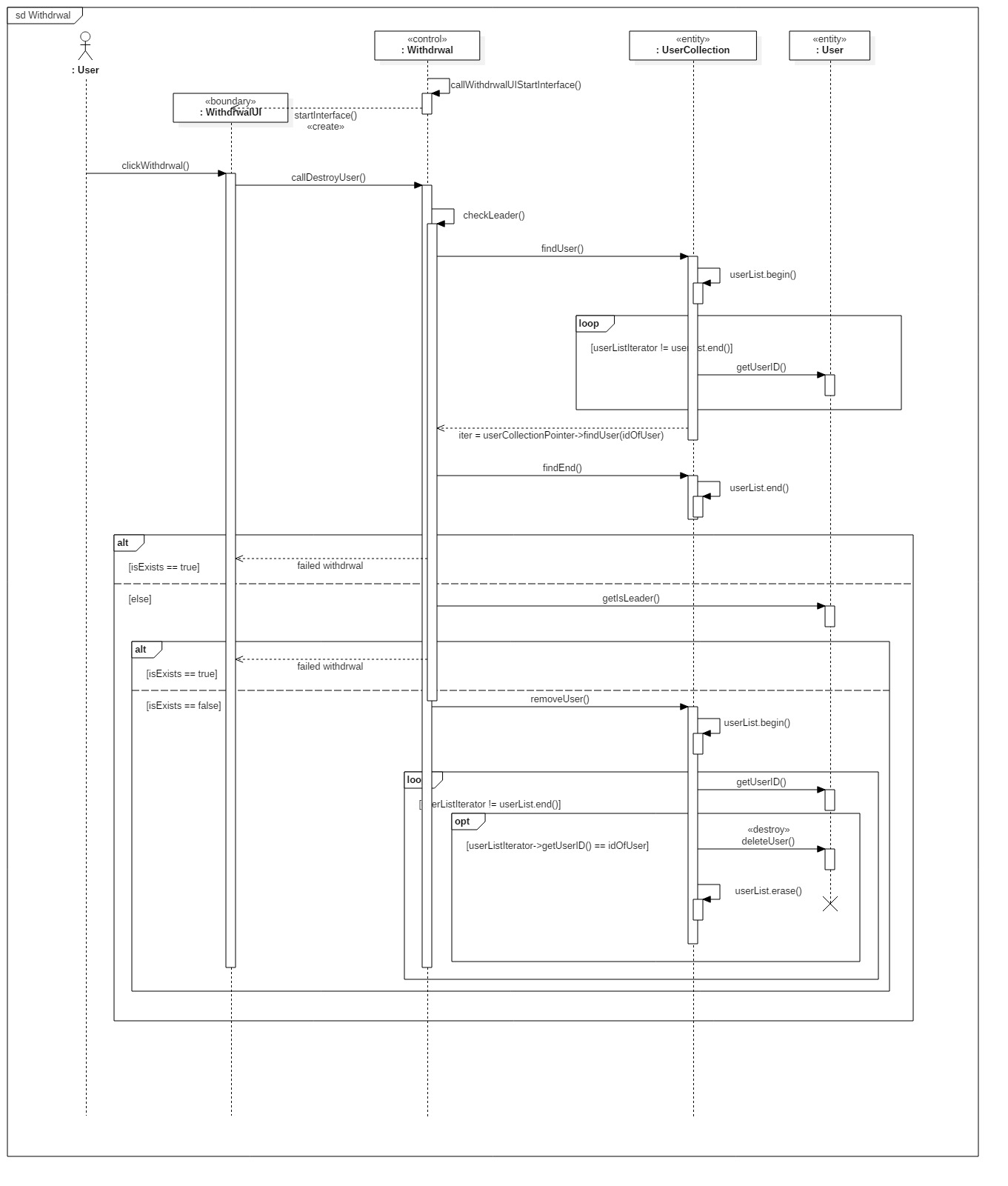
**2.로그인**

****

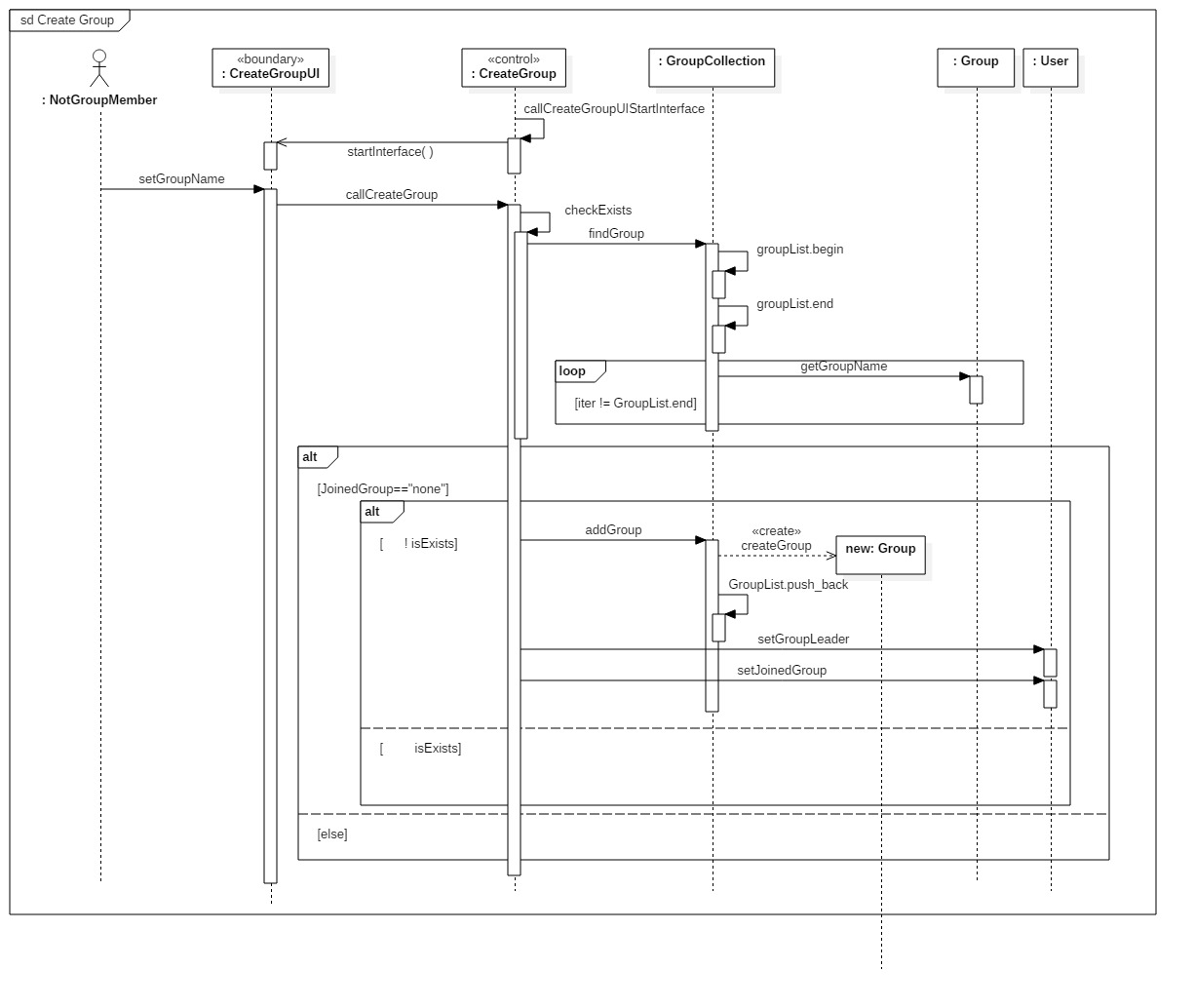
**3.로그아웃**

****

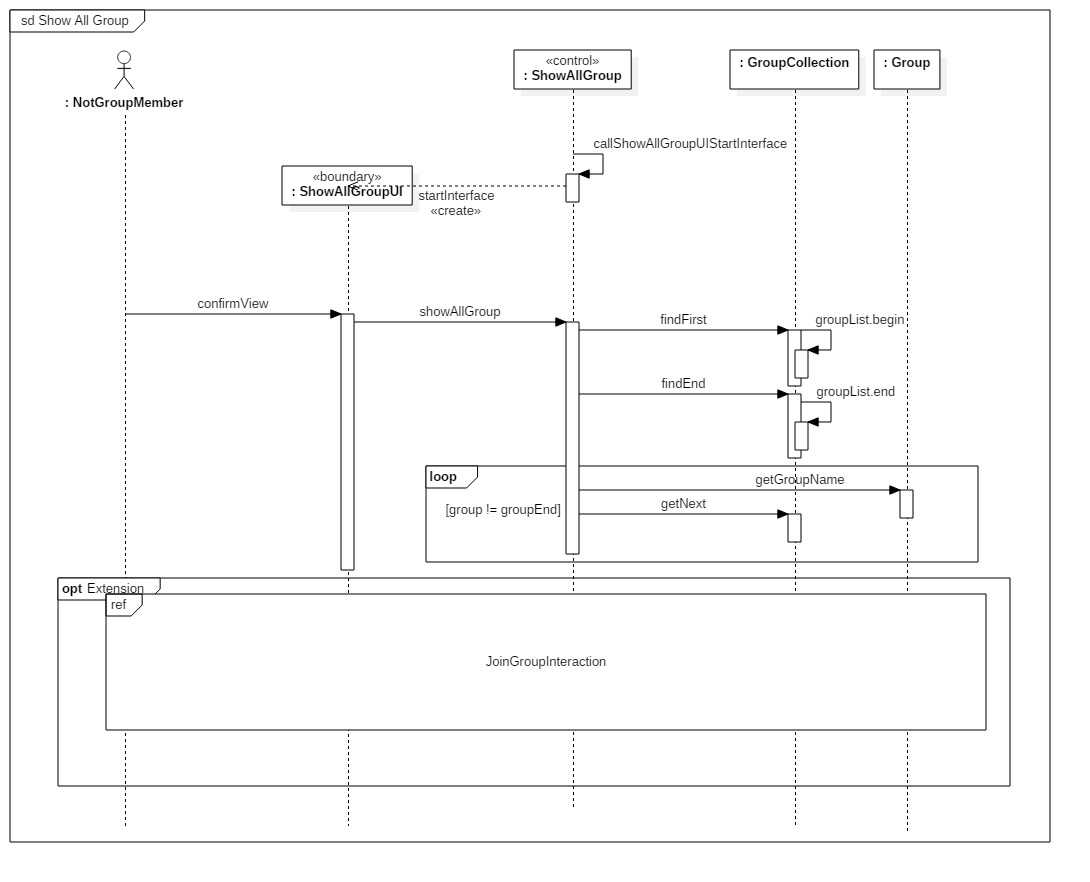
**4.회원탈퇴**

****

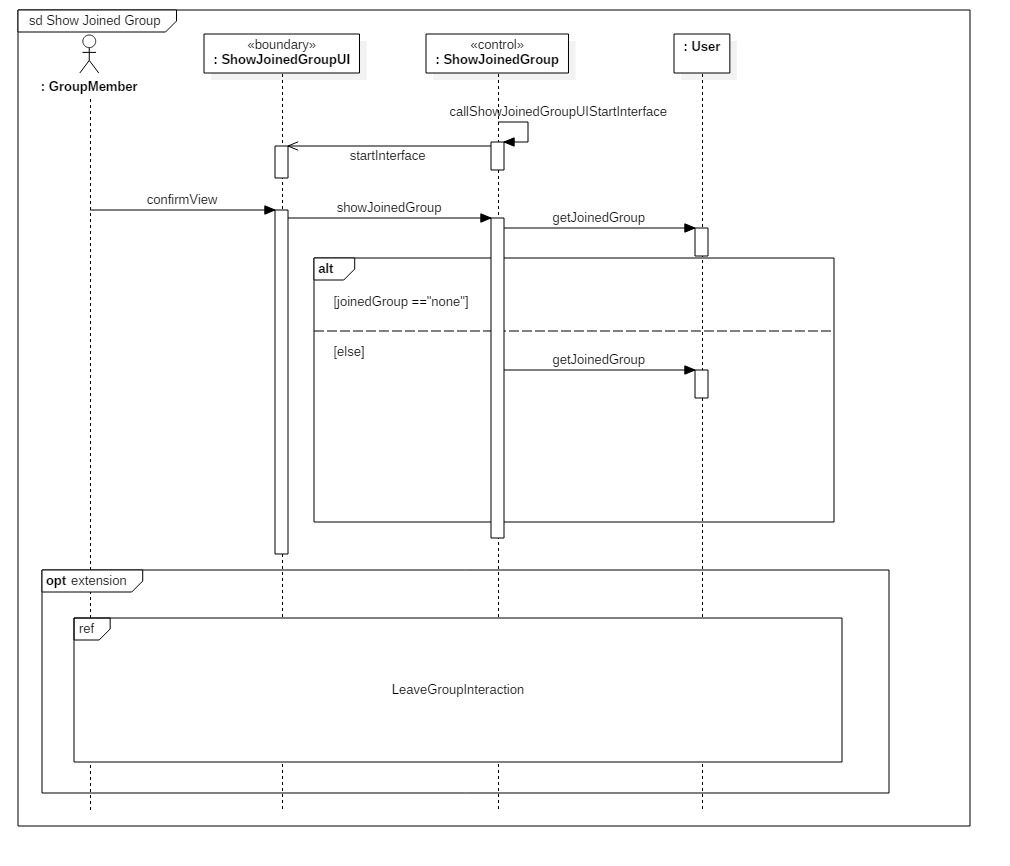
**5.그룹 생성**

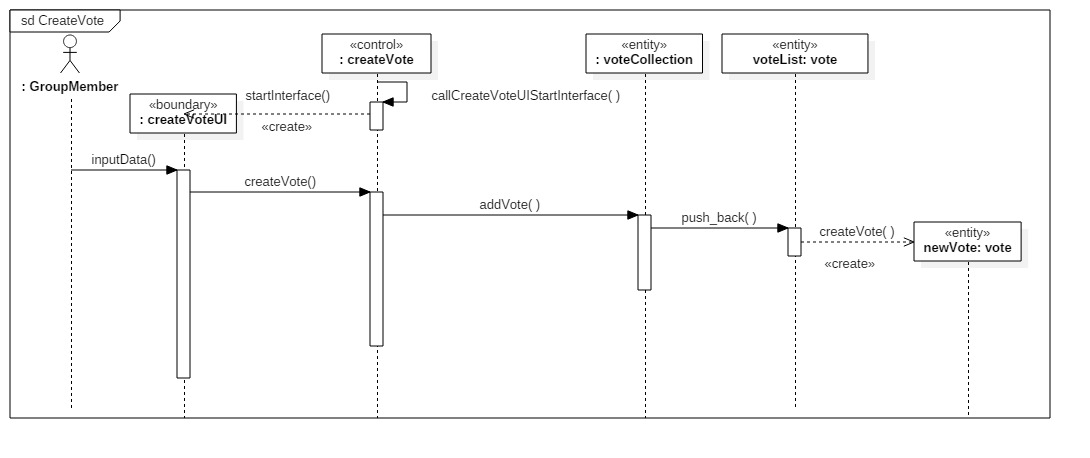
****

**6.전체 그룹 조회**

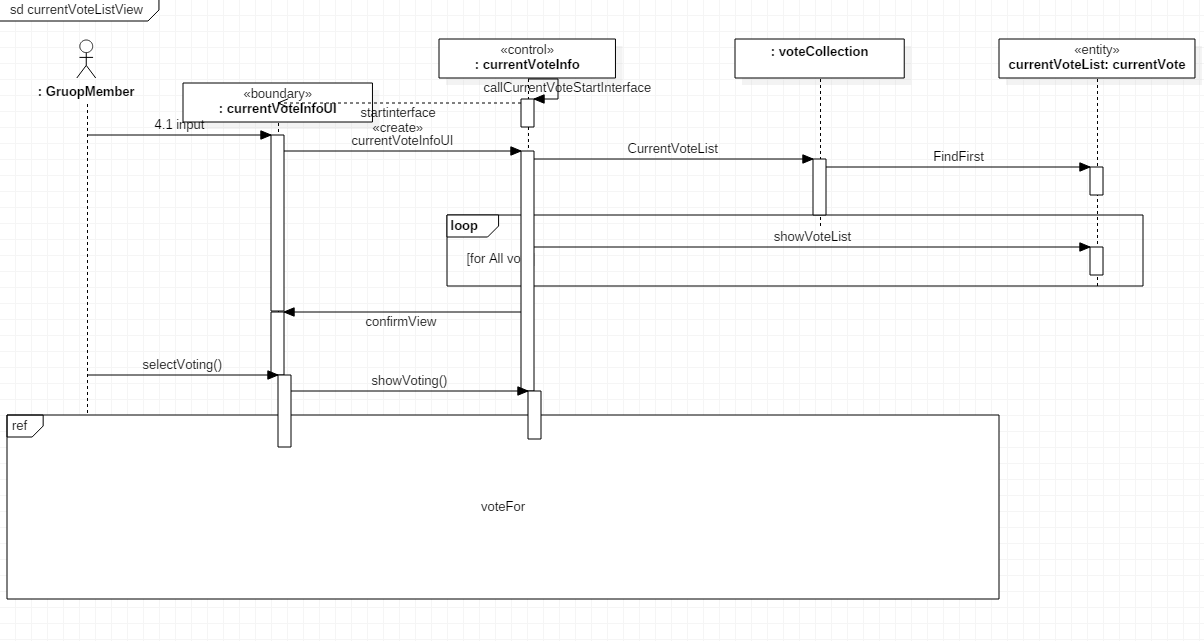
****

**7.가입 그룹 조회**

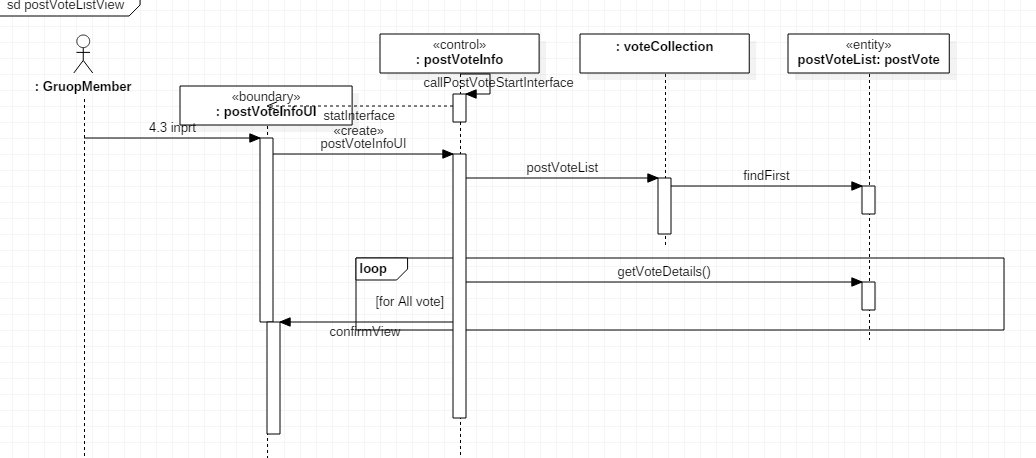
****

**8.투표 제안**

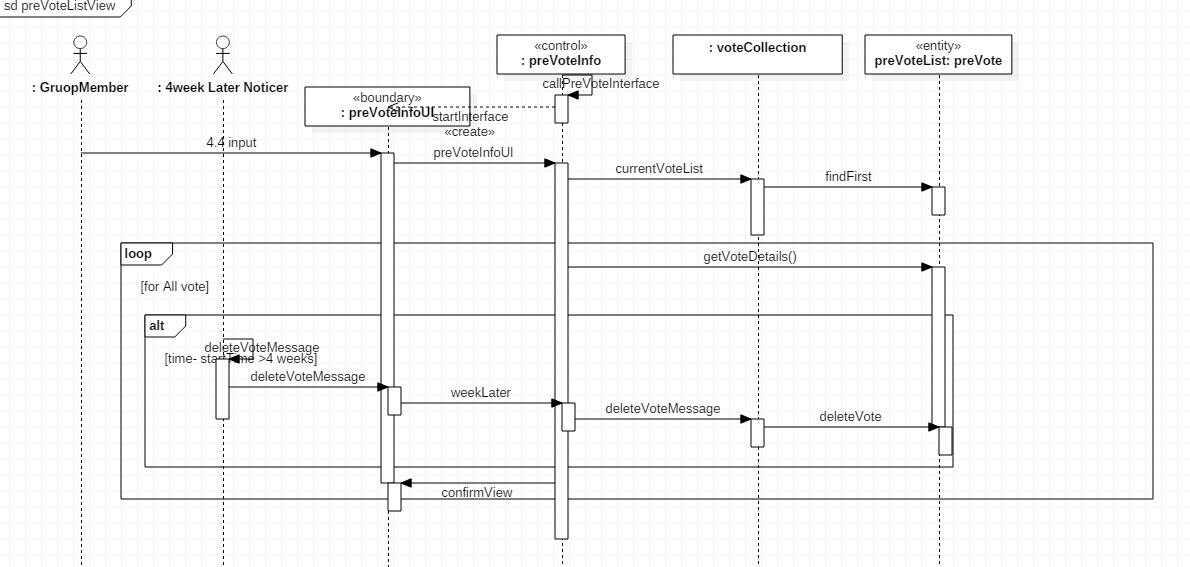
**9. 현재 진행중인 투표 리스트**

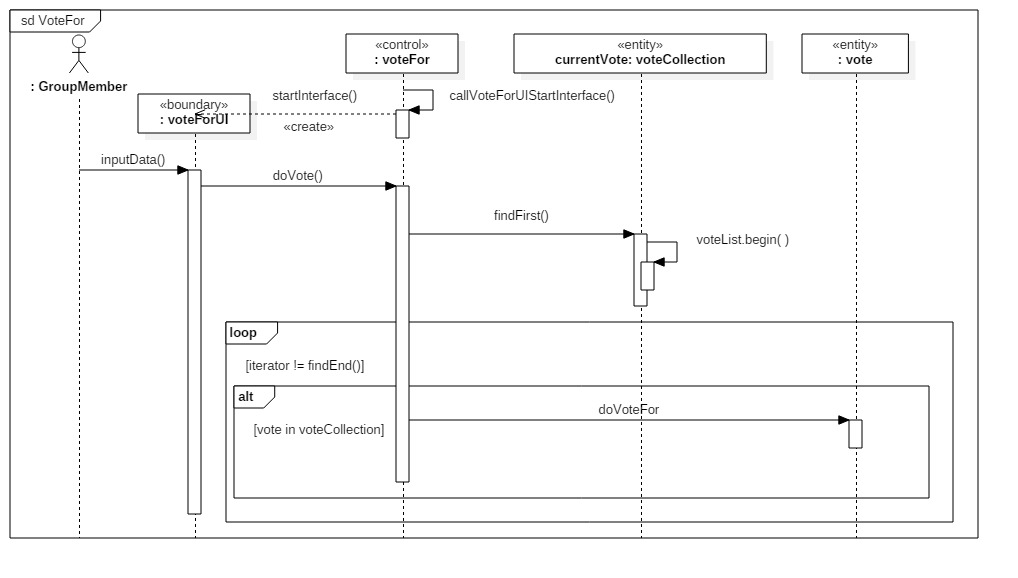
****

**10.향후 진행 예정인 투표 리스트**

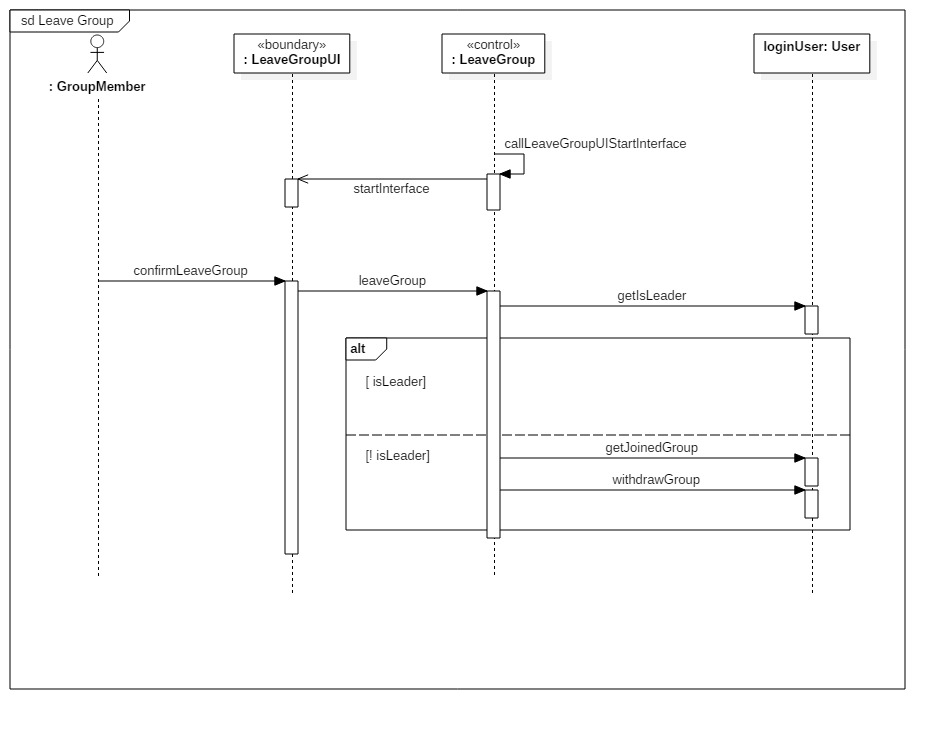
****

**11.종료된 투표 리스트**

****

**12.투표**

**13.그룹탈퇴**

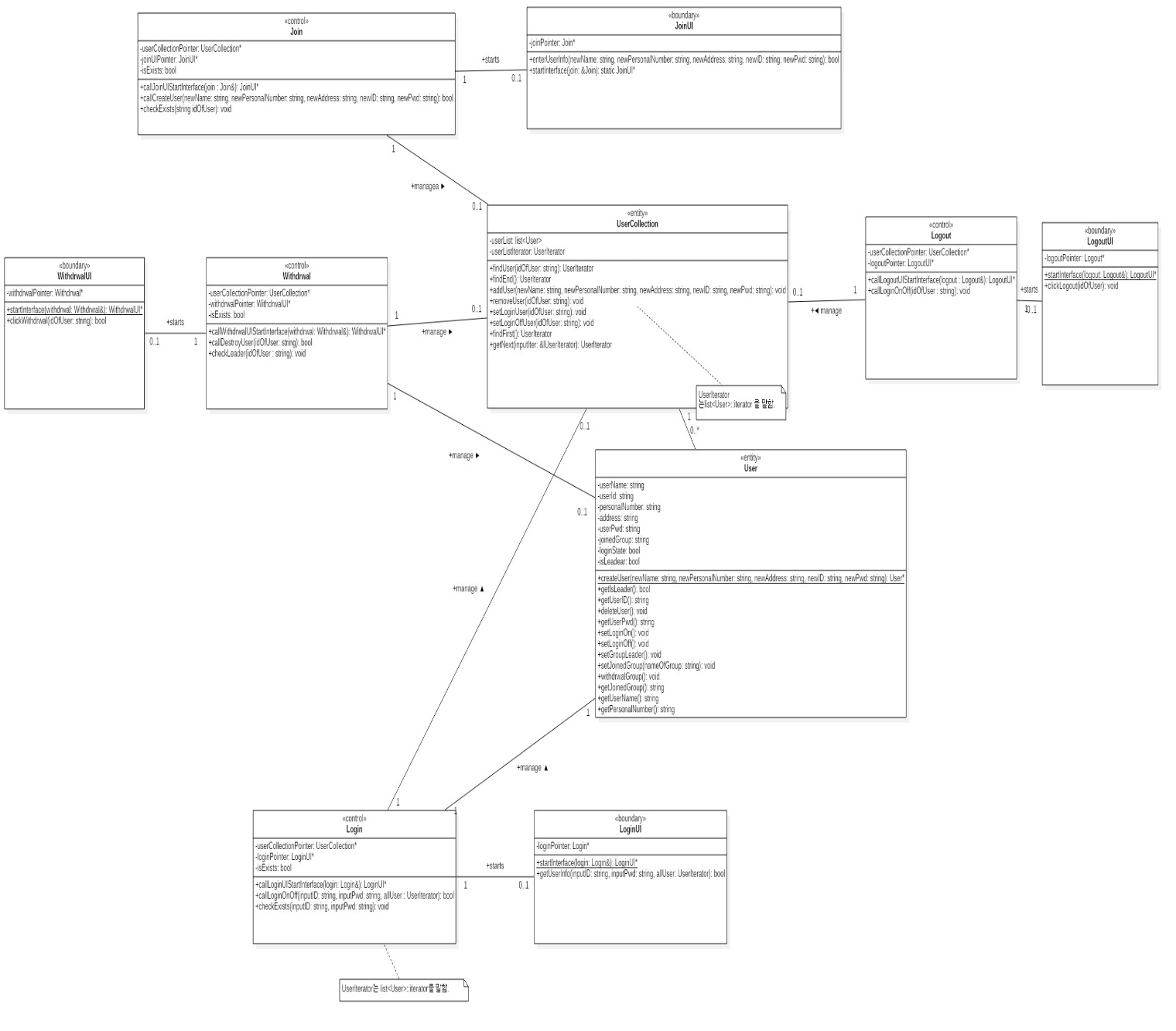
****

**<5>Design Class Diagram**

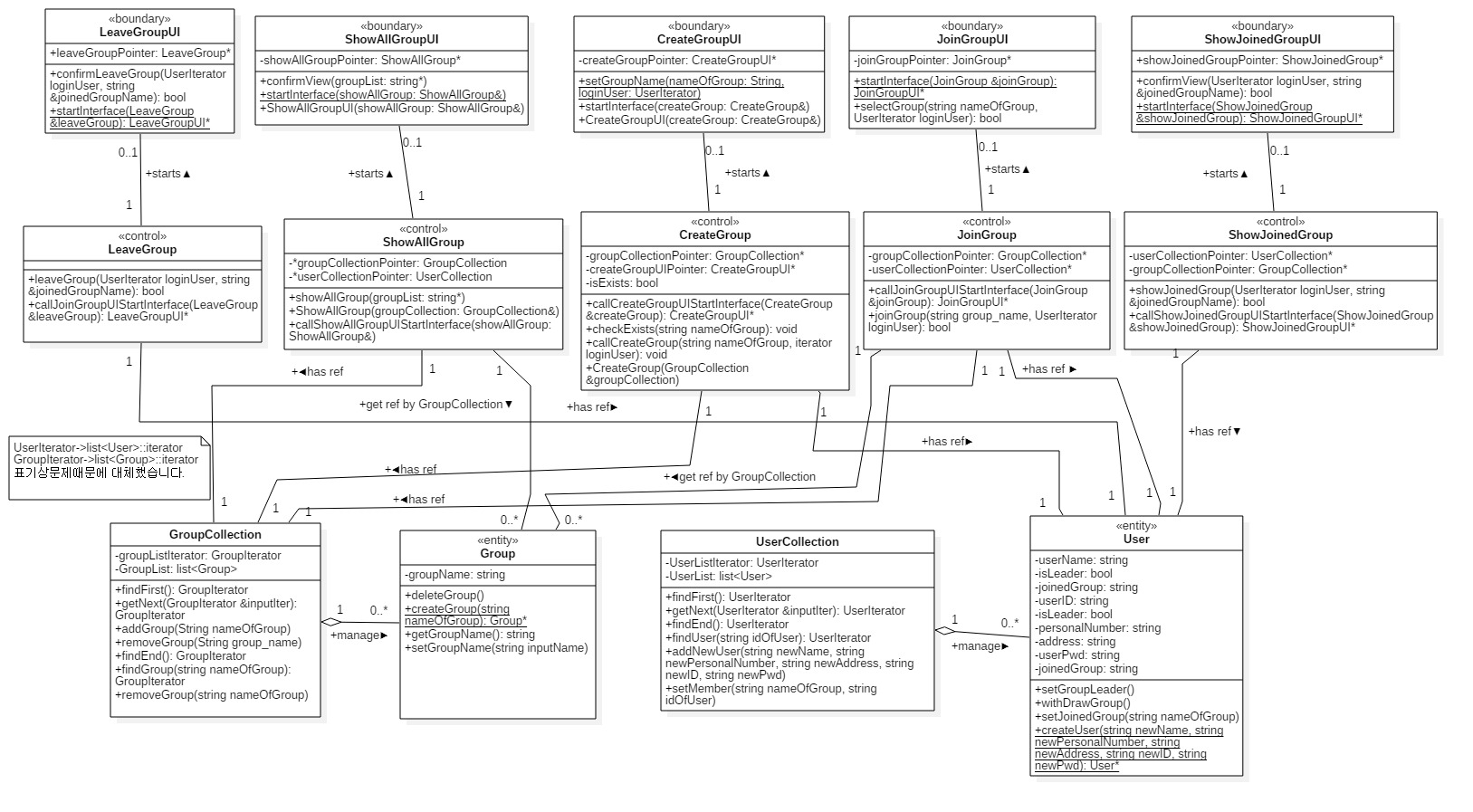
1. 4개의 subSystem

(투표 관리를 2명에서 나눠서 했기 때문에, subSystem도 1,2로 표시했습니다.)

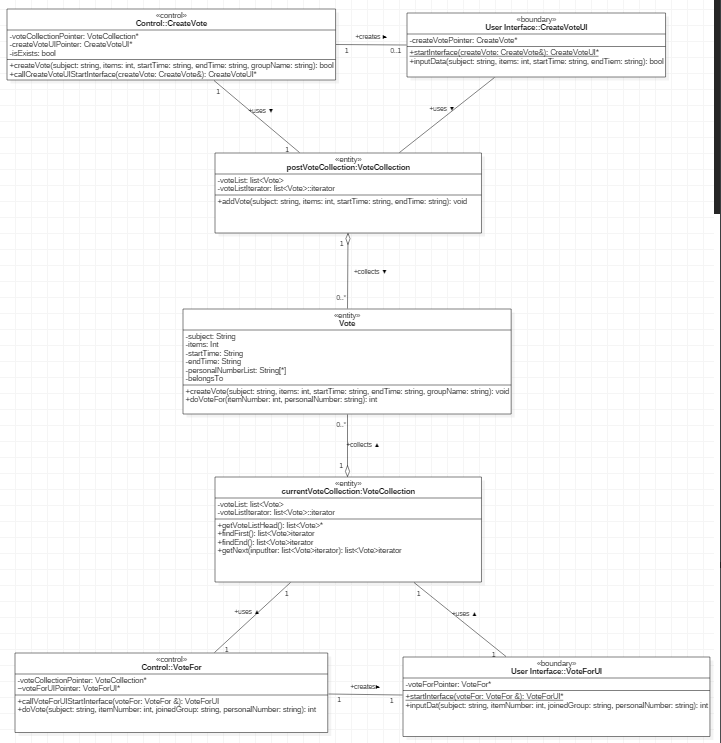
1. **회원 관리 subSystem**

****

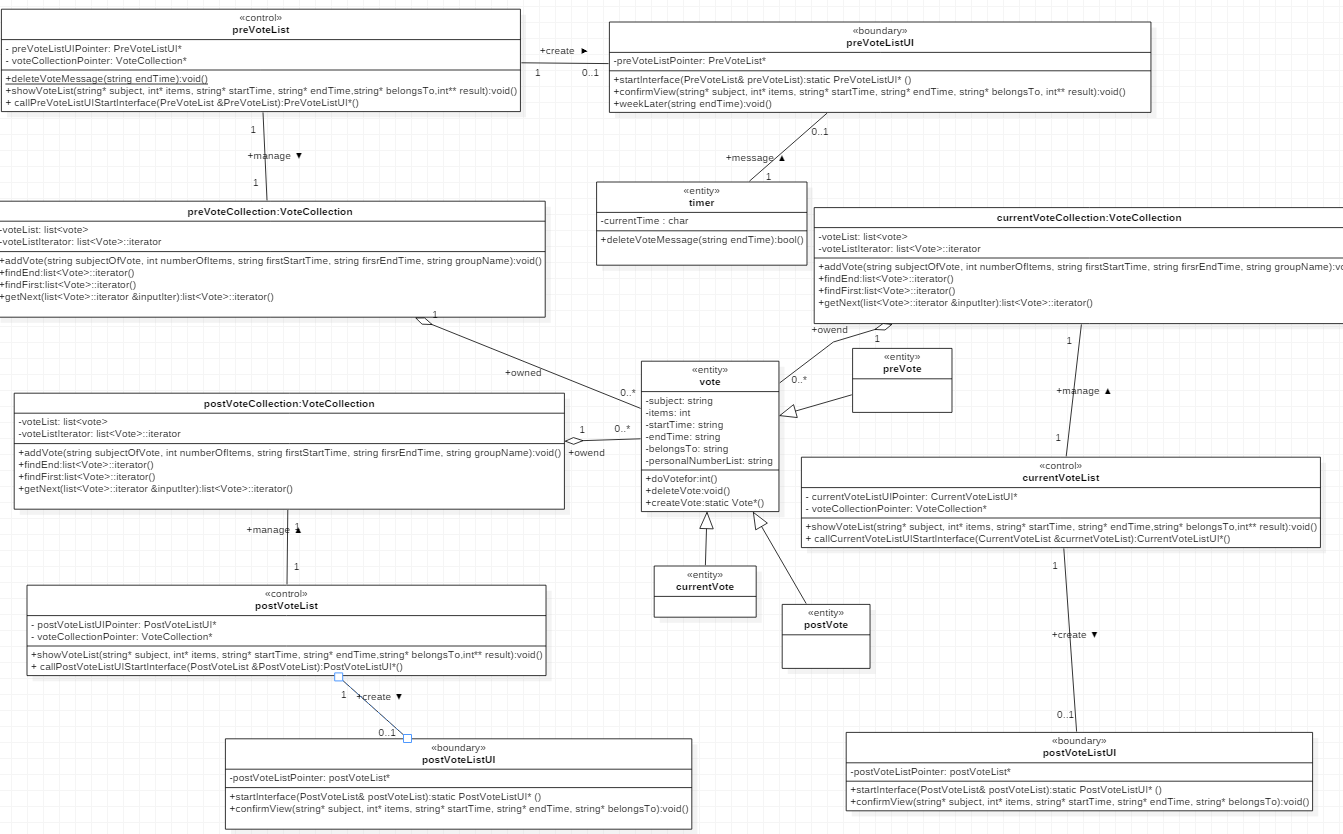
**2. 그룹 관리 subSystem**



**3. 투표 관리 subSystem1**



**4. 투표 관리 subSystem2**

****

**<6>Source Code**

System.h

#pragma once

//Class: 시스템 헤더

//Description: 메인에 쓰이는 인클루드와 전역변수를 넣습니다

//created: 2017/5/28 01:10am

//Author: 장현석

//mail: jang8018@naver.com

#include "Group.h"

#include "GroupCollection.h"

#include "ShowAllGroup.h"

#include "ShowAllGroupUI.h"

#include "UserCollection.h"

#include "User.h"

#include "CreateGroup.h"

#include "CreateGroupUI.h"

#include "LeaveGroup.h"

#include "LeaveGroupUI.h"

#include "ShowJoinedGroup.h"

#include "ShowJoinedGroupUI.h"

#include "VoteCollection.h"

#include "Vote.h"

#include "JoinUI.h"

#include "Join.h"

#include "JoinGroup.h"

#include "JoinGroupUI.h"

#include "CreateVoteUI.h"

#include "CreateVote.h"

#include"VoteFor.h"

#include"VoteForUI.h"

#include "Login.h"

#include "LoginUI.h"

#include "Logout.h"

#include "LogoutUI.h"

#include "Withdrwal.h"

#include "WithdrwalUI.h"

#include "CurrentVoteListUI.h"

#include "CurrentVoteList.h"

#include "PreVoteListUI.h"

#include "PreVoteList.h"

#include "PostVoteList.h"

#include "PostVoteListUI.h"

#include "ControlAndCollection.h"

#include "Timer.h"

#define INPUT\_FILE\_NAME "input.txt"

#define OUTPUT\_FILE\_NAME "output.txt"

#define MAX\_STRING 50

//생성

Timer timer;

Main.cpp

#include"System.h"

#pragma warning(disable:4996)

using namespace std;

void doTask();

void makeAccount(); //1.1 회원가입

void deleteAccount(); //1.2 회원탈퇴

void logIn(); //2.1 로그인

void logOut(); //2.2 로그아웃

void makeVote(); //3.1 투표제안

void showCurrentVoteList(); //4.1 현재투표리스트 조회

void voting(); //4.2 투표

void showPostVoteList(); //4.3 향후투표예정인 투표리스트 조회

void showPreVoteList(); //4.4 종료된 투표리스트 조회

void showEntireGroup(); //5.1 전체그룹 조회

void joinGroupMember(); //5.2 그룹가입

void makeGroup(); //5.3 그룹생성

void show\_JoinedGroup(); //5.4 가입그룹 조회

void leave\_Group(); //5.5 그룹탈퇴

void changeCurrnetTime(); //6.1 현재시간 설정

void changeSession(); //7.1 Session 변경

void guestSession(); //7.2 guest Session으로 변경

void program\_Exit(); //8.1 종료

FILE\* in\_fp, \*out\_fp;

User guest("guest", "guest", "guest", "guest", "guest");

list<User> forGuest;

list<User>::iterator currentLoginUser;

int main(void) {

in\_fp = fopen(INPUT\_FILE\_NAME, "r+");

out\_fp = fopen(OUTPUT\_FILE\_NAME, "w+"); ;

ControlAndCollection::GetInstance(); //컨트롤과 콜렉션을 생성합니다.

forGuest.push\_back(guest);

currentLoginUser = forGuest.begin();

doTask();

return 0;

}

void doTask()

{

// 메뉴 파싱을 위한 level 구분을 위한 변수

int menu\_level\_1 = 0, menu\_level\_2 = 0;

// 종료 메뉴(8.1)가 입력되기 전까지 반복함

while (menu\_level\_1 != 8 || menu\_level\_2 != 1)

{

// 입력파일에서 메뉴 숫자 2개를 읽기

fscanf(in\_fp,"%d %d", &menu\_level\_1, &menu\_level\_2);

// 메뉴 구분 및 해당 연산 수행

switch (menu\_level\_1)

{

case 1:

{

switch (menu\_level\_2)

{

case 1: // "1.1. 회원가입“ 메뉴 부분

{

makeAccount();

break;

}

case 2: // "1.2. 회원탈퇴“ 메뉴 부분

{

deleteAccount();

break;

}

}

break;

}

case 2:

{

switch (menu\_level\_2)

{

case 1: // "2.1. 로그인“ 메뉴 부분

{

logIn();

break;

}

case 2: // "2.2. 로그아웃“ 메뉴 부분

{

logOut();

break;

}

}

break;

}

case 3:

{

if (currentLoginUser->getJoinedGroup() == "none") {

fprintf(out\_fp, "가입된 그룹이 없습니다.\n");

break;

}

switch (menu\_level\_2)

{

case 1: // "3.1. 투표 제안“ 메뉴 부분

{

makeVote();

break;

}

}

break;

}

case 4:

{

if (currentLoginUser->getJoinedGroup() == "none") {

fprintf(out\_fp, "가입된 그룹이 없습니다.\n");

break;

}

switch (menu\_level\_2)

{

case 1: // "4.1. 현재 진행 중인 투표 리스트“ 메뉴 부분

{

showCurrentVoteList();

break;

}

case 2: // "4.2. 투표“ 메뉴 부분

{

voting();

break;

}

case 3: // "4.3. 향후 진행 예정인 투표 리스트“ 메뉴 부분

{

showPostVoteList();

break;

}

case 4:

{

showPreVoteList();

break;

}

}

break;

}

case 5:

{

switch (menu\_level\_2)

{

case 1: // "5.1. 전체 그룹 조회“ 메뉴 부분

{

showEntireGroup();

break;

}

case 2: // "5.2. 그룹 가입“ 메뉴 부분

{

joinGroupMember();

break;

}

case 3: // "5.3. 그룹 생성“ 메뉴 부분

{

makeGroup();

break;

}

case 4: // "5.4. 가입 그룹 조회“ 메뉴 부분

{

show\_JoinedGroup();

break;

}

case 5: // "5.5. 그룹 탈퇴“ 메뉴 부분

{

leave\_Group();

break;

}

}

break;

}

case 6:

{

switch (menu\_level\_2)

{

case 1: // "6.1 현재시간 설정“ 메뉴 부분

{

changeCurrnetTime();

break;

}

}

break;

}

case 7:

{

switch (menu\_level\_2)

{

case 1: // "7.1. session 변경“ 메뉴 부분

{

changeSession();

break;

}

case 2: // "7.2. guest sessiond으로 변경“ 메뉴 부분

{

guestSession();

break;

}

}

break;

}

case 8:

{

switch (menu\_level\_2)

{

case 1: // "8.1. 종료“ 메뉴 부분

{

program\_Exit();

break;

}

}

}

}

fprintf(out\_fp, "\n");

}

}

//1.1 회원가입

// Function : void makeAccount()

// Description: 회원 가입 하는 함수

// Parameters : void

// Return Value : void

//

// Created: 2017/5/29 8:30 pm

// Author: 조현정

//

void makeAccount() {

Join \*join = ControlAndCollection::GetInstance()->getJoin();

JoinUI\* joinUI = ControlAndCollection::GetInstance()->getJoin()->callJoinUIStartInterface(\*ControlAndCollection::GetInstance()->getJoin());

char name[MAX\_STRING], SSN[MAX\_STRING],

address[MAX\_STRING], ID[MAX\_STRING], password[MAX\_STRING];

bool error;

// 입력 형식 : 이름, 주민번호, 주소, ID, 비밀번호를 파일로부터 읽음

fscanf(in\_fp, "%s %s %s %s %s\n", name, SSN, address, ID, password);

error = joinUI->enterUserInfo(name, SSN, address, ID, password);

if (error)

fprintf(out\_fp, "1.1. Error: 이미 존재하는 ID 입니다.\n");

else

fprintf(out\_fp, "1.1. 회원가입 \n> %s %s %s %s %s \n", name, SSN, address, ID, password);

}

//1.2 회원탈퇴

// Function : void deleteAccount()

// Description: 회원 탈퇴 하는 함수

// Parameters : void

// Return Value : void

//

// Created: 2017/5/30 5:08 pm

// Author: 조현정

//

void deleteAccount() {

Withdrwal \*withdrwal = ControlAndCollection::GetInstance()->getWithdrwal();

WithdrwalUI \*withdrwalUI = withdrwal->callWithdrwalUIStartInterface(\*withdrwal);

char ID[MAX\_STRING];

bool error;

strcpy(ID, currentLoginUser->getUserID().c\_str());

error = withdrwalUI->clickWithdrwal(ID);

if (error)

fprintf(out\_fp, "1.2. Error: 탈퇴할 수 없는 회원입니다.\n");

else

{

currentLoginUser = forGuest.begin();

fprintf(out\_fp, "1.2. 회원탈퇴\n>%s \n", ID);

}

}

//2.1 로그인

// Function : void logIn()

// Description: 로그인 하는 함수

// Parameters : void

// Return Value : void

//

// Created: 2017/5/30 2:30 am

// Author: 조현정

//

void logIn() {

Login \*login = ControlAndCollection::GetInstance()->getLogin();

UserCollection \*userCollection = ControlAndCollection::GetInstance()->getUserCollection();

LoginUI\* loginUI = login->callLoginUIStartInterface(\*login);

char ID[MAX\_STRING], Pwd[MAX\_STRING];

bool error;

fscanf(in\_fp, "%s %s", ID, Pwd);

error = loginUI->getUserInfo(ID, Pwd, currentLoginUser);

if (error)

fprintf(out\_fp, "2.1. Error: 일치하는 정보가 없습니다. \n");

else {

currentLoginUser = userCollection->findUser(ID);

fprintf(out\_fp, "2.1. 로그인\n> %s %s \n", ID, Pwd);

}

}

//2.2 로그아웃

//Function :void logOut()

//Description : 로그아웃 하는 함수

//Parameters : void

//Return Value : void

//Created : 2017/05/30 9:38 am

//Author : 조현정

void logOut() {

Logout \*logout = ControlAndCollection::GetInstance()->getLogout();

LogoutUI \*logoutUI = logout->callLogoutUIStartInterface(\*logout);

char ID[MAX\_STRING];

strcpy(ID, currentLoginUser->getUserID().c\_str());

logoutUI->clickLogout(ID);

currentLoginUser = forGuest.begin();

fprintf(out\_fp, "2.2. 로그아웃\n> %s\n", ID);

}

//3.1 투표 제안

//Function :void makeVote()

//Description : 투표 제안 기능을 위해 input 값을 전달하고 output 값을 받아 출력한다.

//Parameters : void

//Return Value : void

//Created : 2017/6/1 15:00 pm

//Author : 김윤영

void makeVote() {

CreateVote \*createVote = ControlAndCollection::GetInstance()->getCreateVote();

CreateVoteUI \*createVoteUI = createVote->callCreateVoteUIStartInterface(\*createVote);

char subject[MAX\_STRING],startTime[MAX\_STRING],endTime[MAX\_STRING];

int numberOfItems;

int errorFurture1[5],errorFuture2[5];

int currentMinute, startTimeMinute;

bool error;

fscanf(in\_fp, "%s %d %s %s \n",subject,&numberOfItems, startTime,endTime);

timer.stringTimeTointTime(startTime,errorFurture1);

timer.stringTimeTointTime(timer.getTime(), errorFuture2);

startTimeMinute=timer.intTimeToMinute(errorFurture1);

currentMinute=timer.intTimeToMinute(errorFuture2);

if(startTimeMinute<currentMinute){

fprintf(out\_fp, "3.1. 투표 시작 시간은 현재 시간보다 이후여야 합니다.\n");

}

else

{

error = createVoteUI->inputDate(subject, numberOfItems, startTime, endTime, currentLoginUser->getJoinedGroup());

if (error)

fprintf(out\_fp, "3.1. 누락된 항목이 있습니다. \n");

else {

fprintf(out\_fp, "3.1. 투표 제안\n");

fprintf(out\_fp, "%s %d %s %s\n", subject, numberOfItems, startTime, endTime);

}

}

}

//4.1 현재 진행중인 투표 조회

//Function: 현재 진행중인 투표 조회

//Description: 함수

//created: 2017/5/24 22:00pm

//Author: 홍유남

//mail: bcdass@naver.com

//Function: 현재 진행중인 투표 조회

//Description: 함수 수정

//created: 2017/5/31 10:30pm

//Author: 홍유남

//mail: bcdass@naver.com

void showCurrentVoteList()

{

CurrentVoteList\* currentVoteList = ControlAndCollection::GetInstance()->getCurrentVoteList();

CurrentVoteListUI\* currentVoteListUI = currentVoteList->callCurrentVoteListUIStartInterface(\*currentVoteList);

string subject[MAX\_STRING], endTime[MAX\_STRING], startTime[MAX\_STRING], belongsTo[MAX\_STRING];

int items[MAX\_STRING];

currentVoteListUI->confirmView(subject, items, startTime, endTime, belongsTo);

fprintf(out\_fp, "4.1. 현재 진행 중인 투표 리스트\n");

for (int i = 0; subject[i].size() != 0; i++) {

char temp1[MAX\_STRING];

strcpy(temp1, belongsTo[i].c\_str());

if (currentLoginUser->getJoinedGroup() == temp1) {

char forOutput1[MAX\_STRING];

int forOutput2, compare[5];

int endTimeTokenInt[5], currentTimeTokenInt[5];

//현재시간 파싱

timer.passingTime(timer.getTime(), currentTimeTokenInt);

strcpy(forOutput1, subject[i].c\_str());

forOutput2 = items[i];

// strcpy(forOutput3, endTime[i].c\_str());

//종료시간 파싱

timer.passingTime(endTime[i], endTimeTokenInt);

//시간 비교

timer.compareTime(compare, endTimeTokenInt, currentTimeTokenInt);

timer.plusTime(compare);

fprintf(out\_fp, "> %s %d %d:%d:%d:%d:%d \n", forOutput1, forOutput2, compare[0], compare[1], compare[2], compare[3], compare[4]);

}

}

}

//4.2 투표

//Function :void voting()

//Description : 투표 기능을 위해 input 값을 전달하고 output 값을 받아 출력한다.

//Parameters : void

//Return Value : void

//Created : 2017/6/1 15:00 pm

//Author : 김윤영

void voting() {

VoteFor \*voteFor = ControlAndCollection::GetInstance()->getVoteFor();

VoteForUI \*voteForUI = voteFor->callVoteForUIStartInterface(\*voteFor);

char subject[MAX\_STRING];

int itemNumber;

int error;

fscanf(in\_fp, "%s %d \n", subject, &itemNumber);

error = voteForUI->inputData(subject, itemNumber, currentLoginUser->getJoinedGroup(), currentLoginUser->getPersonalNumber());

if (error==0) {

fprintf(out\_fp, "4.2. 없는 투표 주제이거나 항목번호가 잘못되었습니다. \n");

}

else if (error == 1) {

fprintf(out\_fp, "4.2. 이미 투표하였습니다. \n");

}

else if(error==2) {

fprintf(out\_fp, "4.2. 투표\n");

fprintf(out\_fp, "%s %d\n", subject, itemNumber);

}

}

//Function: 향후 진행 예정인 투표리스트 조회

//Description: 함수

//created: 2017/5/24 22:00pm

//Author: 홍유남

//mail: bcdass@naver.com

//4.3 향후 진행 예정인 투표 리스트

void showPostVoteList()

{

PostVoteList \*postVoteList = ControlAndCollection::GetInstance()->getPostVoteList();

PostVoteListUI\* postVoteListUI = postVoteList->callPostVoteListUIStartInterface(\*postVoteList);

string subject[MAX\_STRING], endTime[MAX\_STRING], startTime[MAX\_STRING], belongsTo[MAX\_STRING];

int items[MAX\_STRING];

postVoteListUI->confirmView(subject, items, startTime, endTime, belongsTo);

fprintf(out\_fp, "4.3. 향후 진행 예정인 투표 리스트\n");

for (int i = 0; subject[i].size() != 0; i++) {

char temp1[MAX\_STRING];

strcpy(temp1, belongsTo[i].c\_str());

if (currentLoginUser->getJoinedGroup() == temp1) {

char forOutput1[MAX\_STRING], forOutput3[MAX\_STRING], forOutput4[MAX\_STRING];

int forOutput2;

strcpy(forOutput1, subject[i].c\_str());

forOutput2 = items[i];

strcpy(forOutput3, startTime[i].c\_str());

strcpy(forOutput4, endTime[i].c\_str());

fprintf(out\_fp, "> %s %d %s %s \n", forOutput1, forOutput2, forOutput3, forOutput4);

}

}

}

//Function:종료된 투표 조회

//Description: 함수

//created: 2017/5/24 22:00pm

//Author: 홍유남

//mail: bcdass@naver.com

//4.4 종료된 투표 조회

void showPreVoteList()

{

PreVoteList \*preVoteList = ControlAndCollection::GetInstance()->getPreVoteList();

PreVoteListUI\* preVoteListUI = preVoteList->callPreVoteListUIStartInterface(\*preVoteList);

string subject[MAX\_STRING], endTime[MAX\_STRING], startTime[MAX\_STRING], belongsTo[MAX\_STRING];

int items[MAX\_STRING], \*result[MAX\_STRING];

preVoteListUI->confirmView(subject, items, startTime, endTime, belongsTo, result);

fprintf(out\_fp, "4.4. 종료된 투표 리스트\n");

for (int i = 0; subject[i].size() != 0; i++) {

//timer에서 4주 시간이 지나면 지났다는 것을 UI에 알려준다

if (timer.deleteVoteMessage(endTime[i])) {

preVoteListUI->weekLater(endTime[i]);

continue;

}

char temp1[MAX\_STRING];

strcpy(temp1, belongsTo[i].c\_str());

if (currentLoginUser->getJoinedGroup() == temp1) {

char forOutput1[MAX\_STRING];

int forOutput2;

int\* forOutpu3;

strcpy(forOutput1, subject[i].c\_str());

forOutput2 = items[i];

forOutpu3 = result[i];

fprintf(out\_fp, " %s", forOutput1);

for (int j = 0; j < items[i]; j++)

{

fprintf(out\_fp, " %d:%d", j+1, forOutpu3[j]);

}

fprintf(out\_fp, "\n");

}

}

}

//Function:전체그룹 조회

//Description: 전체그룹조회함수를 부릅니다

//created: 2017/5/28 23:00pm

//Author: 장현석

//mail: jang8018@naver.com

//5.1 전체 그룹 조회

void showEntireGroup() {

ShowAllGroup \*showAllGroup = ControlAndCollection::GetInstance()->getShowAllGroup();

ShowAllGroupUI \*showAllGroupUI = showAllGroup->callShowAllGroupUIStartInterface(\*showAllGroup);

string groupList[MAX\_STRING];

showAllGroupUI->confirmView(groupList);

fprintf(out\_fp, "5.1. 전체그룹 조회\n");

for (int i = 0; groupList[i].size() != 0; i++) {

char forOutput[MAX\_STRING];

strcpy(forOutput, groupList[i].c\_str());

fprintf(out\_fp, "> %s \n", forOutput);

}

}

//Function:그룹 가입

//Description: 그룹가입함수를 부릅니다

//created: 2017/5/28 23:10pm

//Author: 장현석

//mail: jang8018@naver.com

//5.2 그룹 가입

void joinGroupMember() {

JoinGroup \*joinGroup = ControlAndCollection::GetInstance()->getJoinGroup();

JoinGroupUI \*joinGroupUI = joinGroup->callJoinGroupUIStartInterface(\*joinGroup);

char buf[MAX\_STRING];

bool error;

//문자열을 받습니다

fscanf(in\_fp, "%s", buf);

string inputGroupName(buf);

//가입함수 호출

error = joinGroupUI->selectGroup(inputGroupName, currentLoginUser);

if (error == true)

fprintf(out\_fp, "5.2. Error: 존재하지않는 그룹 이름입니다.\n");

else

fprintf(out\_fp, "5.2. 그룹 가입\n> %s\n", buf);

}

//Function:그룹 생성

//Description: 그룹생성함수 부릅니다

//created: 2017/5/27 11:10pm

//Author: 장현석

//mail: jang8018@naver.com

//5.3 그룹 생성

void makeGroup() {

CreateGroup \*createGroup = ControlAndCollection::GetInstance()->getCreateGroup();

CreateGroupUI \*createGroupUI = createGroup->callCreateGroupUIStartInterface(\*createGroup);

GroupCollection \*groupCollection = ControlAndCollection::GetInstance()->getGroupCollection();

char buf[MAX\_STRING];

bool error;

//문자열을 받습니다.

fscanf(in\_fp, "%s", buf);

string inputGroupName(buf);

//생성함수 호출

error=createGroupUI->setGroupName(inputGroupName, currentLoginUser);

if (error == true)

fprintf(out\_fp, "5.3. Error: 그룹을 생성할 수 없습니다.\n");

else

fprintf(out\_fp, "5.3. 그룹 생성\n> %s \n", buf);

//groupCollection.printAllGroup();

}

//Function:가입그룹조회

//Description: 가입그룹조회를 부릅니다

//created: 2017/5/27 11:30pm

//Author: 장현석

//mail: jang8018@naver.com

//5.4 가입그룹 조회

void show\_JoinedGroup()

{

ShowJoinedGroup \*showJoinedGroup = ControlAndCollection::GetInstance()->getShowJoinedGroup();

ShowJoinedGroupUI \*showJoinedGroupUI = showJoinedGroup->callShowJoinedGroupUIStartInterface(\*showJoinedGroup);

char buf[MAX\_STRING];

bool error;

string joinedGroupName;

error = showJoinedGroupUI->confirmView(currentLoginUser, joinedGroupName);

if (error == true)

fprintf(out\_fp, "5.4. Error: 가입된 그룹이 없습니다.\n");

else

{

strcpy(buf, joinedGroupName.c\_str());

fprintf(out\_fp, "5.4. 가입그룹 조회\n> %s \n", buf);

}

}

//Function:그룹탈퇴

//Description: 그룹탈퇴함수를 부릅니다

//created: 2017/5/27 12:10pm

//Author: 장현석

//mail: jang8018@naver.com

//5.5 그룹탈퇴

void leave\_Group()

{

LeaveGroup \*leaveGroup = ControlAndCollection::GetInstance()->getLeaveGroup();

LeaveGroupUI \*leaveGroupUI = leaveGroup->callJoinGroupUIStartInterface(\*leaveGroup);

char buf[MAX\_STRING];

bool error;

string joinedGroupName;

error = leaveGroupUI->confirmLeaveGroup(currentLoginUser, joinedGroupName);

if (error == true)

fprintf(out\_fp, "5.5. Error: 그룹리더는 탈퇴할 수 없습니다.\n");

else

{

strcpy(buf, joinedGroupName.c\_str());

fprintf(out\_fp, "5.5. 그룹 탈퇴\n> %s \n", buf);

}

}

//6.1 현재시간설정

//Function :void changeCurrentTIme()

//Description : 시간을 입력받으면, 그것을 현재시간으로 설정해주는 함수.

//Parameters : void

//Return Value : void

//Created : 2017/6/1 15:00 pm

//Author : 김윤영

void changeCurrnetTime() {

//Timer timer;

VoteCollection \*currentVoteCollection = ControlAndCollection::GetInstance()->getCurrentVoteCollection();

VoteCollection \*postVoteCollection = ControlAndCollection::GetInstance()->getPostVoteCollection();

VoteCollection \*preVoteCollection = ControlAndCollection::GetInstance()->getPreVoteCollection();

char currentTime[MAX\_STRING];

fscanf(in\_fp, "%s", currentTime);

timer.setTime(currentTime);

timer.sortVote(currentVoteCollection, postVoteCollection, preVoteCollection);

fprintf(out\_fp, "6.1. 현재시간 설정\n> %s\n", currentTime);

}

// 7.1 Session 변경

//Function :void changeSession()

//Description : 입력된 ID로 세션을 변경해 주는 함수

//Parameters : void

//Return Value : void

//Created : 2017/6/1 15:00 pm

//Author : 김윤영

void changeSession() {

char ID[MAX\_STRING];

//UserCollection userCollection = ControlAndCollection::GetInstance()->getUserCollection();

fscanf(in\_fp, "%s \n", ID);

// list에서 login on 인 유저중 ID= 인것 찾기.

// currentLoginUser = userCollection.findUser(ID);

fprintf(out\_fp, "7.1. Session 변경\n> %s \n", ID);

}

// 7.2 guest Session으로 변경

//Function :void guestSession()

//Description : 현재 유저=guest로 변경해주는 함.

//Parameters : void

//Return Value : void

//Created : 2017/6/1 15:00 pm

//Author : 김윤영

void guestSession() {

fprintf(out\_fp, "7.2. guest session으로 변경\n");

currentLoginUser=forGuest.begin();

}

void program\_Exit() {

fprintf(out\_fp, "8.1. 종료\n");

}WidhdrwalUI.h

#pragma once

#include <string>

#include <iostream>

#include <list>

using namespace std;

class Withdrwal;

class User;

class UserCollection;

// Class : WithdrwalUI

// Description: 회원탈퇴 기능에 대해 boundary 클래스의 역할을 하는 클래스

// Created: 2017/5/30 2:40 pm

// Author: 조현정

// mail: guswjd8718@naver.com

//

class WithdrwalUI

{

private:

Withdrwal\* withdrwalPointer;

public:

static WithdrwalUI\* startInterface(Withdrwal& withdrwal);

bool clickWithdrwal(string idOfUser);

WithdrwalUI(Withdrwal& withdrwal);

~WithdrwalUI();

};

WidhdrwalUI.cpp

#include "WithdrwalUI.h"

#include "Withdrwal.h"

#include "UserCollection.h"

// Function : WithdrwalUI(Withdrwal& withdrwal)

// Description: WithdrwalUI를 생성해주는 생성자

// Parameters : Withdrwal& withdrwal

// Return Value : none.

//

// Created: 2017/5/30 3:50 pm

// Author: 조현정

//

WithdrwalUI::WithdrwalUI(Withdrwal& withdrwal)

{

withdrwalPointer = &withdrwal;

}

// Function : WithdrwalUI\* startInterface(Withdrwal& withdrwal)

// Description: WithdrwalUI를 시작하는 startInterface함수

// Parameters : WithdrwalUI

// Return Value : Withdrwal& withdrwal

//

// Created: 2017/5/30 3:56 pm

// Author: 조현정

//

WithdrwalUI \* WithdrwalUI::startInterface(Withdrwal & withdrwal)

{

WithdrwalUI\* newPointer = new WithdrwalUI(withdrwal);

return newPointer;

}

// Function : bool clickWithdrwal(string idOfUser)

// Description: callDestroyUser의 결과를 반환하는 함수

// Parameters : bool

// Return Value : Withdrwal& withdrwal

//

// Created: 2017/5/30 3:56 pm

// Author: 조현정

//

bool WithdrwalUI::clickWithdrwal(string idOfUser)

{

return withdrwalPointer->callDestroyUser(idOfUser);

}

// Function : WithdrwalUI(Withdrwal& withdrwal)

// Description: WithdrwalUI를 파괴하는 소멸자

// Parameters : void

// Return Value : none.

//

// Created: 2017/5/30 3:52 pm

// Author: 조현정

//

WithdrwalUI::~WithdrwalUI()

{

}

Withdwal.h

#include <string>

#include <iostream>

#include <list>

using namespace std;

class UserCollection;

class WithdrwalUI;

class User;

// Class : Withdrwal

// Description: 회원탈퇴 기능에서 control클래스의 역할을 하는 클래스

// Created: 2017/5/30 2:36 pm

// Author: 조현정

// mail: guswjd8718@naver.com

//

class Withdrwal {

private:

UserCollection\* userCollectionPointer;

WithdrwalUI\* withdrwalPointer;

bool isExists;

public:

bool callDestroyUser(string idOfUser);

void checkLeader(string idOfUser);

Withdrwal(UserCollection& userCollection);

~Withdrwal();

WithdrwalUI\* callWithdrwalUIStartInterface(Withdrwal& withdrwal);

};

Withdwal.cpp

#include "Withdrwal.h"

#include "WithdrwalUI.h"

#include "UserCollection.h"

#include "User.h"

// Function : void calDestroyUser(string idOfUser)

// Description: 회원탈퇴를 할 수 있는지 없는지 판별하여 removeUser함수를 불러주는 함수

// Parameters : string idOfUser

// Return Value : void

//

// Created: 2017/5/30 3:09 pm

// Author: 조현정

//

bool Withdrwal::callDestroyUser(string idOfUser)

{

checkLeader(idOfUser);

if (isExists)

{

//회원탈퇴 실패

return 1;

}

else

{

//회원탈퇴 성공

userCollectionPointer->removeUser(idOfUser);

return 0;

}

}

// Function : void checkLeader(string idOfUser)

// Description: 리더인지 아닌지 판별하여 회원탈퇴를 진행함.

// Parameters : string idOfUser

// Return Value : void

//

// Created: 2017/5/30 3:45 pm

// Author: 조현정

//

void Withdrwal::checkLeader(string idOfUser)

{

list<User>::iterator iter = userCollectionPointer->findUser(idOfUser);

if (iter == userCollectionPointer->findEnd()) //존재하지 않는 아이디. 회원탈퇴 기능 자체 못이용함.

isExists = true;

else

{

if (iter->getIsLeader() == false)

{

//그룹장이 아니므로 회원 탈퇴 성공

isExists = false;

}

else

{

//그룹장이므로 회원탈퇴 실패

isExists = true;

}

}

}

// Function : Withdrwal(UserCollection& userCOllection)

// Description: Withdrwal클래스의 생성자

// Parameters : UserCollection& userCollection

// Return Value : none.

//

// Created: 2017/5/30 3:07 pm

// Author: 조현정

//

Withdrwal::Withdrwal(UserCollection & userCollection)

{

userCollectionPointer = &userCollection;

}

// Function : ~Withdrwal(UserCollection& userCOllection)

// Description: Logout클래스의 소멸자

// Parameters : void

// Return Value : none.

//

// Created: 2017/5/30 3:08 pm

// Author: 조현정

//

Withdrwal::~Withdrwal()

{

}

// Function : void callWithdrwalUIStartInterface(Withdrwal& withdrwal)

// Description: 회원탈퇴UI의 interface를 반환해주는 함수

// Parameters : Withdrwal& withdrwal;

// Return Value : WithdrwalUI\*

//

// Created: 2017/5/30 3:40pm

// Author: 조현정

//

WithdrwalUI \* Withdrwal::callWithdrwalUIStartInterface(Withdrwal & withdrwal)

{

return WithdrwalUI::startInterface(withdrwal);

}

VoteForUI.h

#pragma once

#include <iostream>

#include <string>

#include <list>

using namespace std;

class VoteFor;

//Class: VoteForUI

//Description: 투표 기능(VoteFor)의 boundary 클래스 역할을 수행함.

//created: 2017/6/1 15:00pm

//Author: 김윤영

//mail: kyy01114@hanmail.net

class VoteForUI

{

private:

VoteFor\* voteForPointer;

public:

static VoteForUI\* startInterface(VoteFor &voteFor);

//Function: static VoteForUI\* startInterface(VoteFor &voteFor);

//Description: control class에 의해 startinterface는 수행된다.

//Parameters: VoteFor& voteFor - 투표 기능(votefor)의 control class

//Author: 김윤영

//Created: 2017/6/1 15:00pm

int inputData(string subject, int itemNumber,string joinedGroup, string personalNumber);

//Function: int inputData(string subject, int itemNumber,string joinedGroup, string personalNumber)

//Description: 필요한 데이터를 control로 넘기는 기능을 수행한다.

//Parameters: VoteFor& voteFor - 투표 기능(votefor)의 control class

//Return Value: error value, 2가지 error와 1가지 성공, 총 3가지를 구분하기 위해 int를 사용

//Author: 김윤영

//Created: 2017/6/1 15:00pm

VoteForUI(VoteFor &voteFor);

~VoteForUI();

//Function: VoteForUI(VoteFor &voteFor), ~VoteForUI()

//Description: 생성자와 소멸자, UI는 control에 의해 생성된다.

//Parameters: VoteFor& voteFor - 투표 기능(votefor)의 control class

//Author: 김윤영

//Created: 2017/6/1 15:00pm

};

VoteForUI.cpp

#include "VoteForUI.h"

#include "VoteFor.h"

VoteForUI::VoteForUI(VoteFor &voteFor)

{

voteForPointer = &voteFor;

}

VoteForUI::~VoteForUI()

{

}

VoteForUI \* VoteForUI::startInterface(VoteFor & voteFor)

{

VoteForUI\* newPointer = new VoteForUI(voteFor);

return newPointer;

}

int VoteForUI::inputData(string subject, int itemNumber,string joinedGroup, string personalNumber)

{

return voteForPointer->doVote(subject,itemNumber,joinedGroup,personalNumber);

}

VoteFor.h

#pragma once

#include <iostream>

#include <string>

#include <list>

using namespace std;

class VoteForUI;

class VoteCollection;

//Class: VoteFor

//Description: 투표 기능(VoteFor)의 control 클래스 역할을 수행함.

//created: 2017/6/1 15:00pm

//Author: 김윤영

//mail: kyy01114@hanmail.net

class VoteFor

{

private:

VoteCollection\* voteCollectionPointer;

VoteForUI\* voteForUIPointer;

public:

VoteFor(VoteCollection &voteCollection);

~VoteFor();

//Function: VoteFor(VoteCollection &voteCollection), ~VoteFor()

//Description: 생성자와 소멸자, 투표(voteFor) control class는 votecollection의 레퍼런스를 가지고 있어야한다.

//Parameters: VoteCollection &voteCollection - 투표 collection

//Author: 김윤영

//Created: 2017/6/1 15:00pm

int doVote(string subject, int itemNumber, string joinedGroup, string personalNumber);

//Function: int doVote(string subject, int itemNumber, string joinedGroup, string personalNumber);

//Description: vote에 있는 dovotefor에 들어가기 전에 error들을 걸러준다.

//Parameters: string subject - 투표 주제, int itemNumber - 투표하는 항목번호, string joinedGroup - 유저가 가입된 그룹, string personalNumber - 유저의 주민번호

//Return Value: error value, error을 2가지 error와 성공을 구분하기 위해 int를 사용

//Author: 김윤영

//Created: 2017/6/1 15:00pm

VoteForUI\* callVoteForUIStartInterface(VoteFor &voteFor);

//Function: VoteForUI\* callVoteForUIStartInterface(VoteFor &voteFor);

//Description: control class는 입력 받기전 UI를 생성해야된다.

//Parameters: VoteFor& voteFor - 투표 기능(votefor)의 control class

//Author: 김윤영

//Created: 2017/6/1 15:00pm

};

VoteFor.cpp

#include "VoteFor.h"

#include "VoteForUI.h"

#include "Vote.h"

#include "VoteCollection.h"

VoteFor::VoteFor(VoteCollection &voteCollection)

{

voteCollectionPointer = &voteCollection;

}

VoteFor::~VoteFor()

{

}

int VoteFor::doVote(string subject, int itemNumber, string joinedGroup, string personalNumber)

{

list<Vote>::iterator iter = voteCollectionPointer->findFirst();

while (iter != voteCollectionPointer->findEnd())

{

if (!joinedGroup.compare(iter->getBelongsTo()))

{

//그룹 내의 vote에서

if (!subject.compare(iter->getSubject()))

{

// 주제가 같은 것이 있다면

if (itemNumber <= iter->getItems())

{

// 존재하는 항목 번호 인경우

return iter->doVoteFor(itemNumber,personalNumber);

}

}

}

iter++;

}

return 0; //에러가 있음

}

VoteForUI \* VoteFor::callVoteForUIStartInterface(VoteFor & voteFor)

{

return VoteForUI::startInterface(voteFor);

}

VoteCollection.h

#pragma once

#include <iostream>

#include <string>

#include <list>

//Class: VoteCollection

//Description: VoteCollection 함수

//created: 2017/5/29 10:00

//Author: 홍유남

//mail: bcdass@naver.com

using namespace std;

class Vote;

class VoteCollection {

private:

list<Vote> voteList;

list<Vote>::iterator voteListIterator;

public:

list<Vote>::iterator findFirst();

list<Vote>::iterator getNext(list<Vote>::iterator &inputIter);

list<Vote>::iterator findEnd();

void addVote(string subjectOfVote, int numberOfItems, string firstStartTime, string firsrEndTime, string groupName); //새로운 vote 생성

list<Vote>\* getVoteListHead();

VoteCollection();

~VoteCollection();

};

VoteCollection.cpp

#include "Vote.h"

#include "VoteCollection.h"

using namespace std;

VoteCollection::VoteCollection()

{

}

VoteCollection::~VoteCollection()

{

}

// Function : findFirst

// Description: 투표 리스트의 첫번째 값을 받아옶니다

// Parameters :

// Return Value : iterator

//

// Created: 2017/5/30 22:00pm

// Author: 홍유남

//

list<Vote>::iterator VoteCollection::findFirst()

{

voteListIterator = voteList.begin();

return voteListIterator;

}

// Function : findEnd

// Description: 투표리스트의 끝값을 받아옵니다.

// Parameters :

// Return Value : iterator

//

// Created: 2017/5/30 22:00pm

// Author: 홍유남

//

list<Vote>::iterator VoteCollection::findEnd()

{

voteListIterator = voteList.end();

return voteListIterator;

}

// Function : getNext

// Description: iterator를 다음 투표 리스트를 가르치게 합니다.

// Parameters : iterator

// Return Value : iterator

//

// Created: 2017/5/30 22:00pm

// Author: 홍유남

//

list<Vote>::iterator VoteCollection::getNext(list<Vote>::iterator &inputIter)

{

return inputIter++;

}

// Function : addVote

// Description: 리스트에 투표를 추가합니다

// Parameters : string , int , string, string, string

// Return Value :

//

// Created: 2017/5/30 22:00pm

// Author: 홍유남

//

void VoteCollection::addVote(string subjectOfVote, int numberOfItems, string firstStartTime, string firsrEndTime, string groupName)

{

voteList.push\_back(\*Vote::createVote(subjectOfVote, numberOfItems, firstStartTime, firsrEndTime,groupName));

}

// Function : getVoteListHead

// Description: 투표리스트의 주소값을 받아옵니다.

// Parameters :

// Return Value : &list

//

// Created: 2017/5/30 22:00pm

// Author: 홍유남

//

list<Vote>\* VoteCollection::getVoteListHead()

{

return &voteList;

}

Vote.h

#pragma once

#include <iostream>

#include <string>

#include <list>

#define MAX\_LIST 100

//Class: Vote

//Description: Vote 헤더

//created: 2017/5/24 20:00pm

//Author: 홍유남

//mail: bcdass@naver.com

using namespace std;

class Vote;

//extern list<Vote> VoteList;

class Vote {

private:

string subject;

int items[MAX\_LIST];

string startTime;

string endTime;

string personalNumberList[MAX\_LIST]; // 재투표 방지를 위해서, 투표했던 사람들의 personal number(중복 안되게)

string belongsTo; // 속한 그룹 이름

public:

Vote(string subjectOfVote, int items, string startTime, string endTime,string groupName);

~Vote();

static Vote\* createVote(string subjectOfVote, int numberOfItems, string firstStartTime, string firsrEndTime, string groupName);// 투표 생성

void deleteVote();// 투표삭제

int doVoteFor(int itemNuber, string personalNumber);

string getSubject();

void setSubject(string inputSubject);

int getItems();

void setItems(int inputItems);

string getStartTime();

void setStartTime(string inputStartTime);

string getEndTime();

void setEndTime(string inputEndTime);

string getBelongsTo();

void setBelongsTo(string inputBelongsTo);

int\* getResult();

};

Vote.cpp

#include "Vote.h"

#include "VoteCollection.h"

//Class:

//Description:

//created: 2017/5/30 22:00pm

//Author: 홍유남

//mail: bcdass@naver.com

// Function :

// Description:

// Parameters :

// Return Value :

//

// Created: 2017/5/30 22:00pm

// Author: 홍유남

//

// Function : Vote

// Description: 생성자를 이용한 vote 초기화

// Parameters : string int string string string

// Return Value :

//

// Created: 2017/5/24 20:00

// Author: 홍유남

//

Vote::Vote(string subjectOfVote, int numberOfItems, string firstStartTime, string firstEndTime, string groupName)

{

subject = subjectOfVote;

for (int i = 0; i < MAX\_LIST; i++) {

if (i < numberOfItems)

items[i] = 0;

else

items[i] = -1;

}

startTime = firstStartTime;

endTime = firstEndTime;

belongsTo = groupName;

}

Vote::~Vote()

{

}

Vote\* Vote::createVote(string subjectOfVote, int numberOfItems, string firstStartTime, string firstEndTime, string groupName)// 투표 생성

{

Vote\* newVote = new Vote(subjectOfVote, numberOfItems, firstStartTime, firstEndTime, groupName);

return newVote;

}

// Function : deleteVote

// Description: 소멸자를 이용한 소멸 함수

// Parameters :

// Return Value :

//

// Created: 2017/5/24 20:00

// Author: 홍유남

//

void Vote::deleteVote()

{

Vote::~Vote();

}

int Vote::doVoteFor(int itemNuber, string newPersonalNumber)

{

int i = 0;

for ( i = 0; personalNumberList[i].size() != 0; i++)

{

if (!personalNumberList[i].compare(newPersonalNumber))

{ //이미 존재하는 경우

return 1;

}

}

personalNumberList[i] = newPersonalNumber;

setItems(itemNuber);

return 2;

}

string Vote::getSubject()

{

return subject;

}

void Vote::setSubject(string inputSubject)

{

subject = inputSubject;

}

int Vote::getItems()

{

int cnt = 0;

while (items[cnt++] != -1);

return cnt-1;

}

void Vote::setItems(int inputItems)

{

items[inputItems - 1]++;

}

string Vote::getStartTime()

{

return startTime;

}

void Vote::setStartTime(string inputStartTime)

{

startTime = inputStartTime;

}

string Vote::getEndTime()

{

return endTime;

}

void Vote::setEndTime(string inputEndTime)

{

endTime = inputEndTime;

}

string Vote::getBelongsTo()

{

return belongsTo;

}

void Vote::setBelongsTo(string inputBelongsTo)

{

belongsTo=inputBelongsTo;

}

int \* Vote::getResult()

{

return items;

}

UserCollection.h

#pragma once

#include <iostream>

#include <string>

#include <list>

class User;

using namespace std;

class UserCollection {

private:

list<User> userList;

list<User>::iterator userListIterator;

public:

list<User>::iterator findFirst();

list<User>::iterator getNext(list<User>::iterator &inputIter);

list<User>::iterator findEnd();

list<User>::iterator findUser(string idOfUser);

void addUser(string newName, string newPersonalNumber, string newAddress, string newID, string newPwd);

void removeUser(string idOfUser);

void setLoginUser(string idOfUser);

void setLoginOffUser(string idOfUser);

UserCollection();

~UserCollection();

};

UserCollection..cpp

#include "UserCollection.h"

#include "User.h"

//Class: UserCollection

//Description: UserCollection 구현

//created: 2017/5/27 21:00pm

//Author: 장현석

//mail: jang8018@naver.com

//수정

//1. 2017/05/30 3:36 am by조현정

//setLoginOff 함수 추가

//수정

//2. 2017/05/30 3:11 pm by 조현정

//removeUser함수 추가

//extern된 유저리스트 불러오기

UserCollection::UserCollection(){}

UserCollection::~UserCollection(){}

//Function: list<User>::iterator findFirst()

//Description: 유저 리스트에 첫번째 반복자를 반환한다.

//created: 2017/5/27 21:03pm

//Author: 장현석

//mail: jang8018@naver.com

list<User>::iterator UserCollection::findFirst()

{

userListIterator = userList.begin();

return userListIterator;

}

//Function: list<User>::iterator findEnd()

//Description: 유저 리스트에 끝을 가리키는 반복자를 반환한다.

//created: 2017/5/27 21:05pm

//Author: 장현석

//mail: jang8018@naver.com

list<User>::iterator UserCollection::findEnd()

{

userListIterator = userList.end();

return userListIterator;

}

//Function: list<User>::iterator getNext(list<User>::iterator &inputIter)

//Description: 유저 리스트의 반복자를 다음으로 이동시켜준다.

//created: 2017/5/27 21:07pm

//Author: 장현석

//mail: jang8018@naver.com

list<User>::iterator UserCollection::getNext(list<User>::iterator &inputIter)

{

return inputIter++;

}

//Function: list<User>::iterator findUser(string idOfUser)

//Description: 유저 리스트에서 해당하는 아이디의 반복자를 리턴함.

//created: 2017/5/27 21:10pm

//Author: 장현석

//mail: jang8018@naver.com

list<User>::iterator UserCollection::findUser(string idOfUser)

{

userListIterator = userList.begin();

bool success = false; //성공여부

//리스트 검색

for (userListIterator = userList.begin(); userListIterator != userList.end(); userListIterator++)

{

if (userListIterator->getUserID() == idOfUser)

{

success = true;

break;

}

}

return userListIterator;

}

//Function: void addUser(string newName, string newPersonalNumber, string newAddress, string newID, string newPwd)

//Description: 유저 리스트에 새로운 유저를 추가한다.

//created: 2017/5/27 21:14pm

//Author: 장현석

//mail: jang8018@naver.com

void UserCollection::addUser(string newName, string newPersonalNumber, string newAddress, string newID, string newPwd)

{

userList.push\_back(\*User::createUser(newName,newPersonalNumber,newAddress,newID,newPwd));

}

// Function : void removeUser(string idOfUser)

// Description: 유저를 소멸하고 리스트에서 삭제해주는 함수

// Parameters : string idOfUser

// Return Value : void

//

// Created: 2017/5/30 3:20 pm

// Author: 조현정

//

void UserCollection::removeUser(string idOfUser)

{

userListIterator = userList.begin();

bool find = false;

for (userListIterator = userList.begin(); userListIterator != userList.end(); userListIterator++)

{

if (userListIterator->getUserID() == idOfUser)

{

userListIterator->deleteUser();

userListIterator = userList.erase(userListIterator);

find = true;

break;

}

}

}

//Function: void setLoginUser(string idOfUser)

//Description: 유저의 로그인 상태를 로그인으로 바꾼다.

//created: 2017/5/27 21:17pm

//Author: 장현석

//mail: jang8018@naver.com

void UserCollection::setLoginUser(string idOfUser)

{

list<User>::iterator iter = UserCollection::findUser(idOfUser);

iter->setLoginOn();

}

// Function : void setLogOinffUser(string idOfUser)

// Description: 유저의 setLoginOff함수를 호출해주는 함수

// Parameters : string idOfUser

// Return Value : void

//

// Created: 2017/5/30 3:39 am

// Author: 조현정

//

void UserCollection::setLoginOffUser(string idOfUser)

{

list<User>::iterator iter = UserCollection::findUser(idOfUser);

iter->setLoginOff();

}

User.h

#pragma once

#include <iostream>

#include <string>

#include <list>

using namespace std;

class User;

class User {

private:

string userName;

string userID;

string personalNumber;

string address;

string userPwd;

string joinedGroup;

bool loginState;

bool isLeader;

public:

static User\* createUser(string newName, string newPersonalNumber,string newAddress, string newID, string newPwd);

void deleteUser();

void setLoginOn();

void setLoginOff();

void setGroupLeader();

void setJoinedGroup(string nameOfGroup);

void withdrawGroup();

User(string newName, string newPersonalNumber, string newAddress, string newID, string newPwd);

~User();

string getUserID();

string getUserPwd();

string getJoinedGroup();

string getUserName();

string getPersonalNumber();

bool getIsLeader();

};

User.cpp

#include "User.h"

//Class: User

//Description: User 구현

//created: 2017/5/27 21:00pm

//Author: 장현석

//mail: jang8018@naver.com

User::User(string newName, string newPersonalNumber, string newAddress, string newID, string newPwd)

{

userName = newName;

personalNumber = newPersonalNumber;

userID = newID;

address = newAddress;

userPwd = newPwd;

joinedGroup = "none";

isLeader = false;

loginState = false;

}

User::~User()

{

}

User\* User::createUser(string newName, string newPersonalNumber, string newAddress, string newID, string newPwd)

{

User\* newUser = new User(newName,newPersonalNumber,newAddress,newID,newPwd);

return newUser;

};

void User::deleteUser()

{

User::~User();

};

void User::setLoginOn()

{

loginState = true;

};

void User::setLoginOff()

{

loginState = false;

};

void User::setGroupLeader()

{

isLeader = true;

}

void User::setJoinedGroup(string nameOfGroup)

{

joinedGroup = nameOfGroup;

};

void User::withdrawGroup()

{

joinedGroup = "none";

};

string User::getJoinedGroup()

{

return joinedGroup;

};

string User::getUserID()

{

return userID;

}

string User::getUserPwd()

{

return userPwd;

}

string User::getUserName()

{

return userName;

}

string User::getPersonalNumber()

{

return personalNumber;

}

bool User::getIsLeader()

{

return isLeader;

}

Timer.h

#pragma once

#include <iostream>

#include <string>

#include <list>

#include<ctime>

using namespace std;

//Class: Timer

//Description: Time 헤더

//created: 2017/5/24 20:00pm

//Author: 홍유남

//mail: bcdass@naver.com

//Class: Timer

//Description: Time 헤더 함수 추가

//created: 2017/5/31 10:00pm

//Author: 홍유남

//mail: bcdass@naver.com

// Revsions :

// 1. When & Who : 2017/6/1 15:00pm 김윤영

// 2. What : void sortVote(VoteCollection\* currentVote, VoteCollection\* postVote, VoteCollection\* preVote);

// void stringTimeTointTime(string time, int\* result);, int intTimeToMinute(int\* intTime);

// 총 3가지 함수 구현

#define MAX\_STRING 50

class Timer;

class VoteCollection;

class Vote;

class Timer{

private:

char currentTime[MAX\_STRING];

int year;

int month;

int day;

int hour;

int minute;

list<Vote> allVoteList;

public:

Timer();

~Timer();

void setTime(string inputTime);

string getTime();

void sortVote(VoteCollection\* currentVote, VoteCollection\* postVote, VoteCollection\* preVote);

//Function :void sortVote(VoteCollection\* currentVote, VoteCollection\* postVote, VoteCollection\* preVote);

//Description: 시간이 바뀔때 사용하고 vote가 현재, 향후 진행, 과거 투표인지 구분해서 각각의 collection에 넣어주는 기능

//Parameters: VoteCollection\* currentVote, VoteCollection\* postVote, VoteCollection\* preVote

//Author: 김윤영

//Created: 2017/6/1 15:00pm

void stringTimeTointTime(string time, int\* result);

//Function: void stringTimeTointTime(string time, int\* result);

//Description: string 타입으로 되어있는 시간을 intResult[5](year, month, day hour, minute)

//Parameters: string time, int\* result

//Author: 김윤영

//Created: 2017/6/1 15:00pm

int intTimeToMinute(int\* intTime);

//Function: int intTimeToMinute(int\* intTime);

//Description: intTime[5](year, month, day hour, minute) 를 분으로 바꿔주는 함수.

//Parameters: int\* intTime (year, month, day hour, minute)

//Return Value: 총 분

//Author: 김윤영

//Created: 2017/6/1 15:00pm

void passingTime(string inputTime ,int compare[4]);

void compareTime(int compare[4], int input1[4], int input2[4]);

void plusTime(int compare[4]);

bool deleteVoteMessage(string endTime);

};

Timer.cpp

#include "timer.h"

#include "Vote.h"

#include "VoteCollection.h"

#pragma warning(disable:4996)

Timer::Timer()

{

}

Timer::~Timer()

{

}

void Timer::setTime(string inputTime)

{

strcpy(currentTime, inputTime.c\_str());

char charYear[5] = { 0 }, charMonth[3] = { 0 }, charDay[3] = { 0 }, charHour[3] = { 0 }, charMinute[3] = { 0 };

for (int i = 0; i < inputTime.length(); i++) {

if (i < 4)

charYear[i] = currentTime[i];

else if (i>4 && i < 7)

charMonth[i - 5] = currentTime[i];

else if (i > 7 && i < 10)

charDay[i - 8] = currentTime[i];

else if (i > 10 && i < 13)

charHour[i - 11] = currentTime[i];

else if (i > 13 && i < 16)

charMinute[i - 14] = currentTime[i];

}

year = atoi(charYear);

month = atoi(charMonth);

day = atoi(charDay);

hour = atoi(charHour);

minute = atoi(charMinute);

}

string Timer::getTime()

{

return currentTime;

}

void Timer::sortVote(VoteCollection\* currentVote, VoteCollection\* postVote, VoteCollection\* preVote)

{

allVoteList.clear();

list<Vote>::iterator iter;

iter = currentVote->findFirst();

// currentVote, postVote, preVote 모두 => allListVote에 넣기

while (iter != currentVote->findEnd())

{

Vote temp = \*iter;

allVoteList.push\_back(temp);

iter++;

}

currentVote->getVoteListHead()->clear();

iter = postVote->findFirst();

while (iter != postVote->findEnd())

{

Vote temp = \*iter;

allVoteList.push\_back(temp);

iter++;

}

postVote->getVoteListHead()->clear();

iter = preVote->findFirst();

while (iter != preVote->findEnd())

{

Vote temp = \*iter;

allVoteList.push\_back(temp);

iter++;

}

preVote->getVoteListHead()->clear();

// 분리하기

iter = allVoteList.begin();

while (iter != allVoteList.end())

{

int startTime[5], endTime[5];

stringTimeTointTime(iter->getStartTime(), startTime);

stringTimeTointTime(iter->getEndTime(), endTime);

unsigned int currentMin, startMin, endMin;

currentMin = (year \* 12 \* 30 \* 24 \* 60) + (month \* 30 \* 24 \* 60) + (day \* 24 \* 60) + (hour \* 60) + minute;

startMin = intTimeToMinute(startTime);

endMin = intTimeToMinute(endTime);

Vote temp = \*iter;

if (currentMin > endMin)

{

//prevote

preVote->getVoteListHead()->push\_back(temp);

}

else if (startMin <= currentMin && currentMin <= endMin) {

//currentvote

currentVote->getVoteListHead()->push\_back(temp);

}

else if (currentMin < startMin) {

//postVotecollection

postVote->getVoteListHead()->push\_back(temp);

}

iter++;

}

allVoteList.clear();

}

// Function : passingTime

// Description: 시간을 불러와 int로 파싱

// Parameters : string, int

// Return Value :

//

// Created: 2017/5/30 22:00pm

// Author: 홍유남

//

void Timer::passingTime(string inputTime, int compare[4])

{

char\* timeToken[5], passingTime[50], \*context;

strcpy\_s(passingTime, inputTime.c\_str());

timeToken[0] = strtok\_s(passingTime, ":", &context);

compare[0] = stoi(timeToken[0]);

for (int i = 1; i < 5; i++)

{

timeToken[i] = strtok\_s(NULL, ":", &context);

compare[i] = stoi(timeToken[i]);

}

}

void Timer::stringTimeTointTime(string time, int\* result)

{

char charYear[5] = { 0 }, charMonth[3] = { 0 }, charDay[3] = { 0 }, charHour[3] = { 0 }, charMinute[3] = { 0 };

for (int i = 0; i < time.length(); i++) {

if (i < 4)

charYear[i] = time[i];

else if (i>4 && i < 7)

charMonth[i - 5] = time[i];

else if (i > 7 && i < 10)

charDay[i - 8] = time[i];

else if (i > 10 && i < 13)

charHour[i - 11] = time[i];

else if (i > 13 && i < 16)

charMinute[i - 14] = time[i];

}

result[0] = atoi(charYear);

result[1] = atoi(charMonth);

result[2] = atoi(charDay);

result[3] = atoi(charHour);

result[4] = atoi(charMinute);

}

int Timer::intTimeToMinute(int \* intTime)

{

//년 월 일 시 분을 전부 분으로 바꿔주는 함수.

return ((intTime[0] \* 12 \* 30 \* 24 \* 60) + (intTime[1] \* 30 \* 24 \* 60) + (intTime[2] \* 24 \* 60) + (intTime[3] \* 60) + intTime[4]);

}

// Function : comPareTime

// Description: 받은 시간 2개를 빼서 비교하는 함수

// Parameters : int int int

// Return Value :

//

// Created: 2017/5/30 22:00pm

// Author: 홍유남

//

void Timer::compareTime(int compare[4], int input1[4], int input2[4])

{

for (int i = 0; i < 5; i++)

{

compare[i] = input1[i] - input2[i];

}

}

// Function : plusTime

// Description: 두시가능을 뺀값을 양수화 시켜주는 함수

// Parameters : int

// Return Value :

//

// Created: 2017/5/30 22:00pm

// Author: 홍유남

//

void Timer::plusTime(int compare[4])

{

if (compare[4]<0)

{

compare[4] = compare[4] + 60;

compare[3] = compare[3] - 1;

}

if (compare[3] < 0) {

compare[3] = compare[3] + 24;

compare[2] = compare[2] - 1;

}

if (compare[2] < 0) {

compare[2] = compare[2] + 30;

compare[1] = compare[1] - 1;

}

if (compare[1]<0) {

compare[1] = compare[1] + 12;

compare[0] = compare[0] - 1;

}

}

// Function : deleteVOteMessage

// Description: 28일이 지난 것을 알려주는 함수

// Parameters : string

// Return Value : bool

//

// Created: 2017/5/30 22:00pm

// Author: 홍유남

//

bool Timer::deleteVoteMessage(string endTime)

{

int compare[5];

passingTime(endTime, compare);

if (year - compare[0] > 0) {

return true;

}

else{

if (month - compare[1] > 0)

return true;

else

{

if (day - compare[2] > 27)

return true;

}

}

return false;

}

ShowJoinedGroupUI.h

#pragma once

#include <iostream>

#include <string>

#include <list>

//Class: ShowJoinedGroupUI

//Description: ShowJoinedGroupUI 헤더

//created: 2017/5/30 08:29am

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

class ShowJoinedGroup;

class User;

class ShowJoinedGroupUI

{

private:

ShowJoinedGroup\* showJoinedGroupPointer;

public:

bool confirmView(list<User>::iterator loginUser,string &joinedGroupName);

static ShowJoinedGroupUI\* startInterface(ShowJoinedGroup &showJoinedGroup);

ShowJoinedGroupUI(ShowJoinedGroup &showJoinedGroup);

~ShowJoinedGroupUI();

};

ShowJoinedGroupUI.cpp

#include "ShowJoinedGroupUI.h"

#include "ShowJoinedGroup.h"

#include "User.h"

//Class: ShowJoinedGroupUI

//Description: ShowJoinedGroupUI 구현

//created: 2017/5/30 08:30am

//Author: 장현석

//mail: jang8018@naver.com

// Function : ShowJoinedGroupUI(ShowJoinedGroup &showJoinedGroup)

// Description: 컨트롤을 받아 넘겨주면서 UI를 생성합니다.

// Parameters : ShowJoinedGroup &showJoinedGroup

// Return Value : void

//

// Created: 2017/5/30 08:30am

// Author: 장현석

//

ShowJoinedGroupUI::ShowJoinedGroupUI(ShowJoinedGroup &showJoinedGroup)

{

showJoinedGroupPointer = &showJoinedGroup;

}

ShowJoinedGroupUI::~ShowJoinedGroupUI()

{

}

// Function : confirmView(list<User>::iterator loginUser, string &joinedGroupName)

// Description: 받은 스트링을 넘겨주면서 컨트롤에 showJoinedGroup호출을 요청합니다.

// Parameters : list<User>::iterator loginUser, string &joinedGroupName

// Return Value : bool

//

// Created: 2017/5/30 08:32am

// Author: 장현석

//

bool ShowJoinedGroupUI::confirmView(list<User>::iterator loginUser, string &joinedGroupName)

{

return showJoinedGroupPointer->showJoinedGroup(loginUser,joinedGroupName);

}

// Function : startInterface(ShowJoinedGroup & showJoinedGroup)

// Description:

// Parameters : ShowJoinedGroup & showJoinedGroup

// Return Value : ShowJoinedGroupUI \*

//

// Created: 2017/5/30 08:36am

// Author: 장현석

//

ShowJoinedGroupUI \* ShowJoinedGroupUI::startInterface(ShowJoinedGroup & showJoinedGroup)

{

ShowJoinedGroupUI\* newPointer = new ShowJoinedGroupUI(showJoinedGroup);

return newPointer;

}

ShowJoinedGroup.h

#pragma once

#include <iostream>

#include <string>

#include <list>

//Class: ShowJoinedGroup

//Description: ShowJoinedGroup 헤더

//created: 2017/5/30 08:04am

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

class UserCollection;

class GroupCollection;

class ShowJoinedGroupUI;

class User;

class ShowJoinedGroup

{

private:

UserCollection\* userCollectionPointer;

GroupCollection\* groupCollectionPointer;

public:

ShowJoinedGroupUI\* callShowJoinedGroupUIStartInterface(ShowJoinedGroup &showJoinedGroup);

bool showJoinedGroup(list<User>::iterator loginUser, string &joinedGroupName);

ShowJoinedGroup();

//UserCollection &userCollection

~ShowJoinedGroup();

};

ShowJoinedGroup.cpp

#include "ShowJoinedGroup.h"

#include "UserCollection.h"

#include "ShowJoinedGroupUI.h"

#include "User.h"

//Class: ShowJoinedGroup

//Description: ShowJoinedGroup 구현

//created: 2017/5/30 08:10am

//Author: 장현석

//mail: jang8018@naver.com

ShowJoinedGroup::ShowJoinedGroup()

{

}

ShowJoinedGroup::~ShowJoinedGroup()

{

}

// Function : callShowJoinedGroupUIStartInterface(ShowJoinedGroup & showJoinedGroup)

// Description: 컨트롤을 받아 넘겨주면서 UI인터페이스를 호출합니다.

// Parameters : ShowJoinedGroup & showJoinedGroup

// Return Value : ShowJoinedGroupUI \*

//

// Created: 2017/5/30 08:10am

// Author: 장현석

//

ShowJoinedGroupUI \* ShowJoinedGroup::callShowJoinedGroupUIStartInterface(ShowJoinedGroup & showJoinedGroup)

{

return ShowJoinedGroupUI::startInterface(showJoinedGroup);

}

// Function : showJoinedGroup(list<User>::iterator loginUser, string& joinedGroupName)

// Description: 유저가 속한 그룹이름을 넘겨받은 스트링에 저장하고, 결과를 반환합니다.

// Parameters : list<User>::iterator loginUser, string& joinedGroupName

// Return Value : bool

//

// Created: 2017/5/30 08:15am

// Author: 장현석

//

bool ShowJoinedGroup::showJoinedGroup(list<User>::iterator loginUser, string& joinedGroupName)

{

if (loginUser->getJoinedGroup() == "none")

{

return true;

}

else

{

joinedGroupName = loginUser->getJoinedGroup();

return false;

}

}

ShowAllGroupUI.h

#pragma once

#include <iostream>

#include <string>

#include <list>

//Class: ShowAllGroupUI

//Description: ShowAllGroupUI 헤더

//created: 2017/5/29 09:07am

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

class ShowAllGroup;

class User;

class ShowAllGroupUI

{

private:

ShowAllGroup\* showAllGroupPointer;

public:

void confirmView(string\* groupList);

//void selectGroup(string nameOfGroup, list<User>::iterator loginUser);

static ShowAllGroupUI\* startInterface(ShowAllGroup &showAllGroup);

ShowAllGroupUI(ShowAllGroup &showAllGroup);

~ShowAllGroupUI();

};

ShowAllGroupUI.cpp

#include "ShowAllGroupUI.h"

#include "ShowAllGroup.h"

//Class: ShowAllGroupUI

//Description: ShowAllGroupUI 구현

//created: 2017/5/29 09:10am

//Author: 장현석

//mail: jang8018@naver.com

// Function : ShowAllGroupUI(ShowAllGroup &showAllGroup)

// Description: 컨트롤 주소를받아 포인터를 지정하고, UI를 생성합니다.

// Parameters : ShowAllGroup &showAllGroup

// Return Value : void

//

// Created: 2017/5/29 09:10am

// Author: 장현석

//

ShowAllGroupUI::ShowAllGroupUI(ShowAllGroup &showAllGroup)

{

showAllGroupPointer = &showAllGroup;

}

ShowAllGroupUI::~ShowAllGroupUI()

{

}

// Function : confirmView(string\* groupList)

// Description: 문자열배열을 전달받아 컨트롤로 넘겨주면서 showAllGroup을 호출합니다.

// Parameters : string\* groupList

// Return Value : void

//

// Created: 2017/5/29 09:20am

// Author: 장현석

//

void ShowAllGroupUI::confirmView(string\* groupList)

{

showAllGroupPointer->showAllGroup(groupList);

}

// Function : startInterface(ShowAllGroup &showAllGroup)

// Description: 컨트롤을 받아 UI에 넘겨주면서 생성자를 호출합니다.

// Parameters : ShowAllGroup &showAllGroup

// Return Value : ShowAllGroupUI\*

//

// Created: 2017/5/29 09:22am

// Author: 장현석

//

ShowAllGroupUI\* ShowAllGroupUI::startInterface(ShowAllGroup &showAllGroup)

{

ShowAllGroupUI\* newPointer = new ShowAllGroupUI(showAllGroup);

return newPointer;

}

ShowAllGroup.h

#pragma once

#include <string>

#include <list>

//Class: ShowAllGroup

//Description: ShowAllGroup 헤더

//created: 2017/5/29 08:17am

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

class ShowAllGroupUI;

class GroupCollection;

class UserCollection;

class User;

class ShowAllGroup

{

private:

GroupCollection\* groupColletionPointer;

UserCollection\* userCollectionPointer;

public:

void showAllGroup(string\* groupList);

//void joinGroup(string group\_name, list<User>::iterator loginUser);

ShowAllGroup(GroupCollection &groupCollection);

~ShowAllGroup();

ShowAllGroupUI\* callShowAllGroupUIStartInterface(ShowAllGroup &showAllGroup); //인터페이스에 자기주소전달

};

ShowAllGroup.cpp

#include "ShowAllGroup.h"

#include "ShowAllGroupUI.h"

#include "Group.h"

#include "User.h"

#include "UserCollection.h"

#include "GroupCollection.h"

//Class: ShowAllGroup

//Description: ShowAllGroup 구현

//created: 2017/5/29 08:20am

//Author: 장현석

//mail: jang8018@naver.com

// Function : ShowAllGroup(GroupCollection &groupCollection)

// Description: 그룹콜렉션주소를 받아 컨트롤을 생성합니다.

// Parameters : GroupCollection &groupCollection

// Return Value : void

//

// Created: 2017/5/29 08:20am

// Author: 장현석

//

ShowAllGroup::ShowAllGroup(GroupCollection &groupCollection)

{

groupColletionPointer = &groupCollection;

}

ShowAllGroup::~ShowAllGroup()

{

}

// Function : showAllGroup(string\* groupList)

// Description: 문자열 배열을 받아 그룹리스트에서 그룹명을받아 저장합니다.

// Parameters : string\* groupList

// Return Value : void

//

// Created: 2017/5/29 08:25am

// Author: 장현석

//

void ShowAllGroup::showAllGroup(string\* groupList)

{

list<Group>::iterator iter = groupColletionPointer->findFirst();

list<Group>::iterator iterEnd = groupColletionPointer->findEnd();

int i = 0;

while (iter != iterEnd)

{

groupList[i++]=iter->getGroupName();

groupColletionPointer->getNext(iter);

}

}

// Function : callShowAllGroupUIStartInterface(ShowAllGroup &showAllGroup)

// Description: UI의 인터페이스를 호출합니다.

// Parameters : ShowAllGroup &showAllGroup

// Return Value : ShowAllGroupUI\*

//

// Created: 2017/5/29 08:30am

// Author: 장현석

//

ShowAllGroupUI\* ShowAllGroup::callShowAllGroupUIStartInterface(ShowAllGroup &showAllGroup)

{

return ShowAllGroupUI::startInterface(showAllGroup);

}

**PreVoteListUI.h**

#pragma once

#include<iostream>

#include<string>

#include<list>

using namespace std;

class PreVoteList;

class Group;

//Class: PreVoteListUI

//Description: PreVoteListUI 클래스

//created: 2017/5/24 11:00pm

//Author: 홍유남

//mail: bcdass@naver.com

class PreVoteListUI

{

private:

PreVoteList\* preVoteListPointer;

public:

PreVoteListUI(PreVoteList &preVoteList);

~PreVoteListUI();

static PreVoteListUI\* startInterface(PreVoteList& preVoteList);

void confirmView(string\* subject, int\* items, string\* startTime, string\* endTime, string\* belongsTo, int\*\* result);

void weekLater(string endTime);

};

**PreVoteListUI.cpp**

#include "PreVoteListUI.h"

#include "PreVoteList.h"

PreVoteListUI::PreVoteListUI(PreVoteList &preVoteList)

{

preVoteListPointer = &preVoteList;

}

PreVoteListUI::~PreVoteListUI()

{

}

// Function : startInterface

// Description: PreVoteList(컨트롤 클래스)의 명령을 받아 UI를 생성

// Parameters : PreVoteList&

// Return Value : PreVoteList\*

// Created: 2017/5/24 11:00pm

// Author: 홍유남

//

PreVoteListUI \* PreVoteListUI::startInterface(PreVoteList & preVoteList)

{

PreVoteListUI\* newPointer = new PreVoteListUI(preVoteList);

return newPointer;

}

// Function : confirmView

// Description: Vote 정보값을 보여줌

// Parameters : string\* int\* string\* string\* string\* int\*\*

// Return Value : PreVoteList\*

// Created: 2017/5/24 11:00pm

// Author: 홍유남

//

void PreVoteListUI::confirmView(string \* subject, int \* items, string \* startTime, string \* endTime, string \* belongsTo, int \*\* result)

{

preVoteListPointer->showVoteList(subject, items, startTime, endTime, belongsTo, result);

}

// Function : weakLater

// Description: 4주가 지난것을 PreVoteList(컨트롤 클래스)에 알려주는 함수

// Parameters : string

// Return Value :

// Created: 2017/5/24 11:00pm

// Author: 홍유남

//

void PreVoteListUI::weekLater(string endTime)

{

preVoteListPointer->deleteVoteMessage(endTime);

}

**PreVoteList.h**

#pragma once

#include <string>

#include <list>

//Class: PreVoteList

//Description: PreVoteList 클래스

//created: 2017/5/24 11:50pm

//Author: 홍유남

//mail: bcdass@naver.com

using namespace std;

class PreVoteListUI;

class VoteCollection;

class GroupCollection;

class PreVoteList

{

private:

PreVoteListUI\* preVoteListUIPointer;

VoteCollection\* voteCollectionPointer;

public:

PreVoteList(VoteCollection &voteCollection);

~PreVoteList();

void showVoteList(string\* subject, int\* items, string\* startTime, string\* endTime,string\* belongsTo,int\*\* result);

PreVoteListUI\* callPreVoteListUIStartInterface(PreVoteList &PreVoteList);

PreVoteListUI\* getPreVoteListUIPointer();

void deleteVoteMessage(string endTime);

};

**PreVoteList.cpp**

#include "PreVoteList.h"

#include "PreVoteListUI.h"

#include "Vote.h"

#include "Group.h"

#include "VoteCollection.h"

#include"GroupCollection.h"

PreVoteList::PreVoteList(VoteCollection &voteCollection)

{

voteCollectionPointer = &voteCollection;

}

PreVoteList::~PreVoteList()

{

}

// Function : showVoteList

// Description: Vote의 정보 전달을 위한 함수

// Parameters : string\* int\* string\* string\* string\* int\*\*

// Return Value :

// Created: 2017/5/24 11:50pm

// Author: 홍유남

//

void PreVoteList::showVoteList(string \* subject, int \* items, string \* startTime, string \* endTime, string \* belongsTo, int\*\* result)

{

list<Vote>::iterator iter = voteCollectionPointer->findFirst();

list<Vote>::iterator iterEnd = voteCollectionPointer->findEnd();

int i = 0;

while (iter != iterEnd)

{

subject[i] = iter->getSubject();

items[i] = iter->getItems();

startTime[i] = iter->getStartTime();

endTime[i] = iter->getEndTime();

belongsTo[i] = iter->getBelongsTo();

result[i++] = iter->getResult();

voteCollectionPointer->getNext(iter);

}

}

PreVoteListUI \* PreVoteList::getPreVoteListUIPointer()

{

return preVoteListUIPointer;

}

// Function : deleteVoteMessage

// Description: 투표 삭제를 위한 함수 timer로부터 온 endtime을 지우라는 것을 투표리스트에 보낸다

// Parameters : string

// Return Value :

// Created: 2017/5/24 11:50pm

// Author: 홍유남

//

void PreVoteList::deleteVoteMessage(string endTime)

{

list<Vote>::iterator iter = voteCollectionPointer->findFirst();

list<Vote>::iterator iterEnd = voteCollectionPointer->findEnd();

int i = 0;

while (iter != iterEnd)

{

if (endTime == iter->getEndTime()) {

iter->deleteVote();

}

voteCollectionPointer->getNext(iter);

}

}

// Function : callPreVoteListUIStartInterface

// Description: startInterface 를 불러오는 함수

// Parameters : PreVOteList&

// Return Value : PreVoteListUI::startInterface()

// Created: 2017/5/24 11:50pm

// Author: 홍유남

//

PreVoteListUI \*PreVoteList::callPreVoteListUIStartInterface(PreVoteList & preVoteList)

{

return PreVoteListUI::startInterface(preVoteList);

}

**postVoteListUI.h**

#pragma once

#include<iostream>

#include<string>

#include<list>

using namespace std;

//Class: PostVoteListUI

//Description: PostVoteListUI 클래스

//created: 2017/5/24 12:50pm

//Author: 홍유남

//mail: bcdass@naver.com

class PostVoteList;

class Group;

class Timer;

class PostVoteListUI

{

private:

PostVoteList\* postVoteListPointer;

public:

PostVoteListUI(PostVoteList &postVoteList);

~PostVoteListUI();

static PostVoteListUI\* startInterface(PostVoteList& postVoteList);

void confirmView(string\* subject, int\* items, string\* start, string\* endTime,string\* belongsTo);

void deleteMessage(Timer &time);

};

**postVoteListUI.cpp**

#include "PostVoteListUI.h"

#include "PostVoteList.h"

PostVoteListUI::PostVoteListUI(PostVoteList &postVoteList)

{

postVoteListPointer = &postVoteList;

}

PostVoteListUI::~PostVoteListUI()

{

}

// Function : startInterface

// Description: PostVoteList(컨트롤 클래스)의 명령을 받아 UI를 생성

// Parameters : PostVoteList&

// Return Value : PostVoteList\*

// Created: 2017/5/24 12:50pm

// Author: 홍유남

//

PostVoteListUI \* PostVoteListUI::startInterface(PostVoteList & postVoteList)

{

PostVoteListUI\* newPointer = new PostVoteListUI(postVoteList);

return newPointer;

}

// Function : confirmView

// Description: Vote 정보값을 보여줌

// Parameters : string\* int\* string\* string\* string\*

// Return Value :

// Created: 2017/5/24 12:50pm

// Author: 홍유남

//

void PostVoteListUI::confirmView(string \* subject, int \* items, string \* startTime, string \* endTime,string \* belonsTo)

{

postVoteListPointer->showVoteList(subject, items, startTime, endTime, belonsTo);

}

**postVoteList.h**

#pragma once

#include <string>

#include <list>

//Class: PostVoteList

//Description: PostVoteList 클래스

//created: 2017/5/24 12:50pm

//Author: 홍유남

//mail: bcdass@naver.com

using namespace std;

class PostVoteListUI;

class VoteCollection;

class GroupCollection;

class PostVoteList

{

private:

PostVoteListUI\* postVoteListUIPointer;

VoteCollection\* voteCollectionPointer;

public:

PostVoteList(VoteCollection &voteCollection);

~PostVoteList();

void showVoteList(string\* subject, int\* items, string\* startTime, string\* endTime, string\* belongsTo);

PostVoteListUI\* callPostVoteListUIStartInterface(PostVoteList &PostVoteList);

PostVoteListUI\* getPostVoteListUIPointer();

};

**postVoteList.cpp**

#include "PostVoteList.h"

#include "PostVoteListUI.h"

#include "Vote.h"

#include "Group.h"

#include "VoteCollection.h"

#include"GroupCollection.h"

PostVoteList::PostVoteList(VoteCollection &voteCollection)

{

voteCollectionPointer = &voteCollection;

}

PostVoteList::~PostVoteList()

{

}

// Function : showVoteList

// Description: Vote의 정보 전달을 위한 함수

// Parameters : string\* int\* string\* string\* string\*

// Return Value :

// Created: 2017/5/24 12:50pm

// Author: 홍유남

//

void PostVoteList::showVoteList(string \* subject, int \* items, string \* startTime, string \* endTime,string \* belongsTo)

{

list<Vote>::iterator iter = voteCollectionPointer->findFirst();

list<Vote>::iterator iterEnd = voteCollectionPointer->findEnd();

int i = 0;

while (iter != iterEnd)

{

subject[i] = iter->getSubject();

items[i] = iter->getItems();

startTime[i] = iter->getStartTime();

endTime[i] = iter->getEndTime();

belongsTo[i++] = iter->getBelongsTo();

voteCollectionPointer->getNext(iter);

}

}

PostVoteListUI \* PostVoteList::getPostVoteListUIPointer()

{

return postVoteListUIPointer;

}

PostVoteListUI \*PostVoteList::callPostVoteListUIStartInterface(PostVoteList & postVoteList)

{

return PostVoteListUI::startInterface(postVoteList);

}

**LogOutUI.h**

#pragma once

#include <iostream>

#include <string>

#include <list>

using namespace std;

class Logout;

class User;

class UserCollection;

//Class : LogoutUI

//Description : 로그아웃 기능의 boundary클래스 역할을 하는 클래스

//Created 2017/05/30 3:25 am

//Author : 조현정

//mail : guswjd8718@naver.com

class LogoutUI

{

private:

Logout\* logoutPointer;

public:

static LogoutUI\* startInterface(Logout& logout);

void clickLogout(string idOfUser);

LogoutUI(Logout& logout);

~LogoutUI();

};

**LogOutUI.cpp**

#include "LogoutUI.h"

#include "Logout.h"

//Function :LogoutUI(Logout& logout)

//Description : LogoutUI클래스의 생성자

//Parameters : Logout& logout

//Return Value : none.

//Created : 2017/05/30 3:43 am

//Author : 조현정

LogoutUI::LogoutUI(Logout& logout)

{

logoutPointer = &logout;

}

//Function :LogoutUI\* startInterface(Logout& logout)

//Description : startInterface함수

//Parameters : Logout& logout

//Return Value : LogoutUI

//Created : 2017/05/30 3:46 am

//Author : 조현정

LogoutUI \* LogoutUI::startInterface(Logout & logout)

{

LogoutUI\* newPointer = new LogoutUI(logout);

return newPointer;

}

//Function :void clickLogout(string idOfUser)

//Description : 로그아웃을 클릭(입력)했을 때 호출되는 함수

//Parameters : string idOfUser

//Return Value : void

//Created : 2017/05/30 3:46 am

//Author : 조현정

void LogoutUI::clickLogout(string idOfUser)

{

logoutPointer->callLoginOnOff(idOfUser);

}

//Function :~LogoutUI()

//Description : LogoutUI클래스의 소멸자

//Parameters : void

//Return Value : none.

//Created : 2017/05/30 3:43 am

//Author : 조현정

LogoutUI::~LogoutUI()

{

}

**LogOut.h**

#pragma once

#include <string>

#include <iostream>

#include <list>

using namespace std;

class UserCollection;

class LogoutUI;

class User;

//class : Logout

//Description : 로그아웃 기능에 대한 control 클래스 역할을 하는 클래스

//Created : 2017/05/30 3:20 am

//Author : 조현정

//mail : guswjd8718@naver.com

class Logout

{

private:

UserCollection\* userCollectionPointer;

LogoutUI\* logoutPointer;

public:

void callLoginOnOff(string idOfUser);

Logout(UserCollection& userCollection);

~Logout();

LogoutUI\* callLogoutUIStartInterface(Logout& logout);

};

**LogOut.cpp**

#include "Logout.h"

#include "LogoutUI.h"

#include "UserCollection.h"

#include "User.h"

//Function : Logout(UserCollection& userCollection)

//Description : Logout클래스의 생성자

//Parameters : UserCollection& userCollection

//Return Value : none.

//Created : 2017/05/30 3:28 am

//Author : 조현정

Logout::Logout(UserCollection& userCollection)

{

userCollectionPointer = &userCollection;

}

//Function : void callLoginOnOff(string idOfUser)

//Description : loginOff함수를 불러주는 함수

//Parameters : string idOfUser

//Return Value : void

//Created : 2017/05/30 3:33 am

//Author : 조현정

void Logout::callLoginOnOff(string idOfUser)

{

userCollectionPointer->setLoginOffUser(idOfUser);

}

//Function : ~Logout()

//Description : Logout클래스의 파괴자

//Parameters : void

//Return Value : none.

//Created : 2017/05/30 3:30 am

//Author : 조현정

Logout::~Logout()

{

}

//Function : LogoutUI\* callLogoutUIStartInterface(Logout& logout)

//Description : 로그아웃UI의 interface를 반환해주는 함수

//Parameters : Logout& logout

//Return Value : LogoutUI\*

//Created : 2017/05/30 3:40 am

//Author : 조현정

LogoutUI \* Logout::callLogoutUIStartInterface(Logout & logout)

{

return LogoutUI::startInterface(logout);

}

**LogInUI.h**

#pragma once

#include<string>

#include<list>

#include<iostream>

using namespace std;

class Login;

class User;

class UserCollection;

// Class : LoginUI

// Description: 로그인 기능에 대해 boundary클래스의 역할을 하는 클래스

// Created: 2017/5/30 2:10 am

// Author: 조현정

// mail: guswjd8718@naver.com

//

class LoginUI

{

private:

Login\* loginPointer;

public:

static LoginUI\* startInterface(Login& login);

bool getUserInfo(string inputID, string inputPwd, list<User>::iterator allUser);

LoginUI(Login& login);

~LoginUI();

};

**LogInUI.cpp**

#include "LoginUI.h"

#include "Login.h"

// Function : LoginUI\* startInterface(Login & login)

// Description: LoginUI 객체를 생성해주는 함수.

// Parameters : Login& login

// Return Value : LoginUI\*

//

// Created: 2017/5/29 2:45 am

// Author: 조현정

//

LoginUI \* LoginUI::startInterface(Login & login)

{

LoginUI\* newPointer = new LoginUI(login);

return newPointer;

}

// Function : bool getUserInfo(string inputID, string inputPwd, list<User>::iterator allUser)

// Description: Login 클래스에게 user에게 입력받은 정보를 전달해주고, 로그인 성공 여부를 반환받는다.

// Parameters : inputID, string inputPwd, list<User>::iterator allUser

// Return Value : bool

//

// Created: 2017/5/29 2:54 am

// Author: 조현정

//

bool LoginUI::getUserInfo(string inputID, string inputPwd, list<User>::iterator allUser)

{

return loginPointer->callLoginOnOff(inputID, inputPwd, allUser);

}

// Function : LoginUI(Login& logoin)

// Description: LoginUI클래스의 생성자

// Parameters : UserCollection& userCollection

// Return Value : void

//

// Created: 2017/5/29 2:40 am

// Author: 조현정

LoginUI::LoginUI(Login& login)

{

loginPointer = &login;

}

// Function : ~LoginUI()

// Description: LoginUI클래스의 소멸자

// Parameters : void

// Return Value : void

//

// Created: 2017/5/29 2:42 am

// Author: 조현정

LoginUI::~LoginUI()

{

}

**LogIn.h**

#pragma once

#include<string>

#include<list>

#include<iostream>

using namespace std;

class UserCollection;

class LoginUI;

class User;

// Class : Login

// Description: 로그인 기능에 대해 Control클래스 역할을 하는 클래스

// Created: 2017/5/30 2:00 am

// Author: 조현정

// mail: guswjd8718@naver.com

//

class Login

{

private:

UserCollection\* userCollectionPointer;

LoginUI\* loginPointer;

bool isExists;

public:

bool callLoginOnOff(string inputID, string inputPwd, list<User>::iterator allUser);

void checkExists(string inputID, string inputPwd);

LoginUI\* callLoginUIStartInterface(Login& login);

Login(UserCollection& userCollection);

~Login();

};

**LogIn.cpp**

#include "Login.h"

#include "LoginUI.h"

#include "UserCollection.h"

#include"User.h"

// Function : Login(UserCollection& userCollection)

// Description: Login class의 생성자

// Parameters : UserCollection& userCollection

// Return Value : void

//

// Created: 2017/5/29 2:15 am

// Author: 조현정

//

Login::Login(UserCollection& userCollection)

{

userCollectionPointer = &userCollection;

}

// Function : bool callLoginOnOff(string inputID, string inputPwd, list<User>::iterator allUser)

// Description: 조건을 체크(아이디,비밀번호)하여 조건에 부합한다면 userCollection의 setLoginUser를 호출하는 함수.

// Parameters : string inputID, string inputPwd, list<User>::iterator allUser

// Return Value : bool

//

// Created: 2017/5/29 2:20am

// Author: 조현정

//

bool Login::callLoginOnOff(string inputID, string inputPwd, list<User>::iterator allUser)

{

checkExists(inputID, inputPwd);

if (isExists)

return 1;

else {

userCollectionPointer->setLoginUser(inputID);

return 0;

}

}

// Function : checkExists(string idOfUser)

// Description: 아이디와 비밀번호의 정보가 일치하는 객체가 있는지 찾는 함수

// Parameters : string idOfUser

// Return Value : void

//

// Created: 2017/5/29 2:23 am

// Author: 조현정

//

void Login::checkExists(string inputID, string inputPwd)

{

list<User>::iterator iter = userCollectionPointer->findUser(inputID);

if (iter == userCollectionPointer->findEnd())

isExists = true;

else {

//ID 찾은 경우

if (iter->getUserPwd() == inputPwd)

isExists = false;

else

isExists = true;

}

}

// Function : callJoinUIStartInterface(Login& logoin)

// Description: LoginUI의 인터페이스를 반환해주는 함수

// Parameters : Login& logoin

// Return Value : LoginUI\*

//

// Created: 2017/5/29 2:30 am

// Author: 조현정

//

LoginUI \* Login::callLoginUIStartInterface(Login & login)

{

return LoginUI::startInterface(login);

}

// Function : ~Login()

// Description: Login class의 소멸자

// Parameters : void

// Return Value : void

//

// Created: 2017/5/29 2:15 am

// Author: 조현정

//

Login::~Login()

{

}

**LeaveGroupUI.h**

#pragma once

#include <iostream>

#include <string>

#include <list>

//Class: LeaveGroupUI

//Description: LeaveGroupUI 헤더

//created: 2017/5/30 22:06pm

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

class LeaveGroup;

class User;

class LeaveGroupUI

{

private:

LeaveGroup\* leaveGroupPointer;

public:

bool confirmLeaveGroup(list<User>::iterator loginUser,string &joinedGroupName);

static LeaveGroupUI\* startInterface(LeaveGroup &leaveGroup);

LeaveGroupUI(LeaveGroup &leaveGroup);

~LeaveGroupUI();

};

**LeaveGroupUI.cpp**

#include "LeaveGroupUI.h"

#include "LeaveGroup.h"

//Class: LeaveGroupUI

//Description: LeaveGroupUI 구현

//created: 2017/5/30 22:10pm

//Author: 장현석

//mail: jang8018@naver.com

// Function : LeaveGroupUI(LeaveGroup &leaveGroup)

// Description: 컨트롤의 주소를 받아 포인터를 지정하고 UI를 생성합니다.

// Parameters : LeaveGroup &leaveGroup

// Return Value : void

//

// Created: 2017/5/30 22:10pm

// Author: 장현석

//

LeaveGroupUI::LeaveGroupUI(LeaveGroup &leaveGroup)

{

leaveGroupPointer = &leaveGroup;

}

LeaveGroupUI::~LeaveGroupUI()

{

}

// Function : confirmLeaveGroup(list<User>::iterator loginUser, string &joinedGroupName)

// Description: 컨트롤에 그룹탈퇴함수 호출을 요청합니다.

// Parameters : list<User>::iterator loginUser, string &joinedGroupName

// Return Value : bool

//

// Created: 2017/5/30 22:14pm

// Author: 장현석

//

bool LeaveGroupUI::confirmLeaveGroup(list<User>::iterator loginUser, string &joinedGroupName)

{

return leaveGroupPointer->leaveGroup(loginUser, joinedGroupName);

}

// Function : startInterface(LeaveGroup & leaveGroup)

// Description: 컨트롤을받아 UI생성자를 호출합니다.

// Parameters : LeaveGroup & leaveGroup

// Return Value : LeaveGroupUI \*

//

// Created: 2017/5/30 22:17pm

// Author: 장현석

//

LeaveGroupUI \* LeaveGroupUI::startInterface(LeaveGroup & leaveGroup)

{

LeaveGroupUI\* newPointer = new LeaveGroupUI(leaveGroup);

return newPointer;

}

**LeaveGroup.h**

#pragma once

#include <iostream>

#include <string>

#include <list>

//Class: LeaveGroup

//Description: LeaveGroup 헤더

//created: 2017/5/30 21:57pm

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

class LeaveGroupUI;

class User;

class LeaveGroup

{

private:

public:

LeaveGroupUI\* callJoinGroupUIStartInterface(LeaveGroup &leaveGroup);

bool leaveGroup(list<User>::iterator loginUser, string &joinedGroupName);

LeaveGroup();

~LeaveGroup();

};

**LeaveGroup.cpp**

#include "LeaveGroup.h"

#include "LeaveGroupUI.h"

#include "User.h"

//Class: LeaveGroup

//Description: LeaveGroup 구현

//created: 2017/5/30 22:00pm

//Author: 장현석

//mail: jang8018@naver.com

LeaveGroup::LeaveGroup()

{

}

LeaveGroup::~LeaveGroup()

{

}

// Function : callJoinGroupUIStartInterface(LeaveGroup & leaveGroup)

// Description: 컨트롤의 주소를 전달하여 그룹탈퇴UI 인터페이스를 호출합니다.

// Parameters : LeaveGroup & leaveGroup

// Return Value : LeaveGroupUI \*

//

// Created: 2017/5/30 22:00pm

// Author: 장현석

//

LeaveGroupUI \* LeaveGroup::callJoinGroupUIStartInterface(LeaveGroup & leaveGroup)

{

return LeaveGroupUI::startInterface(leaveGroup);

}

// Function : leaveGroup(list<User>::iterator loginUser, string &joinedGroupName)

// Description: 로그인한 유저가 속한 그룹을 탈퇴합니다.

// Parameters : list<User>::iterator loginUser, string &joinedGroupName

// Return Value : bool

//

// Created: 2017/5/30 22:03pm

// Author: 장현석

//

bool LeaveGroup::leaveGroup(list<User>::iterator loginUser, string &joinedGroupName)

{

if (loginUser->getIsLeader())

{

return true;

}

else

{

joinedGroupName = loginUser->getJoinedGroup();

loginUser->withdrawGroup();

return false;

}

}

**JoinUI.h**

#pragma once

#include<string>

#include<list>

#include<iostream>

using namespace std;

class Join;

class User;

// Class : JoinUI

// Description: 회원가입 할 때의 boundary 클래스 역할을 하는 클래스

// Created: 2017/5/29 7:20 pm

// Author: 조현정

// mail: guswjd8718@naver.com

//

class JoinUI

{

private:

Join\* joinPointer;

public:

JoinUI(Join& join);

~JoinUI();

static JoinUI\* startInterface(Join& join);

bool enterUserInfo(string newName, string newPersonalNumber, string newAddress, string newID, string newPwd);

};

**JoinUI.cpp**

#include "JoinUI.h"

#include "Join.h"

// Function : JoinUI(Join& join)

// Description: JoinUI class의 생성자

// Parameters : Join& join

// Return Value : void

//

// Created: 2017/5/29 7:22 pm

// Author: 조현정

//

JoinUI::JoinUI(Join& join)

{

joinPointer = &join;

}

// Function : JoinUI()

// Description: JoinUI class의 소멸자

// Parameters : void

// Return Value : void

//

// Created: 2017/5/29 7:24 pm

// Author: 조현정

//

JoinUI::~JoinUI()

{

}

// Function : JoinUI\* startInterface(Join& join)

// Description: JoinUI 객체를 생성해주는 함수.

// Parameters : Join& join

// Return Value : JoinUI\*

//

// Created: 2017/5/29 7:26 pm

// Author: 조현정

//

JoinUI \* JoinUI::startInterface(Join& join)

{

JoinUI\* newPointer = new JoinUI(join);

return newPointer;

}

// Function : bool enterUserInfo(string newName, string newPersonalNumber, string newAddress, string newID, string newPwd)

// Description: Join 클래스에게 user에게 입력받은 정보를 전달해주고, 회원가입 성공 여부를 반환받는다.

// Parameters : string newName, string newPersonalNumber, string newAddress, string newID, string newPwd

// Return Value : bool

//

// Created: 2017/5/29 7:30 pm

// Author: 조현정

//

bool JoinUI::enterUserInfo(string newName, string newPersonalNumber, string newAddress, string newID, string newPwd)

{

return joinPointer->callCreateUser(newName,newPersonalNumber, newAddress, newID, newPwd);

}

**Join.h**

#pragma once

#include<string>

#include<iostream>

#include<list>

using namespace std;

class UserCollection;

class JoinUI;

// Class : Join

// Description: 회원가입 할 때의 control 클래스 역할을 하는 클래스

// Created: 2017/5/29 7:00 pm

// Author: 조현정

// mail: guswjd8718@naver.com

//

class Join

{

private:

UserCollection\* userCollectionPointer;

JoinUI\* joinUIPointer;

bool isExists;

public:

Join(UserCollection& userCollection);

~Join();

JoinUI\* callJoinUIStartInterface(Join& join);

bool callCreateUser(string newName, string newPersonalNumber, string newAddress, string newID, string newPwd);

void checkExists(string idOfUser);

};

**Join.cpp**

#include "Join.h"

#include "JoinUI.h"

#include"UserCollection.h"

#include"User.h"

// Function : Join(UserCollection& userCollection)

// Description: Join class의 생성자

// Parameters : UserCollection& userCollection

// Return Value : void

//

// Created: 2017/5/29 7:5 pm

// Author: 조현정

//

Join::Join(UserCollection& userCollection)

{

userCollectionPointer = &userCollection;

}

// Function : ~Join()

// Description: Join class의 소멸자

// Parameters : void

// Return Value : void

//

// Created: 2017/5/29 7:7 pm

// Author: 조현정

//

Join::~Join()

{

}

// Function : bool callCreateUser(string newName, string newPersonalNumber, string newAddress, string newID, string newPwd)

// Description: 조건을 체크(중복되는 아이디가 아닌지)하여 조건에 부합한다면 userCollection의 addUser를 호출하는 함수.

// Parameters : string newName, string newPersonalNumber, string newAddress, string newID, string newPwd

// Return Value : bool

//

// Created: 2017/5/29 7:10 pm

// Author: 조현정

//

bool Join::callCreateUser(string newName, string newPersonalNumber, string newAddress, string newID, string newPwd)

{

checkExists(newID);

if (isExists)

return 1;

else {

userCollectionPointer->addUser(newName, newPersonalNumber,newAddress, newID, newPwd);

return 0;

}

}

// Function : checkExists(string idOfUser)

// Description: 중복되는 아이디가 있는지 체크해주는 함수

// Parameters : string idOfUser

// Return Value : void

//

// Created: 2017/5/29 7:15 pm

// Author: 조현정

//

void Join::checkExists(string idOfUser)

{

list<User>::iterator iter = userCollectionPointer->findUser(idOfUser);

if (iter == userCollectionPointer->findEnd())

isExists = false;

else

isExists = true;

}

// Function : callJoinUIStartInterface(Join& join)

// Description: JoinUI의 인터페이스를 반환해주는 함수

// Parameters : Join& join

// Return Value : JoinUI\*

//

// Created: 2017/5/29 7:18 pm

// Author: 조현정

//

JoinUI\* Join::callJoinUIStartInterface(Join& join)

{

return JoinUI::startInterface(join);

}

**JoinGroupUI.h**

#pragma once

#include <iostream>

#include <string>

#include <list>

//Class: JoinGroupUI

//Description: JoinGroupUI 헤더

//created: 2017/5/26 21:28pm

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

class JoinGroup;

class User;

class JoinGroupUI

{

private:

JoinGroup\* joinGroupPointer;

public:

bool selectGroup(string nameOfGroup, list<User>::iterator loginUser);

static JoinGroupUI\* startInterface(JoinGroup &joinGroup);

JoinGroupUI(JoinGroup &joinGroup);

~JoinGroupUI();

};

**JoinGroupUI.cpp**

#include "JoinGroupUI.h"

#include "JoinGroup.h"

//Class: JoinGroupUI

//Description: JoinGroupUI 구현

//created: 2017/5/26 21:30pm

//Author: 장현석

//mail: jang8018@naver.com

// Function : JoinGroupUI(JoinGroup &joinGroup)

// Description: 그룹가입 컨트롤의 주소를 받아 포인터를 지정하고, UI를 생성합니다.

// Parameters : JoinGroup &joinGroup

// Return Value : void

//

// Created: 2017/5/26 21:30pm

// Author: 장현석

//

JoinGroupUI::JoinGroupUI(JoinGroup &joinGroup)

{

joinGroupPointer = &joinGroup;

}

JoinGroupUI::~JoinGroupUI()

{

}

// Function : selectGroup(string nameOfGroup, list<User>::iterator loginUser)

// Description: 선택한 그룹의 이름을 입력받아 컨트롤의 그룹가입을 콜합니다.

// Parameters : string nameOfGroup, list<User>::iterator loginUser

// Return Value : bool

//

// Created: 2017/5/26 21:31pm

// Author: 장현석

//

bool JoinGroupUI::selectGroup(string nameOfGroup, list<User>::iterator loginUser)

{

return joinGroupPointer->joinGroup(nameOfGroup, loginUser);

}

// Function : startInterface(JoinGroup & joinGroup)

// Description: 컨트롤주소를 받아 UI의 인터페이스를 생성하고 해당 주소값을 반환합니다.

// Parameters : JoinGroup & joinGroup

// Return Value : JoinGroupUI \*

//

// Created: 2017/5/26 21:35pm

// Author: 장현석

//

JoinGroupUI \* JoinGroupUI::startInterface(JoinGroup & joinGroup)

{

JoinGroupUI\* newPointer = new JoinGroupUI(joinGroup);

return newPointer;

}

**JoinGroup.h**

#pragma once

#include <iostream>

#include <string>

#include <list>

//Class: JoinGroup

//Description: JoinGroup 헤더

//created: 2017/5/26 21:08pm

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

class UserCollection;

class GroupCollection;

class JoinGroupUI;

class User;

class JoinGroup

{

private:

UserCollection\* userCollectionPointer;

GroupCollection\* groupCollectionPointer;

public:

JoinGroupUI\* callJoinGroupUIStartInterface(JoinGroup &joinGroup);

bool joinGroup(string group\_name, list<User>::iterator loginUser);

JoinGroup(GroupCollection &groupCollection,UserCollection &userCollection);

~JoinGroup();

};

**JoinGroup.cpp**

#include "JoinGroup.h"

#include "JoinGroupUI.h"

#include "UserCollection.h"

#include "GroupCollection.h"

#include "User.h"

#include "Group.h"

//Class: JoinGroup

//Description: JoinGroup 구현

//created: 2017/5/26 21:10pm

//Author: 장현석

//mail: jang8018@naver.com

// Function : JoinGroup(GroupCollection &groupCollection, UserCollection &userCollection)

// Description: 그룹콜렉션과 유저콜렉션주소를 받아 포인터를 지정하고, 컨트롤을 생성합니다.

// Parameters : GroupCollection &groupCollection, UserCollection &userCollection

// Return Value : void

//

// Created: 2017/5/26 21:10pm

// Author: 장현석

//

JoinGroup::JoinGroup(GroupCollection &groupCollection, UserCollection &userCollection)

{

groupCollectionPointer = &groupCollection;

userCollectionPointer = &userCollection;

}

JoinGroup::~JoinGroup()

{

}

// Function : callJoinGroupUIStartInterface(JoinGroup & joinGroup)

// Description: 그룹가입 컨트롤을 받아와서 UI생성을 콜합니다.

// Parameters : JoinGroup & joinGroup

// Return Value : JoinGroupUI \*

//

// Created: 2017/5/26 21:12pm

// Author: 장현석

//

JoinGroupUI \* JoinGroup::callJoinGroupUIStartInterface(JoinGroup & joinGroup)

{

return JoinGroupUI::startInterface(joinGroup);

}

// Function : joinGroup(string group\_name, list<User>::iterator loginUser)

// Description: 로그인중인 유저의 반복자와 그룹이름을 받아와 그룹에 가입시킵니다.

// Parameters : string group\_name, list<User>::iterator loginUser

// Return Value : bool

//

// Created: 2017/5/26 21:13pm

// Author: 장현석

//

bool JoinGroup::joinGroup(string group\_name, list<User>::iterator loginUser)

{

list<Group>::iterator iter = groupCollectionPointer->findGroup(group\_name);

if (iter == groupCollectionPointer->findEnd())

{

return true;

}

else

{

loginUser->setJoinedGroup(group\_name);

return false;

}

}

**Group.h**

#pragma once

#include <iostream>

#include <string>

#include <list>

//Class: Group

//Description: Group 헤더 작성

//created: 2017/5/25 19:50pm

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

class Group;

class UserCollection;

class Group {

private:

string groupName;

public:

Group(string nameOfGroup);

~Group();

static Group\* createGroup(string nameOfGroup);

void deleteGroup();

string getGroupName();

void setGroupName(string inputName);

};

**Group.cpp**

#include "Group.h"

#include "GroupCollection.h"

//Class: Group

//Description: Group 구현

//created: 2017/5/25 20:00pm

//Author: 장현석

//mail: jang8018@naver.com

// Function : Group(string nameOfGroup)

// Description: 그룹이름을 받아 그룹을 만듭니다.

// Parameters : CreateGroup &creatGroup

// Return Value : void

//

// Created: 2017/5/25 20:01pm

// Author: 장현석

//

Group::Group(string nameOfGroup)

{

groupName = nameOfGroup;

}

Group::~Group()

{

}

// Function : createGroup(string nameOfGroup)

// Description: 그룹이름을 받아 생성자를 호출한 후, 포인터를 반환합니다.

// Parameters : string nameOfGroup

// Return Value : Group\*

//

// Created: 2017/5/25 20:05pm

// Author: 장현석

//

Group\* Group::createGroup(string nameOfGroup)

{

Group\* newGroup = new Group(nameOfGroup);

return newGroup;

}

// Function : deleteGroup()

// Description: 그룹을 삭제합니다.

// Parameters : none

// Return Value : void

//

// Created: 2017/5/25 20:06pm

// Author: 장현석

//

void Group::deleteGroup()

{

Group::~Group();

}

// Function : getGroupName()

// Description: 그룹이름을 받아옵니다.

// Parameters : none

// Return Value : string

//

// Created: 2017/5/25 20:08pm

// Author: 장현석

//

string Group::getGroupName()

{

return groupName;

}

// Function : setGroupName(string inputName)

// Description: 그룹이름을 바꿉니다.

// Parameters : string inputName

// Return Value : void

//

// Created: 2017/5/25 20:10pm

// Author: 장현석

//

void Group::setGroupName(string inputName)

{

groupName = inputName;

}

**GroupCollection.h**

#pragma once

#include <iostream>

#include <string>

#include <list>

//Class: GroupCollection

//Description: GroupCollection 헤더

//created: 2017/5/25 20:25pm

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

class Group;

class GroupCollection {

private:

list<Group> GroupList;

list<Group>::iterator groupListIterator;

public:

list<Group>::iterator findFirst();

list<Group>::iterator getNext(list<Group>::iterator &inputIter);

list<Group>::iterator findEnd();

list<Group>::iterator findGroup(string nameOfGroup);

void addGroup(string nameOfGroup);

void removeGroup(string nameOfGroup);

GroupCollection();

~GroupCollection();

};

**GroupCollection.cpp**

#include "Group.h"

#include "GroupCollection.h"

//Class: GroupCollection

//Description: GroupCollection 구현

//created: 2017/5/25 20:28pm

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

GroupCollection::GroupCollection() {}

GroupCollection::~GroupCollection(){}

// Function : findFirst()

// Description: 그룹리스트의 첫번째 반복자를 반환합니다

// Parameters : none

// Return Value : list<Group>::iterator

//

// Created: 2017/5/25 20:30pm

// Author: 장현석

//

list<Group>::iterator GroupCollection::findFirst()

{

groupListIterator = GroupList.begin();

return groupListIterator;

}

// Function : findEnd()

// Description: 그룹리스트의 마지막 반복자를 반환합니다

// Parameters : none

// Return Value : list<Group>::iterator

//

// Created: 2017/5/25 20:31pm

// Author: 장현석

//

list<Group>::iterator GroupCollection::findEnd()

{

groupListIterator = GroupList.end();

return groupListIterator;

}

// Function : getNext(list<Group>::iterator &inputIter)

// Description: 그룹리스트의 반복자를 다음으로 넘겨줍니다

// Parameters : list<Group>::iterator &inputIter

// Return Value : list<Group>::iterator

//

// Created: 2017/5/25 20:32pm

// Author: 장현석

//

list<Group>::iterator GroupCollection::getNext(list<Group>::iterator &inputIter)

{

return inputIter++;

}

// Function : findGroup(string nameOfGroup)

// Description: 그룹이름을 입력받아 해당그룹위치의 반복자를 반환합니다.

// Parameters : string nameOfGroup

// Return Value : list<Group>::iterator

//

// Created: 2017/5/25 20:40pm

// Author: 장현석

//

list<Group>::iterator GroupCollection::findGroup(string nameOfGroup)

{

groupListIterator = GroupList.begin();

bool success = false; //성공여부

//리스트 검색

for (groupListIterator = GroupList.begin(); groupListIterator != GroupList.end(); groupListIterator++)

{

if (groupListIterator->getGroupName() == nameOfGroup)

{

success = true;

break;

}

}

return groupListIterator;

}

// Function : addGroup(string nameOfGroup)

// Description: 그룹리스트에 그룹을 생성하고 추가합니다.

// Parameters : string nameOfGroup

// Return Value : void

//

// Created: 2017/5/25 20:45pm

// Author: 장현석

//

void GroupCollection::addGroup(string nameOfGroup)

{

GroupList.push\_back(\*Group::createGroup(nameOfGroup));

}

// Function : removeGroup(string nameOfGroup)

// Description: 그룹이름을 입력받아 그룹 리스트에서 그룹을 삭제합니다.

// Parameters : string nameOfGroup

// Return Value : void

//

// Created: 2017/5/25 20:47pm

// Author: 장현석

//

void GroupCollection::removeGroup(string nameOfGroup)

{

groupListIterator = GroupList.begin();

bool find = false; //성공여부

//리스트에서 삭제과정

for (groupListIterator = GroupList.begin(); groupListIterator != GroupList.end(); groupListIterator++)

{

if (groupListIterator->getGroupName() == nameOfGroup)

{

groupListIterator->deleteGroup();

groupListIterator = GroupList.erase(groupListIterator);

find = true;

break;

}

}

groupListIterator = GroupList.begin();

}

**CurrentVoteListUI.h**

#pragma once

#include<iostream>

#include<string>

#include<list>

using namespace std;

//Class: currentVoteListUI

//Description: currentVoteListUI 클래스

//created: 2017/5/24 13:30pm

//Author: 홍유남

//mail: bcdass@naver.com

class CurrentVoteList;

class Group;

class CurrentVoteListUI

{

private:

CurrentVoteList\* currentVoteListPointer;

public:

CurrentVoteListUI(CurrentVoteList &currentVoteList);

~CurrentVoteListUI();

static CurrentVoteListUI\* startInterface(CurrentVoteList& currentVoteList);

void confirmView(string\* subject, int\* items, string\* startTime, string\* endTime, string\* belongsTo);

};

**CurrentVoteListUI.cpp**

#include "CurrentVoteList.h"

CurrentVoteListUI::CurrentVoteListUI(CurrentVoteList &currentVoteList)

{

currentVoteListPointer = &currentVoteList;

}

CurrentVoteListUI::~CurrentVoteListUI()

{

}

// Function : startInterface

// Description: CurrentVoteList(컨트롤 클래스)의 명령을 받아 UI를 생성

// Parameters : CurrentVoteList&

// Return Value : CurrentVoteList\*

// Created: 2017/5/24 13:30pm

// Author: 홍유남

//

CurrentVoteListUI \* CurrentVoteListUI::startInterface(CurrentVoteList & currentVoteList)

{

CurrentVoteListUI\* newPointer = new CurrentVoteListUI(currentVoteList);

return newPointer;

}

// Function : confirmView

// Description: Vote 정보값을 보여줌

// Parameters : string\* int\* string\* string\* string\*

// Return Value :

// Created: 2017/5/24 13:30pm

// Author: 홍유남

//

void CurrentVoteListUI::confirmView(string \* subject, int \* items, string \* startTime, string \* endTime,string \* belongsTo)

{

currentVoteListPointer->showVoteList(subject, items, startTime, endTime, belongsTo);

}

**CurrentVoteList.h**

#pragma once

#include <string>

#include <list>

//Class: currentVoteList

//Description: currentVoteList 클래스

//created: 2017/5/24 13:30pm

//Author: 홍유남

//mail: bcdass@naver.com

using namespace std;

class CurrentVoteListUI;

class VoteCollection;

class GroupCollection;

class CurrentVoteList

{

private:

CurrentVoteListUI\* currentVoteListUIPointer;

VoteCollection\* voteCollectionPointer;

public:

CurrentVoteList(VoteCollection &voteCollection);

~CurrentVoteList();

void showVoteList(string\* subject, int\* items, string\* startTime, string\* endTime, string\* belongsTo);

CurrentVoteListUI\* callCurrentVoteListUIStartInterface(CurrentVoteList &currentVoteList);

CurrentVoteListUI\* getCurrentVoteListUIPointer();

};

**CurrentVoteList.cpp**

#include "CurrentVoteListUI.h" #include "CurrentVoteList.h"

#include "CurrentVoteListUI.h"

#include "Vote.h"

#include "Group.h"

#include "VoteCollection.h"

#include"GroupCollection.h"

CurrentVoteList::CurrentVoteList(VoteCollection &voteCollection)

{

voteCollectionPointer = &voteCollection;

}

CurrentVoteList::~CurrentVoteList()

{

}

// Function : showVoteList

// Description: Vote의 정보 전달을 위한 함수

// Parameters : string\* int\* string\* string\* string\*

// Return Value :

// Created: 2017/5/24 13:30pm

// Author: 홍유남

//

void CurrentVoteList::showVoteList(string \* subject, int \* items, string \* startTime, string \* endTime, string \* belongsTo)

{

list<Vote>::iterator iter = voteCollectionPointer->findFirst();

list<Vote>::iterator iterEnd = voteCollectionPointer->findEnd();

int i = 0;

while (iter != iterEnd)

{

subject[i] = iter->getSubject();

items[i] = iter->getItems();

startTime[i] = iter->getStartTime();

endTime[i] = iter->getEndTime();

belongsTo[i++] = iter->getBelongsTo();

voteCollectionPointer->getNext(iter);

}

}

CurrentVoteListUI \* CurrentVoteList::callCurrentVoteListUIStartInterface(CurrentVoteList & currentVoteList)

{

return CurrentVoteListUI::startInterface(currentVoteList);

}

CurrentVoteListUI \* CurrentVoteList::getCurrentVoteListUIPointer()

{

return currentVoteListUIPointer;

}

**CreateVoteUI.h**

#pragma once

#include<iostream>

#include<list>

#include<string>

using namespace std;

class Vote;

class CreateVote;

//Class: CreateVoteUI

//Description: 투표 제안 기능(createVote)의 boundary 클래스 역할을 수행함.

//created: 2017/6/1 15:00pm

//Author: 김윤영

//mail: kyy01114@hanmail.net

class CreateVoteUI

{

private:

CreateVote\* createVotePointer;

public:

static CreateVoteUI\* startInterface(CreateVote &createVote);

//Function: static CreateVoteUI\* startInterface(CreateVote &createVote);

//Description: UI는 control class에서 실행된다.

//Parameters: CreateVote &createVote - 투표 제안 기능(createVote)의 control class

//Author: 김윤영

//Created: 2017/6/1 15:00pm

bool inputDate(string newSubject,int numberOfItems,string newStartTIme, string newEndTime,string groupName);

//Function: bool inputDate(string newSubject,int numberOfItems,string newStartTIme, string newEndTime,string groupName);

//Description: 필요한 데이터를 control로 넘기는 기능을 수행한다.

//Parameters: string newSubject,int numberOfItems,string newStartTIme, string newEndTime,string groupName == 투표 생성시 필요한 data들

//Return Value: error value, 성공인지 error인지 구분.

//Author: 김윤영

//Created: 2017/6/1 15:00pm

CreateVoteUI(CreateVote &createVote);

~CreateVoteUI();

//Function: CreateVoteUI(CreateVote &createVote); ~CreateVoteUI();

//Description: 생성자와 소멸자, UI는 control에 의해 생성된다.

//Parameters: CreateVote &createVote - 투표 제안 기능(createVote)의 control class

//Author: 김윤영

//Created: 2017/6/1 15:00pm

};

**CreateVoteUI.cpp**

#include "CreateVoteUI.h"

#include "CreateVote.h"

CreateVoteUI::CreateVoteUI(CreateVote & createVote)

{

createVotePointer = &createVote;

}

CreateVoteUI::~CreateVoteUI()

{

}

CreateVoteUI \* CreateVoteUI::startInterface(CreateVote & createVote)

{

CreateVoteUI\* newPointer = new CreateVoteUI(createVote);

return newPointer;

}

bool CreateVoteUI::inputDate(string newSubject, int numberOfItems, string newStartTIme, string newEndTime, string groupName)

{

return createVotePointer->createVote(newSubject, numberOfItems, newStartTIme, newEndTime,groupName);

}

**CreateVote.h**

#pragma once

#include<iostream>

#include<list>

#include<string>

using namespace std;

class VoteCollection;

class CreateVoteUI;

//Class: CreateVote

//Description: 투표 제안 기능(createVote)의 contorl 클래스 역할을 수행함.

//created: 2017/6/1 15:00pm

//Author: 김윤영

//mail: kyy01114@hanmail.net

class CreateVote

{

private:

VoteCollection\* voteCollectionPointer;

CreateVoteUI\* createVoteUIPointer;

bool isExist;

public:

bool createVote(string newSubject,int numberOfItems, string newStartTime,string newEndTime, string groupName);

//Function: bool createVote(string newSubject,int numberOfItems, string newStartTime,string newEndTime, string groupName);

//Description: createvote를 위한 input를 votecollection으로 전달해준다.

//Parameters: (string newSubject,int numberOfItems, string newStartTime,string newEndTime, string groupName

//Author: 김윤영

//Created: 2017/6/1 15:00pm

CreateVoteUI\* callCreateVoteUIStartInterface(CreateVote &createVote);

//Function: CreateVoteUI\* callCreateVoteUIStartInterface(CreateVote &createVote);

//Description: control class는 입력 받기전 UI를 생성해야된다.

//Parameters: CreateVote &createVote - 투표 제안 기능(createVote)의 control class

//Author: 김윤영

//Created: 2017/6/1 15:00pm

CreateVote(VoteCollection &voteCollection);

~CreateVote();

//Function: CreateVote(VoteCollection &voteCollection); ~CreateVote();

//Description: 생성자와 소멸자, 투표 제안 기능(createVote) control class는 votecollection의 레퍼런스를 가지고 있어야한다.

//Parameters: VoteCollection &voteCollection - 투표 collection

//Author: 김윤영

//Created: 2017/6/1 15:00pm

};

**CreateVote.cpp**

#include "CreateVote.h"

#include "CreateVoteUI.h"

#include "VoteCollection.h"

CreateVote::CreateVote(VoteCollection &voteCollection)

{

voteCollectionPointer = &voteCollection;

}

CreateVote::~CreateVote()

{

}

bool CreateVote::createVote(string newSubject, int numberOfItems, string newStartTime, string newEndTime, string groupName)

{

// 누락된 항목이 있을 경우

if (newSubject.size() == 0 && numberOfItems == 0 && newStartTime.size() == 0 && newEndTime.size() == 0)

return true;

else {

voteCollectionPointer->addVote(newSubject, numberOfItems, newStartTime, newEndTime,groupName);

return false;

}

}

CreateVoteUI \* CreateVote::callCreateVoteUIStartInterface(CreateVote & createVote)

{

return CreateVoteUI::startInterface(createVote);

}

**CreateGroupUI.h**

#pragma once

#include <iostream>

#include <string>

#include <list>

//Class: CreateGroupUI

//Description: CreateGroupUI 헤더

//created: 2017/5/25 09:10am

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

class CreateGroup;

class User;

class CreateGroupUI

{

private:

CreateGroup\* createGroupPointer;

public:

static CreateGroupUI\* startInterface(CreateGroup &creatGroup);

bool setGroupName(string nameOfGroup, list<User>::iterator loginUser);

CreateGroupUI(CreateGroup &createGroup);

~CreateGroupUI();

};

**CreateGroupUI.cpp**

#include "CreateGroup.h"

#include "CreateGroupUI.h"

//Class: CreateGroupUI

//Description: CreateGroupUI 구현

//created: 2017/5/25 09:15am

//Author: 장현석

//mail: jang8018@naver.com

// Function : CreateGroupUI(CreateGroup &createGroup)

// Description: 그룹생성 컨트롤주소를받아 포인터를 할당하고, UI를 생성합니다.

// Parameters : CreateGroup &createGroup

// Return Value : void

//

// Created: 2017/5/25 09:20am

// Author: 장현석

//

CreateGroupUI::CreateGroupUI(CreateGroup &createGroup)

{

createGroupPointer = &createGroup;

}

CreateGroupUI::~CreateGroupUI()

{

}

// Function : startInterface(CreateGroup &creatGroup)

// Description: 컨트롤을 받아 CreateUI클래스의 인터페이스를 생성합니다.

// Parameters : CreateGroup &creatGroup

// Return Value : CreateGroupUI\*

//

// Created: 2017/5/25 09:20am

// Author: 장현석

//

CreateGroupUI\* CreateGroupUI::startInterface(CreateGroup &creatGroup)

{

CreateGroupUI\* newPointer = new CreateGroupUI(creatGroup);

return newPointer;

}

// Function : setGroupName(string nameOfGroup, list<User>::iterator loginUser)

// Description: 유저의 반복자와 그룹이름명을 받아 callCreateGroup을 불러줍니다.

// Parameters : CreateGroup &creatGroup

// Return Value : string nameOfGroup, list<User>::iterator loginUser

//

// Created: 2017/5/25 09:25am

// Author: 장현석

//

bool CreateGroupUI::setGroupName(string nameOfGroup, list<User>::iterator loginUser)

{

return createGroupPointer->callCreateGroup(nameOfGroup, loginUser);

}

**CreateGroup.h**

#pragma once

#include <iostream>

#include <string>

#include <list>

//Class: CreateGroup

//Description: CreateGroup 헤더

//created: 2017/5/25 08:57am

//Author: 장현석

//mail: jang8018@naver.com

using namespace std;

class GroupCollection;

class CreateGroupUI;

class User;

class CreateGroup

{

private:

GroupCollection\* groupColletionPointer;

//CreateGroupUI\* createGroupUIPointer;

bool isExists;

public:

bool callCreateGroup(string nameOfGroup, list<User>::iterator loginUser);

void checkExists(string nameOfGroup);

CreateGroup(GroupCollection &groupCollection);

~CreateGroup();

CreateGroupUI\* callCreateGroupUIStartInterface(CreateGroup &createGroup);

};

**CreateGroup.cpp**

#include "GroupCollection.h"

#include "CreateGroup.h"

#include "CreateGroupUI.h"

#include "Group.h"

#include "User.h"

//Class: CreateGroup

//Description: CreateGroup 구현

//created: 2017/5/25 09:00am

//Author: 장현석

//mail: jang8018@naver.com

// Function : CreateGroup(GroupCollection &groupCollection)

// Description: 그룹 콜렉션주소를 포인터에 할당합니다

// Parameters : GroupCollection &groupCollection

// Return Value : void

//

// Created: 2017/5/25 09:03am

// Author: 장현석

//

CreateGroup::CreateGroup(GroupCollection &groupCollection)

{

groupColletionPointer = &groupCollection;

}

CreateGroup::~CreateGroup()

{

}

// Function : callCreateGroup(string nameOfGroup, list<User>::iterator loginUser)

// Description: 그룹 이름을 받아서 중복을 체크하고, 생성합니다. 중복이면 에러값을 반환합니다.

// Parameters : string nameOfGroup

// list<User>::iterator loginUser

// Return Value : bool

//

// Created: 2017/5/25 09:04am

// Author: 장현석

//

bool CreateGroup::callCreateGroup(string nameOfGroup, list<User>::iterator loginUser)

{

if (loginUser->getJoinedGroup() == "none")

{

checkExists(nameOfGroup);

if (isExists)

{

return 1;

}

else

{

groupColletionPointer->addGroup(nameOfGroup);

loginUser->setGroupLeader();

loginUser->setJoinedGroup(nameOfGroup);

return 0;

}

}

else

return 1;

}

// Function : checkExists(string nameOfGroup)

// Description: 반복자를 받아와서 중복인지 판단합니다.

// Parameters : string nameOfGroup

// Return Value : void

//

// Created: 2017/5/25 09:10am

// Author: 장현석

//

void CreateGroup::checkExists(string nameOfGroup)

{

list<Group>::iterator iter = groupColletionPointer->findGroup(nameOfGroup);

if (iter == groupColletionPointer->findEnd())

isExists = false;

else

isExists = true;

}

// Function : callCreateGroupUIStartInterface(CreateGroup &createGroup)

// Description: UI생성을하고, 컨트롤의 ref를 넘겨줍니다.

// Parameters : CreateGroup &createGroup

// Return Value : CreateGroupUI\*

//

// Created: 2017/5/25 09:08am

// Author: 장현석

//

CreateGroupUI\* CreateGroup::callCreateGroupUIStartInterface(CreateGroup &createGroup)

{

return CreateGroupUI::startInterface(createGroup);

}

**ControlAndCollection.h**

#pragma once

#include <iostream>

//Class: ControlAndCollection

//Description: ControlAndCollection 헤더. 컨트롤과 콜렉션 저장을위한 클래스입니다.

//created: 2017/5/30 20:50pm

//Author: 장현석

//mail: jang8018@naver.com

class UserCollection;

class GroupCollection;

class VoteCollection;

class ShowAllGroup;

class CreateGroup;

class ShowJoinedGroup;

class LeaveGroup;

class JoinGroup;

class CreateVote;

class VoteFor;

class CurrentVoteList;

class PreVoteList;

class PostVoteList;

class Join;

class Login;

class Logout;

class Withdrwal;

using namespace std;

class ControlAndCollection

{

private:

VoteCollection \*currentVoteCollection;

VoteCollection \*postVoteCollection;

VoteCollection \*preVoteCollection;

UserCollection \*userCollection;

GroupCollection \*groupCollection;

ShowAllGroup \*showAllGroup;

CreateGroup \*createGroup;

ShowJoinedGroup \*showJoinedGroup;

LeaveGroup \*leaveGroup;

JoinGroup \*joinGroup;

CreateVote \*createVote;

VoteFor \*voteFor;

CurrentVoteList \*currentVoteList;

PreVoteList \*preVoteList;

PostVoteList \*postVoteList;

Join \*join;

Login \*login;

Logout \*logout;

Withdrwal \*withdrwal;

ControlAndCollection();

ControlAndCollection(const ControlAndCollection& other);

~ControlAndCollection();

static ControlAndCollection\* instance;

public:

VoteCollection \*getCurrentVoteCollection();

VoteCollection \*getPostVoteCollection();

VoteCollection \*getPreVoteCollection();

UserCollection \*getUserCollection();

GroupCollection\* getGroupCollection();

ShowAllGroup\* getShowAllGroup();

CreateGroup \*getCreateGroup();

ShowJoinedGroup \*getShowJoinedGroup();

LeaveGroup \*getLeaveGroup();

JoinGroup \*getJoinGroup();

CreateVote \*getCreateVote();

VoteFor \*getVoteFor();

CurrentVoteList \*getCurrentVoteList();

PreVoteList \*getPreVoteList();

PostVoteList \*getPostVoteList();

Join \*getJoin();

Login \*getLogin();

Logout \*getLogout();

Withdrwal \*getWithdrwal();

static ControlAndCollection\* GetInstance();

};

**ControlAndCollection,cpp**

#include "ControlAndCollection.h"

#include "ShowAllGroup.h"

#include "UserCollection.h"

#include "GroupCollection.h"

#include "CreateGroup.h"

#include "LeaveGroup.h"

#include "ShowJoinedGroup.h"

#include "VoteCollection.h"

#include "Join.h"

#include "JoinGroup.h"

#include "CreateVote.h"

#include"VoteFor.h"

#include "Login.h"

#include "Logout.h"

#include "Withdrwal.h"

#include "CurrentVoteList.h"

#include "PreVoteList.h"

#include "PostVoteList.h"

//Class: ControlAndCollection

//Description: ControlAndCollection 구현. 컨트롤과 콜렉션 저장을위한 클래스입니다.

//created: 2017/5/30 21:00pm

//Author: 장현석

//mail: jang8018@naver.com

//스태틱 초기화입니다.

ControlAndCollection\* ControlAndCollection::instance = nullptr;

// Function : ControlAndCollection()

// Description: ControlAndCollection생성자 입니다.

// Parameters : none

// Return Value : void

//

// Created: 2017/6/01 18:00pm

// Author: 장현석

//

ControlAndCollection::ControlAndCollection()

{

groupCollection = new GroupCollection();

showAllGroup = new ShowAllGroup(\*groupCollection);

currentVoteCollection = new VoteCollection();

postVoteCollection = new VoteCollection();

preVoteCollection = new VoteCollection();

groupCollection = new GroupCollection();

userCollection = new UserCollection();

showAllGroup = new ShowAllGroup(\*groupCollection);

createGroup = new CreateGroup(\*groupCollection);

showJoinedGroup = new ShowJoinedGroup;

leaveGroup = new LeaveGroup;

joinGroup = new JoinGroup(\*groupCollection, \*userCollection);

createVote = new CreateVote(\*postVoteCollection);

voteFor = new VoteFor(\*currentVoteCollection);

currentVoteList = new CurrentVoteList(\*currentVoteCollection);

preVoteList = new PreVoteList(\*preVoteCollection);

postVoteList = new PostVoteList(\*postVoteCollection);

join = new Join(\*userCollection);

login = new Login(\*userCollection);

logout = new Logout(\*userCollection);

withdrwal = new Withdrwal(\*userCollection);

}

ControlAndCollection::~ControlAndCollection()

{

}

// Function : 애트리뷰트 get함수

// Description: private에있는 애트리뷰트를 반환합니다

// Parameters : none

// Return Value : 각각의 값

//

// Created: 2017/6/01 18:30pm

// Author: 장현석

//

VoteCollection \* ControlAndCollection::getCurrentVoteCollection()

{

return currentVoteCollection;

}

VoteCollection \* ControlAndCollection::getPostVoteCollection()

{

return postVoteCollection;

}

VoteCollection \* ControlAndCollection::getPreVoteCollection()

{

return preVoteCollection;

}

UserCollection \* ControlAndCollection::getUserCollection()

{

return userCollection;

}

GroupCollection\* ControlAndCollection::getGroupCollection()

{

return groupCollection;

}

ShowAllGroup\* ControlAndCollection::getShowAllGroup()

{

return showAllGroup;

}

CreateGroup \* ControlAndCollection::getCreateGroup()

{

return createGroup;

}

ShowJoinedGroup \* ControlAndCollection::getShowJoinedGroup()

{

return showJoinedGroup;

}

LeaveGroup \* ControlAndCollection::getLeaveGroup()

{

return leaveGroup;

}

JoinGroup \* ControlAndCollection::getJoinGroup()

{

return joinGroup;

}

CreateVote \* ControlAndCollection::getCreateVote()

{

return createVote;

}

VoteFor \* ControlAndCollection::getVoteFor()

{

return voteFor;

}

CurrentVoteList \* ControlAndCollection::getCurrentVoteList()

{

return currentVoteList;

}

PreVoteList \* ControlAndCollection::getPreVoteList()

{

return preVoteList;

}

PostVoteList \* ControlAndCollection::getPostVoteList()

{

return postVoteList;

}

Join \* ControlAndCollection::getJoin()

{

return join;

}

Login \* ControlAndCollection::getLogin()

{

return login;

}

Logout \* ControlAndCollection::getLogout()

{

return logout;

}

Withdrwal \* ControlAndCollection::getWithdrwal()

{

return withdrwal;

}

// Function : GetInstance()

// Description: ControlAndCollection싱글턴 인스턴스를 생성하거나 반환합니다.

// Parameters : none

// Return Value : ControlAndCollection \*

//

// Created: 2017/5/30 21:15pm

// Author: 장현석

//

ControlAndCollection \* ControlAndCollection::GetInstance()

{

if (!instance) {

instance = new ControlAndCollection();

}

return ControlAndCollection::instance;

}