클라이언트 서버 프로그래밍

중간 레포트

제출일	2021. 10. 31 (일)	**제출자**	60191902 김정우
과목	클라이언트 서버 프로그래밍		
담당교수	김정호 교수님		

Homework Assignment

보고서 및 논문 윤리 서약

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.

2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.

3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.

4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁받지도 않겠습니다.

나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

학　　과 : 응용소프트웨어학과

과　　목 : 클라이언트 서버 프로그래밍

담당교수 : 김정호

학번 : 60191902

이름 : 김정우　　(서명)

<목차>

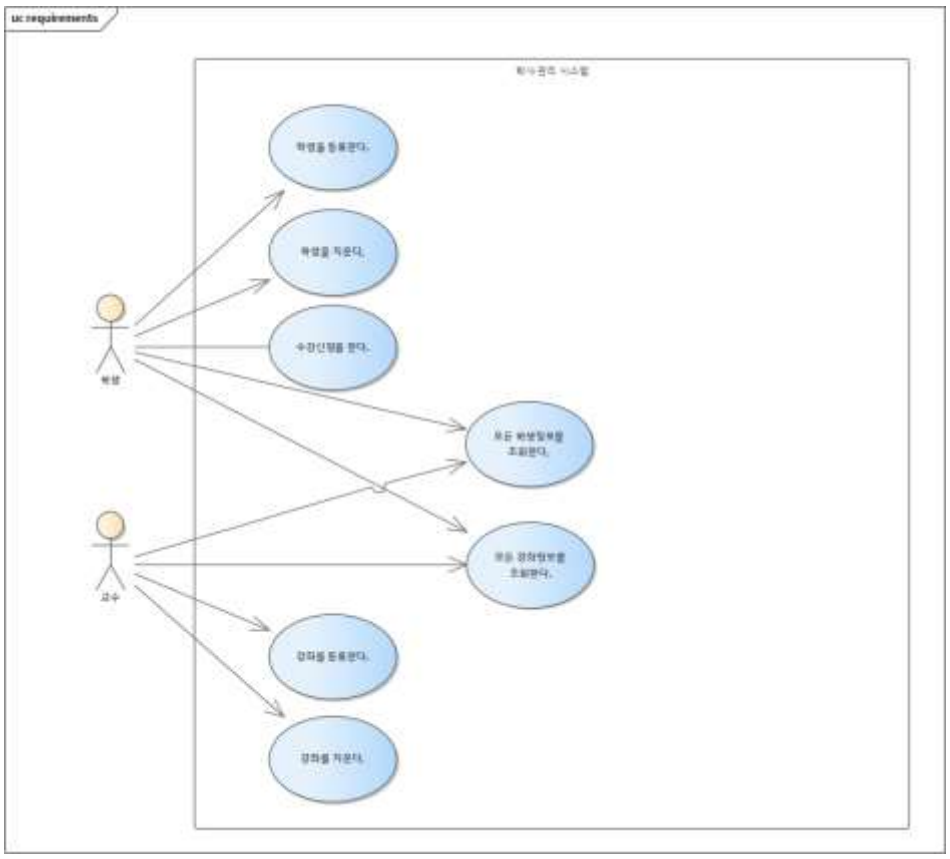
Ⅰ. 개요

본 보고서는 멀티 프로세스를 이용한 학사 관리 시스템 구축에 대한 내용이 기술되어 있다. 이를 위해 첫번째로, 분석을 통해 기능 요구사항, 비기능 요구사항을 정리한다. 두번째로, 클라이언트, 서버 설계를 클래스 다이어그램과 클래스 명세서로 정리한다. 세번째로, RDB설계에 대해 구축 목표 – 요구사항 – Conceptual Design - Logical Database Design – Physical Design 순으로 정리한다. 네번째로, 본격적인 프로그램 구현코드를 통해 시스템이 어떻게 동작하는지 기본흐름, 대안흐름, 예외흐름으로 세세하게 알아보고 마지막 다섯번째로는 결론을 도출한다.

Ⅱ. 분석

가. 기능[1] 요구사항

<Usecase Diagram>



1. 학생, 교수는 모든 학생정보를 조회한다.
2. 학생, 교수는 모든 강좌정보를 조회한다.
3. 학생은 학생을 등록한다.
4. 학생은 학생을 지운다.
5. 교수는 과목을 등록한다.

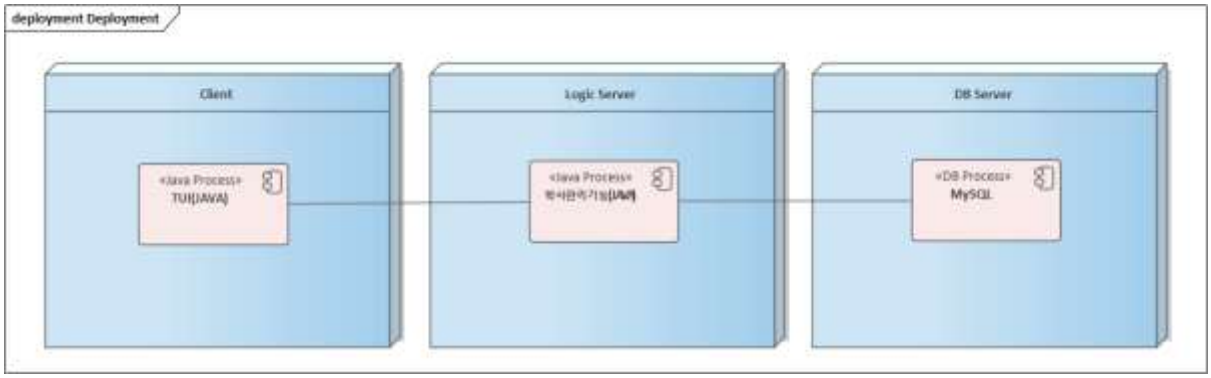
[1] 입력값 대비 출력값의 변화, 동사로 표현되며 기능이 안되면 시스템의 존재이유가 사라진다.

6. 교수는 과목을 지운다.
7. 학생은 수강 신청을 한다.

나. 비기능[2] 요구사항

1. 멀티 프로세스[3]를 활용해 하나의 프로세스가 비정상적으로 종료되더라도 다른 프로세스가 영향을 받지 않게 함으로써 유지 보수성을 향상한다.
2. gRPC를 이용해 빠른 속도로 통신함으로써 성능을 향상시킨다.
3. Enum을 활용해 단순한 코드를 만들고, 가독성이 좋아짐으로써 유지보수성을 향상시킨다.
4. JPA를 활용해 생산성, 유지보수, 성능을 향상시킨다.

Ⅲ. 설계서

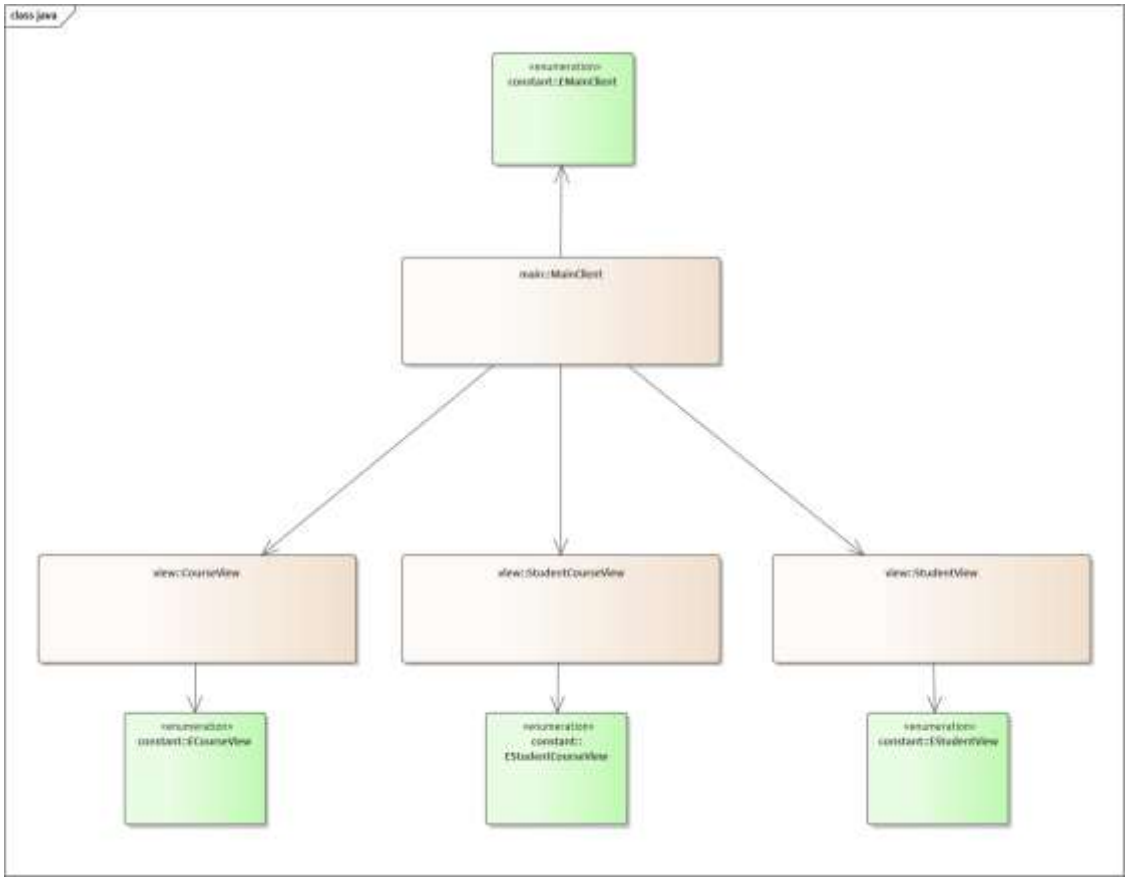


위 그림은 Client, Logic Server, DB Server 시스템간의 전체적인 흐름을 EA를 통해 나타낸 것이다. 아래 목차를 통해 클라이언트, 서버, DB 각각의 설계에 대해 알아본다.

[2] 기능을 제약하는 요소들, 부사로 표현되며 How much, How many의 문제이다.

[3] 소프트웨어 시스템에서 프로세스가 여러 개 발생하는 것

가. 클라이언트 설계서



위 그림은 클라이언트 설계의 전체적인 구조를 Class Diagram으로 나타낸 것이다.

1. MainClient Class

A. 속성

Name	DataType	Description
studentServerBlockingStub	StudentServiceBlockingStub	서버의 StudentService를 이용하기 위해 사용하는 속성
courseServerBlockingStub	CourseServiceBlockingStub	서버의 CourseService를 이용하기 위해 사용하는 속성
studentCourseServiceBlockingStub	StudentCourseServiceBlockingStub	서버의 StudentCourseService를 이용하기 위해 사용하는 속성
studentView	StudentView	StudentService와 관련된 화면을 보여주기 위해 사용되는 속성

courseView	CourseView	CourseService와 관련된 화면을 보여주기 위해 사용되는 속성
studentCourseView	StudentCourseView	StudentCourseView와 관련된 화면을 보여주기 위해 사용되는 속성

B. 기능

Name	Parameter	Return Value	Description
setConnection	void	ManagedChannel	서버와 연결설정하는 기능
printMenu	void	void	사용자에게 메뉴를 보여주기 위한 기능
choiceMenu	void	void	사용자의 메뉴선택을 입력받는 기능
validationCorrectMenuInput	String	boolean	사용자의 메뉴 선택이 올바른지 검증하는 기능

2. EMainClient Enum

A. 속성

Name	DataType	Description
content	String	열거형의 내용을 String으로 저장하기 위한 속성

B. 기능

Name	Parameter	Return value	Description
getContent	void	String	열거형의 String 내용을 가져오는 기능

C. 열거 속성

Name	Content
ePortNumber	"localhost:50050"
one	void
two	void
three	void
four	void

five	void
six	void
seven	void
eight	void
eSwitchChoiceFail	"올바르지 않은 선택입니다."
eColon	" : "
eMenuStar	"**********"
eMenuTitle	"********** 학사관리 시스템**********"
eMenuGuide	"**********번호를 영어로 입력하세요**********"
eMenuOne	"**********(one) : 모든 학생 정보 불러오기**********"
eMenuTwo	"**********(two) : 모든 강좌 정보 불러오기**********"
eMenuThree	"**********(three) : 학생 등록하기**********"
eMenuFour	"**********(four) : 학생 지우기**********"
eMenuFive	"**********(five) : 과목 등록하기**********"
eMenuSix	"**********(six) : 과목 지우기**********"
eMenuSeven	"**********(seven) : 수강 신청**********"
eMenuEight	"**********(eight) : 나가기**********"
eEnterTheMenuAgain	"메뉴 입력이 잘못되었습니다. 메뉴를 다시 입력해 주세요."
eFalse	false

3. StudentView Class

A. 속성

Name	DataType	Description
studentServerBlockingStub	StudentServiceBlockingStub	서버의 StudentService를 이용하기 위해 사용하는 속성

B. 기능

Name	Parameter	Return value	Description
allStudentsDataResponse	void	void	서버에서 모든 학생 정보를 가져오는 기

			능
showAllStudents	List<AllStudentsDataResponse.Student>	void	서버에서 가져온 학생정보를 화면에 보여주는 기능
receiveAddStudentData	BufferedReader	AddStudentRequest	추가하기 위한 학생정보를 사용자에게 입력받는 기능
addStudentDataResponse	BufferedReader	void	입력한 학생정보를 이용해 서버의 학생 추가하기 메서드를 실행하기 위한 기능
receiveDeleteStudentData	BufferedReader	DeleteStudentRequest	사용자에게 지우고자 하는 학생정보를 입력받는 기능
deleteStudentDataResponse	BufferedReader	void	입력한 학생정보를 이용해 서버의 학생 지우기 메서드를 실행하기 위한 기능
validationAddStudentData	String, String, String	void	입력한 학생정보가 숫자를 포함하는지, 아닌지 여부 체크하는 기능
validationStudentName	String	void	숫자를 포함하면 예외처리하는 기능
validationStudentNumber	String	void	숫자만 포함하지 않는다면 예외처리
validationStudentMajor	String	void	숫자를 포함하면 예외처리

4. EStudentView Enum

A. 속성

Name	DataType	Description

content	String	열거형의 내용을 String으로 저장하기 위한 속성
number	int	열거형의 내용을 int로 저장하기 위한 속성

B. 기능

Name	Parameter	Return value	Description
getContent	void	String	열거형의 String 내용을 가져오는 기능
getNumber	void	int	열거형의 int 내용을 가져오는 기능

C. 열거 속성

Name	Content
eStudentName	"학생 명 : "
eStudentNumber	"학생 번호 : "
eStudentMajor	"학생 전공 : "
eStudentApplicationForCourseNumber	"학생 수강 과목 : "
eAddStudentSuccessMessage	"ADD STUDENT SUCCESS"
eAddStudentFailMessage	"ADD STUDENT FAIL"
eMenuStar	"**********"
eMenuAddStudentGuide	"**********학생 정보 입력하기**********"
eMenuStudentNameGuide	"학생 이름을 입력하세요."
eMenuStudentNumberGuide	"학번을 입력하세요."
eMenuStudentMajorGuide	"전공을 입력하세요."
eDeleteStudentSuccessMessage	"DELETE STUDENT SUCCESS"
eDeleteStudentFailMessage	"DELETE STUDENT FAIL"
eMenuDeleteStudentGuide	"**********지울 학생 정보 입력하기**********"
eSpacing	"₩n"
eNull	null
eZero	0
eMatchContainNumber	".*[0-9].*"
eStudentNameMatchMessage	"학생이름에 숫자를 포함할 수 없습니다."
eStudentNumberMatchMessage	"학생번호에 숫자만 포함할 수 있습니다."

eMajorMatchMessage	"전공에 숫자를 포함할 수 없습니다."
eMatchOnlyNumber	"[0-9]+"

5. CourseView Class

A. 속성

Name	DataType	Description
courseServerBlockingStub	CourseServiceBlockingStub	서버의 CourseService를 이용하기 위해 사용하는 속성

B. 기능

Name	Parameter	Return value	Description
allCoursesDataResponse	void	void	서버에서 모든 강좌 정보를 가져오는 기능
showAllCourses	List<AllCoursesDataResponse.Course>	void	서버에서 가져온 강좌정보를 화면에 보여주는 기능
receiveAddCourseData	BufferedReader	AddCourseRequest	추가하기 위한 강좌 정보를 사용자에게 입력받는 기능
addCourseDataResponse	BufferedReader	void	입력한 강좌정보를 이용해 서버의 강좌 추가하기 메서드를 실행하기 위한 기능
ReceiveDeleteCourseData	BufferedReader	DeleteCourseRequest	사용자에게 지우고자 하는 강좌정보를 입력받는 기능
deleteCourseDataResponse	BufferedReader	void	입력한 강좌정보를 이용해 서버의 강좌 지우기 메서드를 실행하기 위한 기능

validationAddCourseData	String, String, String, List<String>	void	입력받은 강좌정보 각각에 대해 숫자 포함 하는지 아닌지의 검증하는 기능
validationCourseNumber	String	void	강좌번호가 숫자만 포함하는지 검증하는 기능
validationProfessorLastName	String	void	교수성이 숫자를 포함하는지 검증하는 기능
validationCourseName	String	void	강좌이름이 숫자를 포함하는지 검증하는 기능
validationAdvancedCourseNumbers	List<String>	void	선강좌번호가 숫자만 포함하는지 검증하는 기능

6. ECourseView Enum

A. 속성

Name	DataType	Description
content	String	열거형의 내용을 String으로 저장하기 위한 속성
number	int	열거형의 내용을 int로 저장하기 위한 속성

B. 기능

Name	Parameter	Return value	Description
getContent	void	String	열거형의 String 내용을 가져오는 기능
getNumber	void	int	열거형의 int 내용을 가져오는 기능

C. 열거 속성

Name	Content
eCourseNumber	"강좌 번호 : "
eCourseName	"강좌 명 : "
eProfessorLastName	"교수 성 : "
eAdvancedName	"강좌 선이수 과목 : "
eAddCourseSuccessMessage	"ADD COURSE SUCCESS"
eAddCourseFailMessage	"ADD COURSE FAIL"
eMenuStar	"**********"
eMenuAddCourseGuide	"**********강좌 정보 입력하기**********"
eMenuCourseNumberGuide	"강좌 번호를 입력하세요."
eMenuProfessorLastNameGuide	"교수 성을 입력하세요."
eMenuCourseNameGuide	"강좌 이름을 입력하세요."
eMenuAdvancedCourseNumberGuide	"선이수 강좌 번호를 입력하세요. 입력을 마치셨으면 x를 입력하세요."
eMenuDeleteCourseGuide	"**********지울 강좌 정보 입력하기**********"
eDeleteCourseSuccessMessage	"DELETE COURSE SUCCESS"
eDeleteCourseFailMessage	"DELETE COURSE FAIL"
eX	"x"
eSpacing	"₩n"
eNull	null
eZero	0
eMatchContainNumber	".*[0-9].*"
eCourseNumberMatchMessage	"강좌번호에 숫자만 포함되어야 합니다."
eProfessorLastNameMatchMessage	"교수 성에 숫자가 포함되서는 안됩니다."
eCourseNameMatchMessage	"강좌 이름에 숫자가 포함되서는 안됩니다."
eAdvancedCourseNumberMatchMessage	"선이수 강좌번호에 숫자만 포함되어야 합니다."
eMatchOnlyNumber	"[0-9]+"

7. StudentCourseView Class

A. 속성

Name	DataType	Description

studentCourseServiceBlockingStub	StudentCourseServiceBlockingStub	서버의 StudentCourseService를 이용하기 위해 사용하는 속성

B. 기능

Name	Parameter	Return value	Description
receiveApplicationForCourseRequest	BufferedReader	ApplicationForCourseRequest	사용자에게 수강신청에 필요한 정보들을 입력받는 기능
applicationForCourse	BufferedReader	void	입력받은 수강신청 정보를 이용해 서버의 수강신청하기 메서드를 실행하는 기능
validationApplicationForCourseData	String, String	void	입력받은 수강신청정보가 숫자만 포함하는지 검증하는 기능
validationStudentNumber	String	void	학생번호가 숫자만 포함하는지 검증하는 기능
validationCourseNumber	String	void	강좌번호가 숫자만 포함하는지 검증하는 기능

8. EStudentCourseView Enum

A. 속성

Name	DataType	Description
content	String	열거형의 내용을 String으로 저장하기 위한 속성

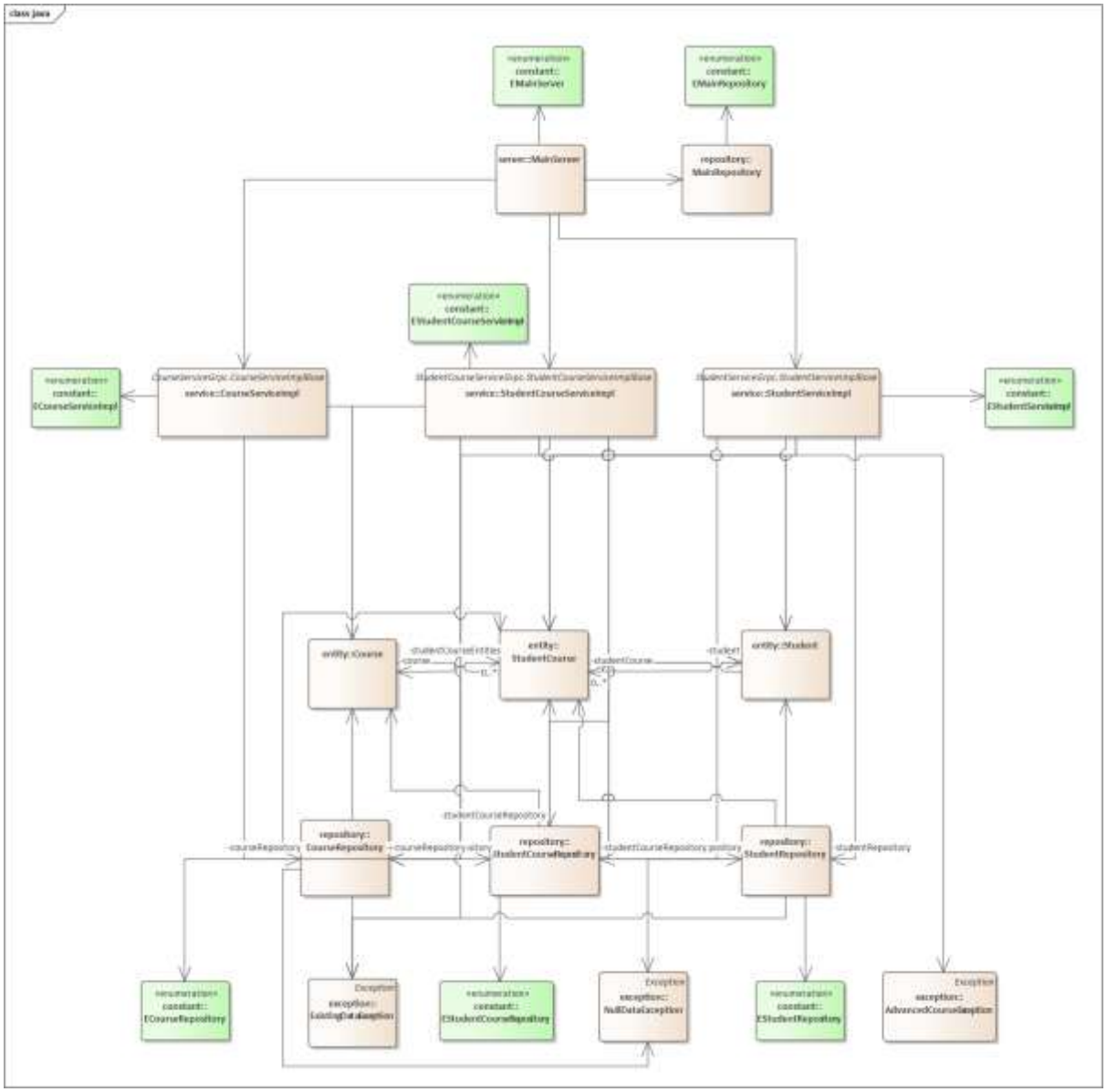
B. 기능

Name	Parameter	Return value	Description
getContent	void	String	열거형의 String 내용을 가져오는 기능

C. 열거 속성

Name	Content
eApplicationForCourseSuccessMessage	"APPLICATION FOR COURSE SUCCESS"
eApplicationForCourseFailMessage	"APPLICATION FOR COURSE FAIL"
eMenuStar	"**********"
eMenuApplicationForCourseGuide	"**********수강신청 정보 입력하기**********"
eMenuStudentNumberGuide	"학번을 입력하세요."
eMenuCourseNumberGuide	"강좌번호를 입력하세요."
eMatchContainNumber	".*[0-9].*"
eStudentNumberMatchMessage	"학생번호에 숫자만 포함할 수 있습니다."
eCourseNumberMatchMessage	"강좌번호에 숫자만 포함되어야 합니다."
eMatchOnlyNumber	"[0-9]+"

나. 서버 설계서



1. MainServer Class

A. 속성

Name	DataType	Description
mainRepository	MainRepository	데이터베이스의 연결을 위한 기능을 이용하기 위해 사용하는 속성
studentCourseRepository	StudentCourseRepository	서비스와 레포지토리를 연결하기 위한 속성
studentRepository	StudentRepository	서비스와 레포지토리를 연결하기 위

		한 속성
courseRepository	CourseRepository	서비스와 레포지토리를 연결하기 위한 속성

B. 기능

Name	Parameter	Return Value	Description
setConnection	void	void	서버의 연결설정을 위한 기능

2. EMainServer Enum

A. 속성

Name	DataType	Description
content	String	열거형의 내용을 String으로 저장하기 위한 속성
number	int	열거형의 내용을 int로 저장하기 위한 속성

B. 기능

Name	Parameter	Return value	Description
getContent	void	String	열거형의 String 내용을 가져오는 기능
getNumber	void	int	열거형의 int 내용을 가져오는 기능

C. 열거 속성

Name	Content
eServerStartSuccessMessage	"ACADEMIC SERVER STARTED ON PORT 50050"
eServerStartErrorMessage	"ACADEMIC SERVER DID NOT START"
eServerStartShutDownMessage	"ACADEMIC SERVER SHUT DOWN ON INTERRUPTED"
ePortNumber	50050

3. StudentServiceImpl Class

A. 속성

Name	DataType	Description
studentCourseRepository	StudentCourseRepository	서비스와 레포지토리를 연결하기 위한 속성
studentRepository	StudentRepository	서비스와 레포지토리를 연결하기 위한 속성
courseRepository	CourseRepository	서비스와 레포지토리를 연결하기 위한 속성

B. 기능

Name	Parameter	Return Value	Description
getAllStudentsData	EmptyRequest, StreamObserver<AllStudentsDataResponse>	void	모든 학생의 정보를 가져오기 위한 기능
setAllStudentsDataResponse	List<Student>	AllStudentsDataResponse	Response를 설정하기 위한 기능
addStudentData	AddStudentRequest, StreamObserver<IsCompletedResponse>	void	학생데이터를 추가하기 위한 기능
deleteStudentData	DeleteStudentRequest, StreamObserver<IsCompletedResponse>	void	학생데이터를 삭제하기 위한 기능
validationStudent	AddStudentRequest	void	학생데이터를 추가할 때 검증하는 기능
validationStudentNumber	DeleteStudentRequest	void	학생데이터를 삭제할 때 검증하는 기능

4. EStudentServiceImpl Enum

A. 속성

Name	DataType	Description
content	String	열거형의 내용을 String으로 저장하기

		위한 속성
number	int	열거형의 내용을 int로 저장하기 위한 속성
check	Boolean	열거형의 내용을 Boolean으로 저장하기 위한 속성

B. 기능

Name	Parameter	Return value	Description
getContent	void	String	열거형의 String 내용을 가져오는 기능
getNumber	void	int	열거형의 int 내용을 가져오는 기능
getBoolean	void	Boolean	열거형의 Boolean내용을 가져오는 기능

C. 열거 속성

Name	Content
eEmptyRequestStudentExceptionMessage	"THE STUDENT INPUT IS A EMPTY VALUE."
eEmptyRequestStudentNumberExceptionMessage	"THE STUDENT NUMBER IS A EMPTY VALUE."
eEmpty	""
eColon	" : "
eZero	0
eTrue	true
eFalse	false

5. CourseServiceImpl Class

A. 속성

Name	DataType	Description
courseRepository	CourseRepository	서비스와 레포지토리를 연결하기 위한 속성

B. 기능

Name	Parameter	Return Value	Description

getAllCoursesData	EmptyRequest, StreamObserver<AllCoursesDataResponse>	void	모든 강좌의 정보를 가져오기 위한 기능
setAllCoursesDataResponse	List<Course>	AllCoursesDataResponse	Response를 설정하기 위한 기능
addCourseData	AddCourseRequest, StreamObserver<IsCompletedResponse>	void	강좌데이터를 추가하기 위한 기능
deleteCourseData	DeleteCourseRequest, StreamObserver<IsCompletedResponse>	void	강좌데이터를 삭제하기 위한 기능
validationCourse	AddCourseRequest	void	강좌데이터를 추가할 때 검증하는 기능
validationCourseNumber	DeleteCourseRequest	void	강좌데이터를 삭제할 때 검증하는 기능

6. EcourseServiceImpl Enum

A. 속성

Name	DataType	Description
content	String	열거형의 내용을 String으로 저장하기 위한 속성
number	int	열거형의 내용을 int로 저장하기 위한 속성
check	Boolean	열거형의 내용을 Boolean으로 저장하기 위한 속성

B. 기능

Name	Parameter	Return value	Description
getContent	void	String	열거형의 String 내용을 가져오는 기능
getNumber	void	int	열거형의 int 내용을 가져오는 기능
getBoolean	void	Boolean	열거형의 Boolean내용을 가져오는 기능

C. 열거 속성

Name	Content
eEmptyRequestCourseExceptionMessage	"THE COURSE INPUT IS A EMPTY VALUE."
eEmptyRequestCourseNumberExceptionMessage	"THE COURSE NUMBER IS A EMPTY VALUE."
eEmpty	""
eColon	" : "
eZero	0
eTrue	true

7. StudentCourseServiceImpl

A. 속성

Name	DataType	Description
studentCourseRepository	StudentCourseRepository	서비스와 레포지토리를 연결하기 위한 속성
studentRepository	StudentRepository	서비스와 레포지토리를 연결하기 위한 속성
courseRepository	CourseRepository	서비스와 레포지토리를 연결하기 위한 속성

B. 기능

Name	Parameter	Return Value	Description
applicationForCourse	ApplicationForCourseRequest, StreamObserver<IsCompletedResponse>	void	수강 신청을 하기 위한 기능
validationCourseId	ApplicationForCourseRequest	void	강좌번호를 검증하기 위한 기능
validationStudentId	ApplicationForCourseRequest	void	학생번호를 검증하기 위한 기능
validationExistingCourse	Course, List<StudentCourse>	void	이미 듣고 있는 강좌인지 검증하는 기능
validationAdvancedCourse	Course, List<StudentCourse>	void	선이수강좌를 들었

			는지 검증하는 기능

8. EStudentServiceImpl Enum

A. 속성

Name	DataType	Description
content	String	열거형의 내용을 String으로 저장하기 위한 속성
number	int	열거형의 내용을 int로 저장하기 위한 속성
check	Boolean	열거형의 내용을 Boolean으로 저장하기 위한 속성

B. 기능

Name	Parameter	Return value	Description
getContent	void	String	열거형의 String 내용을 가져오는 기능
getNumber	void	int	열거형의 int 내용을 가져오는 기능
getBoolean	void	Boolean	열거형의 Boolean내용을 가져오는 기능

C. 열거 속성

Name	Content
eEmptyRequestCourseIdExceptionMessage	"THE COURSE ID'S INPUT IS A EMPTY VALUE."
eEmptyRequestStudentIdExceptionMessage	"THE STUDENT ID'S INPUT IS A EMPTY VALUE."
eAlreadyTakingCourseExceptionMessage	"THIS IS A COURSE YOU ARE ALREADY TAKING"
eTakeAdvancedCourseExceptionMessage	"YOU DIDN'T TAKE THE ADVANCED COURSE"
eEmpty	""
eColon	" : "
eZero	0
eTrue	true
eFalse	false

9. StudentRepository Class

A. 속성

Name	DataType	Description
emf	EntityManagerFactory	엔티티매니저들을 관리하기 위한 속성
em	EntityManager	엔티티매니저 속성

B. 기능

Name	Parameter	Return Value	Description
findAll	void	List<Student>	모든 학생 정보를 DB에서 가져오기 위한 기능
save	Student	Boolean	학생을 DB에 저장하기 위한 기능
deleteStudentByFindStudent	Student	Boolean	학생정보로 학생을 DB에서 지우기 위한 기능
findStudentByStudentNumber	String, Boolean	Student	학생 번호로 DB에서 학생을 찾기 위한 기능
createStudent	AddStudentRequest	Student	학생을 만들어서 DB에 저장하기 위한 기능
addStudentCourse	Student, StudentCourse	void	수강신청 테이블에 데이터를 추가하기 위한 기능

10. EstudentRepository Enum

A. 속성

Name	DataType	Description
content	String	열거형의 내용을 String으로 저장하기

		위한 속성
number	int	열거형의 내용을 int로 저장하기 위한 속성
check	Boolean	열거형의 내용을 Boolean으로 저장하기 위한 속성

B. 기능

Name	Parameter	Return value	Description
getContent	void	String	열거형의 String 내용을 가져오는 기능
getNumber	void	int	열거형의 int 내용을 가져오는 기능
getBoolean	void	Boolean	열거형의 Boolean내용을 가져오는 기능

C. 열거 속성

Name	Content
eFindAllStudentQuery	"select s from Student s"
eNoStudentDataExceptionMessage	"NO STUDENT DATA FOUND"
eFindStudentCourseByStudentQuery	select sc from StudentCourse sc where sc.student = :student"
eStudent	"student"
eFindStudentByStudentNumberQuery	"select s from Student s where s.studentNumber = :studentNumber"
eStudentNumber	"studentNumber"
eNoStudentDataByStudentNumberExceptionMessage	"NO STUDENT DATA FOUND BY STUDENT_NUMBER"
eManyStudentsDataByStudentNumberExceptionMessage	"THERE ARE SEVERAL STUDENTS WITH THE SAME STUDENT_NUMBER"
eAlreadyStudentNumberExceptionMessage	"THIS STUDENT NUMBER ALREADY EXISTS."
eOne	1
eZero	0
eTrue	true
eFalse	false

11. CourseRepository Class

A. 속성

Name	DataType	Description
emf	EntityManagerFactory	엔티티매니저들을 관리하기 위한 속성
em	EntityManager	엔티티매니저 속성
studentCourseRepository	StudentCourseRepository	StudentCourseRepository를 이용하기 위한 속성

B. 기능

Name	Parameter	Return Value	Description
findAll	void	List<Course>	모든 강좌 정보를 DB에서 가져오기 위한 기능
findCoursesByCourseNumber	ProtocolStringList	List<Course>	강좌번호로 DB에서 강좌들을 찾기 위한 기능
createCourse	AddCourseRequest	Course	강좌를 생성하고 DB에 저장하기 위한 기능
deleteCourseByCourseNumber	String	boolean	강좌번호로 DB에서 강좌를 삭제하기 위한 기능
deleteStudentCourseByCourse	List<StudentCourse>	void	강좌로 Student_Course테이블의 데이터를 지우기 위한 기능
deleteAdvancedCourseByCourse	Course	void	강좌로 선강좌테이블의 데이터를 지우기 위한 기능
findAllNoException	void	List<Course>	DB의 모든 강좌를 찾기 위한 기능
findCourseByCourseNumber	String	Course	강좌번호로 DB에서

			강좌들을 찾기위한 기능
addCourseWithAdvancedCourse	AddCourseRequest, List<Course>	Course	선강좌테이블에 데이터를 추가하고 강좌를 만들기 위한 기능

12. EcourseRepository Enum

A. 속성

Name	DataType	Description
content	String	열거형의 내용을 String으로 저장하기 위한 속성
number	int	열거형의 내용을 int로 저장하기 위한 속성

B. 기능

Name	Parameter	Return value	Description
getContent	void	String	열거형의 String 내용을 가져오는 기능
getNumber	void	int	열거형의 int 내용을 가져오는 기능

C. 열거 속성

Name	Content
eFindAllCourseQuery	"select c from Course c"
eNoCourseDataExceptionMessage	"NO COURSE DATA FOUND"
eFindCourseByCourseNumberQuery	"select c from Course c where c.courseNumber = :courseNumber"
eCourseNumber	"courseNumber"
eNoCourseDataByCourseNumberExceptionMessage	"NO COURSE DATA FOUND BY COURSE_NUMBER"
eDeleteStudentCourseDataWarningMessage	"WARNING! DELETE STUDENT'S COURSE DATA"
eDeleteAdvancedCourseDataWarningMessage	"WARNING! DELETE COURSE'S ADVANCED_COURSE DATA"
eFindCourseListByCourseNumberQuery	"select c from Course c where c.courseNumber = :courseNumber"

eManyCoursesByCourseNumberExceptionMessage	"THERE ARE SEVERAL COURSES WITH THE SAME COURSE_NUMBER"
eOne	1
eZero	0

13. StudentCourseRepository Class

A. 속성

Name	DataType	Description
emf	EntityManagerFactory	엔티티매니저들을 관리하기 위한 속성
em	EntityManager	엔티티매니저 속성

B. 기능

Name	Parameter	Return Value	Description
createStudentCourse	Course	StudentCourse	학생정보로 Student_Course테이블을 찾기 위한 기능
findStudentCourseByStudent	List<StudentCourse>	Student	학생정보로 Student_Course테이블의 데이터를 찾기 위한 기능
deleteStudentCourse	void	List<StudentCourse>	Student_Course테이블의 데이터를 지우기 위한 기능
findStudentCourseByCourse	List<StudentCourse>	Course	강좌정보로 Student_Course테이블의 데이터를 찾기 위한 기능

14. EStudentCourseRepository Enum

A. 속성

Name	DataType	Description
content	String	열거형의 내용을 String으로 저장하기 위한 속성

B. 기능

Name	Parameter	Return value	Description
getContent	void	String	열거형의 String 내용을 가져오는 기능

C. 열거 속성

Name	Content
eFindStudentCourseByStudentQuery	"select sc from StudentCourse sc where sc.student = :student"
eStudent	"student"
eFindStudentCourseByCourseQuery	"select sc from StudentCourse sc where sc.course = :course"
eCourse	"course"

15. MainRepository Class

A. 속성

Name	DataType	Description
emf	EntityManagerFactory	엔티티매니저들을 관리하기 위한 속성
em	EntityManager	엔티티매니저 속성

B. 기능

Name	Parameter	Return Value	Description
initDB	void	void	DB의 데이터를 초기화하기 위한 기능

16. EMainRepository Enum

A. 속성

Name	DataType	Description
content	String	열거형의 내용을 String으로 저장하기 위한 속성

B. 기능

Name	Parameter	Return value	Description
getContent	void	String	열거형의 String 내용을 가져오는 기능

C. 열거 속성

Name	Content
eAcademic	"academic"

다. 관계형 데이터베이스 설계서

<The project organization and its purpose>

- 학사관리 정보가 크게 증가하여 관리에 어려움, 따라서 학사관리를 쉽게 관리하기 위해 필요한 모든 정보들을 데이터베이스화 하고자 함

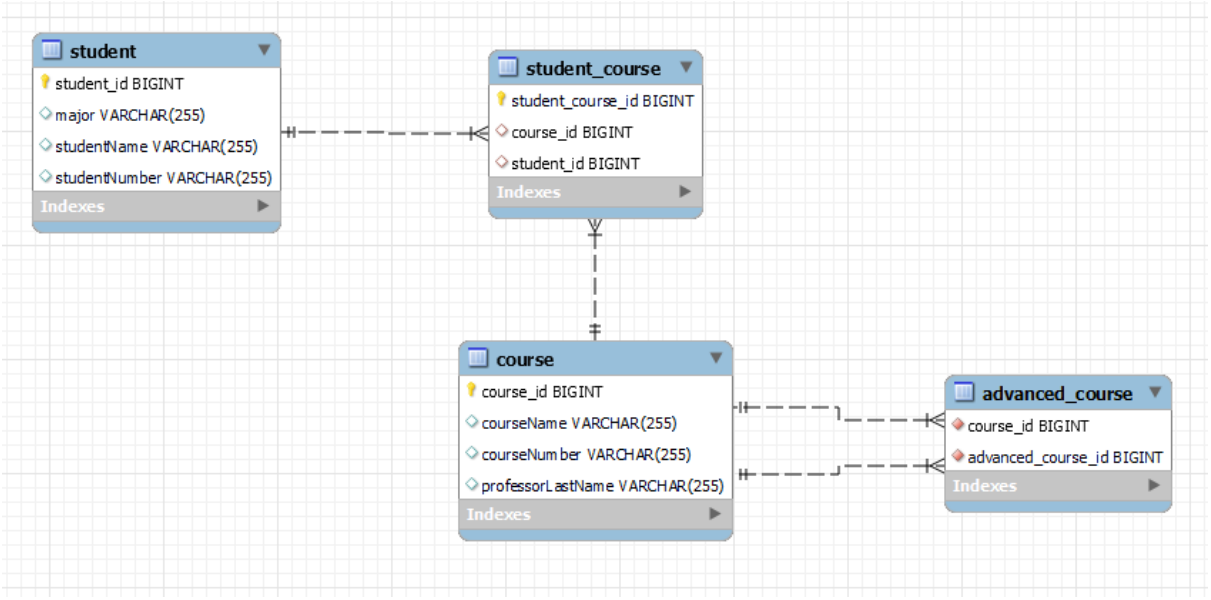
<User/Data requirements and business rules.>

- 각각의 학생은 student_id(identifier), major, studentName, studentNumber를 가진다.
- 각각의 강좌는 course_id(identifier), courseName, courseNumber, professorLastName을 가진다.
- 각각의 수강신청은 student_course_id(identifier), student_id, course_id를 가진다.
- 한 명의 학생은 수강신청을 여러 번 할 수 있다.
- 한 개의 강좌는 여러 개의 수강신청에 포함될 수 있다.

- 각각의 선이수강좌는 course_id, advanced_course_id를 가진다.

- 여러 개의 강좌는 여러 개의 선이수강좌를 가질 수 있다.

<Conceptual Design: ER-Diagram>



위의 그림은 학사 관리 데이터베이스의 ER-Diagram을 나타낸 것이다. student 테이블과 student_course테이블 관계는 1:M, student_course 테이블과 course 테이블 관계는 M:1, course 테이블과 advanced_course 테이블의 관계는 M:M이다.

<Logical Database design>

student(student_id, major, studentName, studentNumber)

course(course_id, courseName, courseNumber, professorLastName)

student_course(student_course_id, course_id, student_id)

Foreign key(course_id) references course(course_id)

Foreign key(student_id) references student(student_id)

advanced_course(course_id, advanced_course_id)

Foreign key(course_id) references course(course_id)

Foreign key(advanced_course_id) references course(course_id)

<Physical Design/Data dictionary>

Table Name : student

Attribute	Description	Data Type	Constraint
student_id	Primary key	BIGINT	Not null
major	Student's major	VARCHAR(255)	
studentName	Student's studentName	VARCHAR(255)	
studentNumber	Student's studentNumber	VARCHAR(255)	

Table Name : course

Attribute	Description	Data Type	Constraint
course_id	Primary key	BIGINT	Not null
courseName	Course's courseName	VARCHAR(255)	
courseNumber	Course's courseNumber	VARCHAR(255)	
professorLastName	Course's professorLastName	VARCHAR(255)	

Table Name : student_course

Attribute	Description	Data Type	Constraint
student_course_id	Primary key	BIGINT	Not null
course_id	ApplicationForCourse's course_id(Foreign key)	BIGINT	
student_id	ApplicationForCourse's student_id(Foreign key)	BIGINT	

Table Name : advanced_course

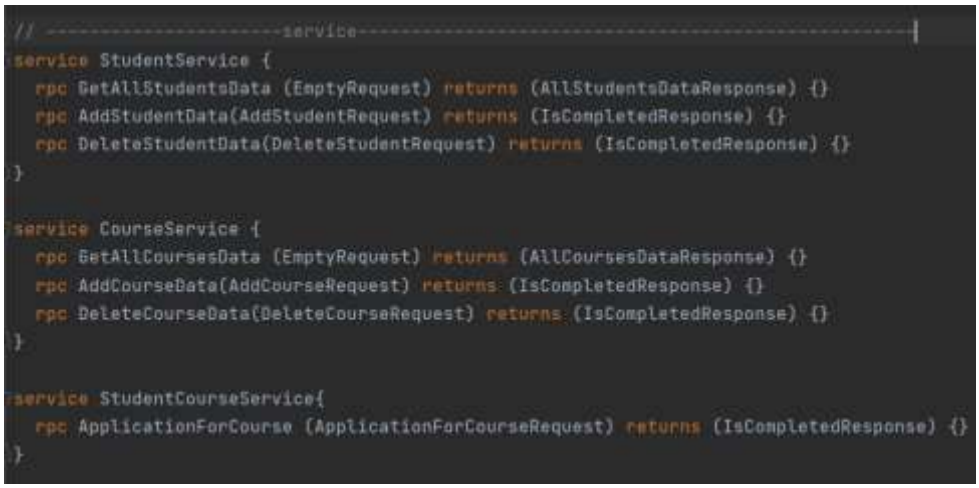
Attribute	Description	Data Type	Constraint
course_id	Course's id(Foreign key)	BIGINT	Not null
advanced_course_id	Advanced Course id(Foreign key)	BIGINT	Not null

Ⅳ. 프로그램 설명, 결과

가. Proto 설명

해당 프로그램은 gRPC로 구현되었으므로 academic.proto 파일을 통해 주요 서비스를 간략히 알아보고, 서비스를 이용하기 위한 Request, Response의 명세서를 살펴본다.

<Service>



위의 그림은 academic.proto 파일의 service를 정의한 부분이다. 총 3개의 서비스로 나뉘어 있다.

첫번째로 **StudentService**는 학생과 관련된 서비스를 제공하며, 3개의 메서드로 구성되어 있다. 모든 학생 정보를 가져오는 GetAllStudentsData 메서드, 학생 정보를 추가하는 AddStudentData 메서드, 학생 정보를 지우는 DeleteStudentData 메서드이다.

두번째로, **CourseService**는 강좌와 관련된 서비스를 제공하며, 3개의 메서드로 구성되어 있다. 모든 강좌 정보를 가져오는 GetAllCourseData 메서드, 강좌 정보를 추가하는 AddCourseData

메서드, 강좌 정보를 지우는 DeleteCourseData 메서드이다.

세번째로 **StudentCourseService**는 수강신청과 관련된 서비스를 제공하며 메서드이름은 ApplicationForCourse이다.

<Request>

```
// ---------------------request ----------
message EmptyRequest {
}

message AddStudentRequest {
  string studentNumber = 1;
  string studentName = 2;
  string major = 3;
}

message AddCourseRequest {
  string courseNumber = 1;
  string professorLastName = 2;
  string courseName = 3;
  repeated string advancedCourseNumber = 4;
}

message DeleteStudentRequest{
  string studentNumber = 1;
}

message DeleteCourseRequest{
  string courseNumber = 1;
}

message ApplicationForCourseRequest{
  string studentNumber = 1;
  string courseNumber = 2;
}
```

위의 그림은 academic.proto 파일의 request를 정의한 부분이다.

첫번째로, **EmptyRequest**는 null값을 보낼 때 사용한다. GetAllStudentsData 메서드, GetAllCourses 메서드의 요청 파리미터이다.

두번째로, **AddStudentRequest**는 AddStudentsData 메서드의 요청 파라미터이다.

StudentNumber는 학번, studentName은 학생이름, major는 전공을 의미한다.

세번째로, **AddCourseRequest**는 AddCourseData 메서드의 요청 파라미터이다.

CourseNumber는 강좌번호, professorLastName은 교수 성, courseName은 강좌 이름,

advancedCourseNumber는 선강좌번호인데 repeated를 이용하여 String타입의 변수를 반복적으로 입력할 수 있게 구현했다.

네번째로, **DeleteStudentRequest**는 DeleteStudentData의 요청 파라미터이다. StudentNumber는 학번을 의미한다.

다섯번째로, **DeleteCourseRequest**는 DeleteCourseData의 요청 파라미터이다.

CourserNumber는 강좌번호를 의미한다.

여섯번째로, **ApplicationForCourseRequest**는 ApplicationForCourseRequest의 요청 파라미터이다. studentNumber, courseNumber는 학생번호, 강좌번호를 의미한다.

<Response>

```
// ---------------------response ------------

message AllStudentsDataResponse {

  message Student {
    string studentNumber = 1;
    string studentName = 2;
    string major = 3;
    repeated string courseNumber = 4;
  }
  repeated Student students = 2;
}

message AllCoursesDataResponse {

  message Course {
    string courseNumber = 1;
    string professorLastName = 2;
    string courseName = 3;
    repeated string advancedCourseNumber = 4;
  }
  repeated Course courses = 2;
}

message IsCompletedResponse {
  bool isCompleted = 1;
}
```

위의 그림은 academic.proto 파일의 response를 정의한 부분이다.

첫번째로, **AllStudentDataRepsonse**는 GetAllStudentsData 메서드의 Retrun Value를 정의한

것이다. Student의 StudentNumber는 학생번호, studentName은 학생이름, major는 전공, courseNumber는 학생이 듣고 있는 강좌번호를 배열로 정의한 것이다. 이러한 Student 메시지를 배열로 정의한 것이 students이다.

두번째로, **AllCourseDataResponse**는 GetAllCoursesData 메서드의 Return Value를 정의한 것이다. Course의 courseNumber는 강좌번호, professorLastName은 교수 성, courseName은 강좌명, advancedCourseNumber는 선강좌번호를 배열로 정의한 것이다. 이러한 Course 메시지를 배열로 정의한 것이 courses이다.

세번째로, **IsCompletedReponse**는 AddStudentData, AddCourseData, DeleteStudentData, DeleteCourseData, ApplicationForCourse 메서드의 Retrun Value를 정의한 것이다.

IsCompleted는 boolean의 형태이며, 메서드가 올바르게 수행되었는지 여부를 나타낸다.

나. 기능별 설명, 결과

1. 모든 학생 정보를 조회한다.

[기본흐름1]

```
case one: studentView.allStudentsDataResponse(); break;
```

사용자는 'one'을 클라이언트 콘솔창에 입력해 학생 정보를 조회하는 메서드를 실행한다.

[대안흐름1]

만약, 사용자가 'one'이 아닌 오타를 입력했다면 아래와 같은 메시지와 함께 메뉴값을 다시 입력받을 수 있게 한다.

```
onee
메뉴 입력이 잘못되었습니다. 메뉴를 다시 입력해 주세요.

**************************************************
******************** 학사관리 시스템******************
**************************************************
**********************번호를 영어로 입력하세요****************
************(one) : 모든 학생 정보 불러오기**************
```

이때 사용하는 검증 메서드는 아래와 같다.

```
private boolean validationCorrectMenuInput(String inputMenu) {
    if(inputMenu.equals(EMainClient.one.getContent()) || inputMenu.equals(EMainClient.two.getContent())
            || inputMenu.equals(EMainClient.three.getContent()) || inputMenu.equals(EMainClient.four.getContent())
            || inputMenu.equals(EMainClient.five.getContent()) || inputMenu.equals(EMainClient.six.getContent())
            || inputMenu.equals(EMainClient.seven.getContent()) || (inputMenu.equals(EMainClient.eight.getContent()))){
        return true; } else{ System.out.println(EMainClient.eEnterTheMenuAgain.getContent());
        return false; } }
```

[예외흐름1]

만약, 서버가 닫혀있는데 'one'을 입력했다면 아래와 같은 예외처리를 통해 메세지를 전달하고 switch문을 다시 실행하도록 한다.

```
INFO: StatusRuntimeException : UNAVAILABLE: io exception
```

위의 그림은 서버가 닫혀있을 때 클라이언트의 예외처리 화면이다.

[기본흐름2]

```
public void allStudentsDataResponse(){
    showAllStudents(this.studentServerBlockingStub.getAllStudentsData( request: null).getStudentsList());
}
```

성공적이라면, StudentView클래스의allStudentsDataReponse메서드를 실행하고 이후 서버의 getAllStudentsData메서드를 실행한다.

```
@Override
public void getAllStudentsData(EmptyRequest empty, StreamObserver<AllStudentsDataResponse> responseObserver) {
    try {
        List<Student> result = studentRepository.findAll();
        AllStudentsDataResponse response = setAllStudentsDataResponse(result);
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    } catch (Exception e) {
        logger.info( msg: e.getClass().getSimpleName() + EStudentServiceImpl.eColon.getContent() + e.getMessage());
        Status status = Status.INTERNAL.withDescription(e.getMessage());
        responseObserver.onError(status.asRuntimeException());
    }
}
```

서버의 StudentServiceImpl 클래스의getAllStudentsData메서드이다. 이를 실행한다.

[기본흐름3]

```
public List<Student> findAll() throws NullDataException {
    List<Student> resultList = em.createQuery(EStudentRepository.eFindAllStudentQuery.getContent(), Student.class)
            .getResultList();
    if (resultList.size() == EStudentRepository.eZero.getNumber()) {
        throw new NullDataException(EStudentRepository.eNoStudentDataExceptionMessage.getContent()); }
    return resultList;
}
```

StudentRepository 클래스의 findAll메서드를 실행함으로써 데이터베이스의 모든 Student데이터를 가져온다.

[예외흐름2]

여기서 만약, 쿼리가 잘못된 문법으로 입력되었다면

```
INFO: IllegalArgumentException : org.hibernate.QueryException: No data type for node: org.hibernate.hql.internal.ast.tree.IdentNode
 \-[IDENT] IdentNode: 's' {originalText=s}
 [select s from entity.Student ss]
```

서버에서 위와 같은 Query Exception을 처리하고 다시 switch문을 시작한다.

[예외흐름3]

또한 만약, Student Table에 데이터가 없어 resultList의 size가 0이라면 NullDataException으로 예외처리를 하고 다시 Switch문을 시작한다.

```
INFO: NullDataException : NO STUDENT DATA FOUND
```

위의 그림은 서버의 NullDataException처리 화면이다.

[기본흐름4]

정상적으로 모든 학생 데이터를 가져온다면, getAllStudentsData 메서드의 setAllStudentsDataReponse 메서드를 통해 응답메세지를 만든다.

```
private AllStudentsDataResponse setAllStudentsDataResponse(List<Student> result) {
    ArrayList<AllStudentsDataResponse.Student> studentResults = new ArrayList<>();
    for (Student student : result) {
        ArrayList<String> courseNumbers = new ArrayList<>();
        List<StudentCourse> studentCourses = student.getStudentCourse();
        for (StudentCourse studentCourse : studentCourses) {
            courseNumbers.add(studentCourse.getCourse().getCourseNumber());
        }
        AllStudentsDataResponse.Student studentResult = AllStudentsDataResponse
                .Student
                .newBuilder()
                .setStudentName(student.getStudentName())
                .setStudentNumber(student.getStudentNumber())
                .setMajor(student.getMajor())
                .addAllCourseNumber(courseNumbers).build();
        studentResults.add(studentResult);
    }
    return AllStudentsDataResponse.newBuilder()
            .addAllStudents(studentResults).build();
}
```

응답메세지를 만들고, 클라이언트는 Reponse메시지를 받는다.

[기본흐름5]

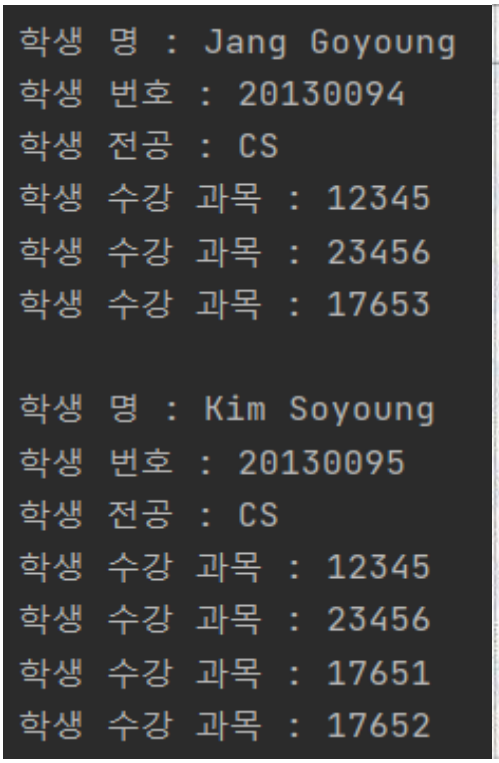
이후, 클라이언트 StudentView클래스의 allStudentsDataReponse메서드에서 showAllStudents메서드를 호출한다.

```
private void showAllStudents(List<AllStudentsDataResponse.Student> results) {
    for (AllStudentsDataResponse.Student result : results) {
        System.out.println();
        System.out.print(EStudentView.eStudentName.getContent()+result.getStudentName()+EStudentView.eSpacing.getContent() +
                EStudentView.eStudentNumber.getContent()+result.getStudentNumber()+EStudentView.eSpacing.getContent() +
                EStudentView.eStudentMajor.getContent()+result.getMajor()+EStudentView.eSpacing.getContent());
        for (int i = EStudentView.eZero.getNumber(); i < result.getCourseNumberCount(); i++) {
            System.out.println(EStudentView.eStudentApplicationForCourseNumber.getContent()
                    +result.getCourseNumberList().get(i));
        }
    }
}
```

위의 그림은 StudentView 클래스의 showAllStudents메서드이다. 결과값 배열의 크기만

큼 반복문으로 모든 학생 정보를 출력한다.

<'모든 학생 정보를 조회한다' 기능의 실행 결과>



student_id	major	studentName	studentNumber
76	CS	Kim Yunmi	20100123
77	ME	Kim Hokyung	20100125
78	CS	Jung Philsoo	20100323
79	EE	Ahn Jonghyuk	20080678
80	ME	Lee Mijung	20110298
81	EE	Kim Sungsuk	20120808
82	ME	Park Kitea	20080603
83	CS	Ko Kyungmin	20070452
84	CS	Kim Chulmin	20130091
85	EE	Park Kiyoung	20110876
86	CS	Kim Minsu	20100128
87	CS	Kim JungMi	20100131
88	CS	Jang Goyoung	20130094
89	CS	Kim Soyoung	20130095

student_course_id	course_id	student_id
66	1	88
67	2	88
68	5	88
69	1	89
70	2	89
71	3	89
72	4	89

위의 왼쪽 사진은 콘솔창의 일부화면이고 오른쪽 사진은 student테이블의 일부 데이터 화면, 맨 아래 사진은 student_course클래스의 일부 데이터화면이다. 이를 통해 student테이블의 모든 정보와 학생이 수강하고 있는 정보를 student_course테이블에서 가져옴을 알 수 있다.

2. 모든 강좌 정보를 조회한다.

[기본흐름1]

```
case two: courseView.allCoursesDataResponse();break;
```

사용자는 'two'를 클라이언트 콘솔창에 입력해 강좌 정보를 조회하는 메서드를 실행한다.

[대안흐름1]

만약, 사용자가 'two'가 아닌 오타를 입력했다면 아래와 같은 메시지와 함께 메뉴값을 다시 입력받을 수 있게 한다.

```
twoo
메뉴 입력이 잘못되었습니다. 메뉴를 다시 입력해 주세요.

**************************************************
********************* 학사관리 시스템*****************
**************************************************
*********************번호를 영어로 입력하세요****************
************(one) : 모든 학생 정보 불러오기**************
************(two) : 모든 강좌 정보 불러오기**************
```

이때 사용하는 검증 메서드는 아래와 같다.

```
private boolean validationCorrectMenuInput(String inputMenu) {
    if(inputMenu.equals(EMainClient.one.getContent()) || inputMenu.equals(EMainClient.two.getContent())
            || inputMenu.equals(EMainClient.three.getContent()) || inputMenu.equals(EMainClient.four.getContent())
            || inputMenu.equals(EMainClient.five.getContent()) || inputMenu.equals(EMainClient.six.getContent())
            || inputMenu.equals(EMainClient.seven.getContent()) || (inputMenu.equals(EMainClient.eight.getContent()))){
        return true; } else{ System.out.println(EMainClient.eEnterTheMenuAgain.getContent());
        return false; } }
```

[예외흐름1]

만약, 서버가 닫혀있는데 'two'를 입력했다면 '1. 학생 정보를 조회한다'의 io Exception 예외처리와 같이 처리한다.

[기본흐름2]

```
public void allCoursesDataResponse(){
    showAllCourses(courseServerBlockingStub.getAllCoursesData( request: null).getCoursesList());
}
```

성공적이라면 CourseView클래스의 allCoursesDataReponse메서드를 실행한다. 이후, 서버의 getAllCoursesData메서드를 실행한다.

```
@Override
public void getAllCoursesData(EmptyRequest empty, StreamObserver<AllCoursesDataResponse> responseObserver) {
    try {
        List<Course> result = courseRepository.findAll();
        AllCoursesDataResponse response = setAllCoursesDataResponse(result);
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    } catch (Exception e) {
        logger.info( msg: e.getClass().getSimpleName() + ECourseServiceImpl.eColon.getContent() + e.getMessage());
        Status status = Status.INTERNAL.withDescription(e.getMessage());
        responseObserver.onError(status.asRuntimeException());
        return;
    }
}
```

위의 그림은 CourseServiceImpl클래스의 getAllCoursesData메서드이다.

[기본흐름3]

위의 getAllCoursesData 메서드에서 CourseRepository클래스의 findAll메서드를 통해 모든 강좌 정보를 가져온다.

```
public List<Course> findAll() throws NullDataException {
    List<Course> resultList = em.createQuery(ECourseRepository.eFindAllCourseQuery.getContent(), Course.class).getResultList();
    if (resultList.size() == ECourseRepository.eZero.getNumber()) {
        throw new NullDataException(ECourseRepository.eNoCourseDataExceptionMessage.getContent()); }
    return resultList;
}
```

[예외흐름2]

만약, 쿼리가 잘못된 문법으로 입력되었다면 '1. 학생 정보를 조회한다'의 QueryException 예외처리와 같이 처리한다.

[예외흐름3]

또한 만약, Course Table에 데이터가 없어 resultList의 size가 0이라면 NullDataException으로 예외처리를 하고 다시 Switch문을 시작한다.

```
INFO: NullDataException : NO COURSE DATA FOUND
```

위의 그림은 서버의 NullDataException 예외처리 화면이다.

[기본흐름4]

정상적으로 모든 학생 데이터를 가져온다면, getAllCoursesData 메서드의 setAllCoursesDataReponse 메서드를 통해 응답메세지를 만든다.

```
private AllCoursesDataResponse setAllCoursesDataResponse(List<Course> result) {
    ArrayList<AllCoursesDataResponse.Course> courseResults = new ArrayList<>();
    for (Course course : result) {
        ArrayList<String> advancedCourseNumbers = new ArrayList<>();
        List<Course> advancedCourses = course.getAdvancedCourseList();
        for (Course advancedCourse : advancedCourses) {
            advancedCourseNumbers.add(advancedCourse.getCourseNumber());
        }
        AllCoursesDataResponse.Course courseResult = AllCoursesDataResponse
                .Course
                .newBuilder()
                .setCourseNumber(course.getCourseNumber())
                .setProfessorLastName(course.getProfessorLastName())
                .setCourseName(course.getCourseName())
                .addAllAdvancedCourseNumber(advancedCourseNumbers).build();
        courseResults.add(courseResult);
    }
    return AllCoursesDataResponse.newBuilder()
            .addAllCourses(courseResults).build();
}
```

위의 그림은 CourseServiceImpl클래스의 setAllCoursesDataResponse메서드이다. 응답메세지를 만들고 getAllCoursesData에서 최종적으로 클라이언트에 Response를 던져준다.

[기본흐름5]

이후, 클라이언트 CourseView클래스의 allCoursesDataReponse메서드에서 showAllCourses메서드를 호출한다. 아래 그림이 shwoAllCourses메서드이다.

```
private void showAllCourses(List<AllCoursesDataResponse.Course> results) {
    for (AllCoursesDataResponse.Course result : results) {
        System.out.println();
        System.out.print(ECourseView.eCourseNumber.getContent()+result.getCourseNumber()+ECourseView.eSpacing.getContent() +
                ECourseView.eCourseName.getContent()+result.getCourseName() + ECourseView.eSpacing.getContent() +
                ECourseView.eProfessorLastName.getContent()+result.getProfessorLastName()+ECourseView.eSpacing.getContent());
        for (int i = ECourseView.eZero.getNumber(); i < result.getAdvancedCourseNumberCount(); i++) {
            System.out.println(ECourseView.eAdvancedName.getContent() + result.getAdvancedCourseNumberList().get(i));
        }
    }
}
```

<'모든 강좌 정보를 조회한다' 기능의 실행결과>

클라이언트 콘솔창에서 아래 그림과 같은 정보를 확인할 수 있다.

```
강좌 번호 : 17653
강좌 명 : Managing_Software_Development
교수 성 : Kim

강좌 번호 : 17654
강좌 명 : Analysis_of_Software_Artifacts
교수 성 : Ahn
강좌 선이수 과목 : 17651
```

course_id	courseName	courseNumber	professorLastName
1	Java_Programming	12345	Park
2	C++_Programming	23456	Park
3	Models_of_Software_Systems	17651	Kim
4	Methods_of_Software_Development	17652	Ko
5	Managing_Software_Development	17653	Kim
6	Analysis_of_Software_Artifacts	17654	Ahn
7	Architectures_of_Software_Systems	17655	Lee

course_id	advanced_course_id
3	1
4	2
6	3
7	1
7	3

위의 첫번째 그림은 콘솔창의 일부 화면이고 두번째 그림은 course테이블의 데이터, 세번째 그림은 advanced_course테이블의 데이터 정보이다. 이를 통해 '모든 강좌 정보를 조회한다' 기능은 course테이블의 모든 강좌와 advanced_course테이블의 해당하는 강좌의 모든 선이수강좌를 보여줌을 알 수 있다.

3. 학생을 등록한다.

[기본흐름1]

```
case three: studentView.addStudentDataResponse(objReader);break;
```

사용자는 'three'를 클라이언트 콘솔창에 입력해 학생을 등록하는 메서드를 실행한다.

[대안흐름1]

만약, 사용자가 'three'가 아닌 오타를 입력했다면 아래와 같은 메시지와 함께 메뉴값을 다시 입력받을 수 있게 한다.

```
threee
메뉴 입력이 잘못되었습니다. 메뉴를 다시 입력해 주세요.

**************************************************
********************* 학사관리 시스템*****************
**************************************************
********************번호를 영어로 입력하세요***************
************(one) : 모든 학생 정보 불러오기**************
************(two) : 모든 강좌 정보 불러오기**************
```

이때 사용하는 검증 메서드는 아래와 같다.

```
private boolean validationCorrectMenuInput(String inputMenu) {
    if(inputMenu.equals(EMainClient.one.getContent()) || inputMenu.equals(EMainClient.two.getContent())
            || inputMenu.equals(EMainClient.three.getContent()) || inputMenu.equals(EMainClient.four.getContent())
            || inputMenu.equals(EMainClient.five.getContent()) || inputMenu.equals(EMainClient.six.getContent())
            || inputMenu.equals(EMainClient.seven.getContent()) || (inputMenu.equals(EMainClient.eight.getContent()))){
        return true; } else{ System.out.println(EMainClient.eEnterTheMenuAgain.getContent());
        return false; } }
```

[예외흐름1]

만약, 서버가 닫혀있는데 'three'를 입력했다면 '1. 학생 정보를 조회한다'의 io Exception 예외처리와 같이 처리한다.

[기본흐름2]

```
public void addStudentDataResponse(BufferedReader objReader) throws IOException {
    boolean isCompleted = this.studentServerBlockingStub.addStudentData(receiveAddStudentData(objReader)).getIsCompleted();
    if(isCompleted) System.out.println(EStudentView.eAddStudentSuccessMessage.getContent());
    else System.out.println(EStudentView.eAddStudentFailMessage.getContent());
}
```

위의 그림은 StudentView클래스의 addStudentDataResponse메서드이다. 먼저 receiveAddStudentData 메서드를 통해 사용자로부터 학생 정보를 입력받는다.

[기본흐름3]

아래 그림은 StudentView 클래스의 receiveAddStudentData메서드이다.

```
private AddStudentRequest receiveAddStudentData(BufferedReader objReader) throws IOException {
    System.out.println(EStudentView.eMenuStar.getContent());
    System.out.println(EStudentView.eMenuAddStudentGuide.getContent());
    System.out.println(EStudentView.eMenuStudentNameGuide.getContent()); String studentName = objReader.readLine().trim();
    System.out.println(EStudentView.eMenuStudentNumberGuide.getContent()); String studentNumber = objReader.readLine().trim();
    System.out.println(EStudentView.eMenuStudentMajorGuide.getContent()); String studentMajor = objReader.readLine().trim();
    validationAddStudentData(studentName, studentNumber, studentMajor);
    AddStudentRequest request = AddStudentRequest.newBuilder()
            .setStudentName(studentName)
            .setStudentNumber(studentNumber)
            .setMajor(studentMajor)
            .build();
    return request;
}
```

사용자로부터 studentName, studentNumber, major를 입력받고 validationAddStudentData 메서드를 통해 검증한다.

```
private void validationAddStudentData(String studentName, String studentNumber, String studentMajor) {
    validationStudentName(studentName);
    validationStudentNumber(studentNumber);
    validationStudentMajor(studentMajor);
}

private void validationStudentName(String studentName) {
    if(studentName.matches(EStudentView.eMatchContainNumber.getContent())){
        throw new IllegalArgumentException(EStudentView.eStudentNameMatchMessage.getContent()); }
}
private void validationStudentNumber(String studentNumber) {
    if(studentNumber.matches(EStudentView.eMatchContainNumber.getContent()) == false){
        throw new IllegalArgumentException(EStudentView.eStudentNumberMatchMessage.getContent()); }
}
private void validationStudentMajor(String studentMajor) {
    if(studentMajor.matches(EStudentView.eMatchContainNumber.getContent())){
        throw new IllegalArgumentException(EStudentView.eMajorMatchMessage.getContent()); }
}
```

위와 같은 로직을 이용해 검증한다.

[예외흐름2]

만약, 사용자가 잘못 입력했을 경우 아래 그림과 같은 예외를 던진다.

```
INFO: IllegalArgumentException : 학생이름에 숫자를 포함할 수 없습니다.
```

```
INFO: IllegalArgumentException : 학생번호에 숫자만 포함할 수 있습니다.
```

```
INFO: IllegalArgumentException : 전공에 숫자를 포함할 수 없습니다.
```

위와 같은 예외를 만나지 않는다면 서버의 addStudentData메서드를 호출한다.

[기본흐름4]

위의 그림은 StudentServiceImpl클래스의 addStudentData메서드이다. 먼저, 요청값을 validationStudent메서드를 통해 빈값이 들어왔는지 검증한다.

[예외흐름3]

만약, 빈 값이 들어왔다면 아래 그림과 같은 예외처리를 한다.

```
INFO: IllegalArgumentException : THE STUDENT INPUT IS A EMPTY VALUE.
```

[기본흐름5]

빈 값이 들어오지 않는다면, StudentRepository클래스의 findStudentByStudentNumber 메서드를 호출한다.

```
public Student findStudentByStudentNumber(String studentNumber, Boolean checkForStudentAdd)
        throws NullDataException, ExistingDataException {
    List<Student> findStudentList = em.createQuery(EStudentRepository.eFindStudentByStudentNumberQuery.getContent(),
            Student.class)
            .setParameter(EStudentRepository.eStudentNumber.getContent(), studentNumber)
            .getResultList();

    if (checkForStudentAdd == EStudentRepository.eFalse.getCheck() &&
            findStudentList.size() == EStudentRepository.eZero.getNumber()) {
        throw new NullDataException(EStudentRepository.eNoStudentDataByStudentNumberExceptionMessage.getContent()); }
    else if (checkForStudentAdd == EStudentRepository.eFalse.getCheck() &&
            findStudentList.size() > EStudentRepository.eOne.getNumber()) {
        throw new ExistingDataException(EStudentRepository.eManyStudentsDataByStudentNumberExceptionMessage.getContent()); }
    else if (checkForStudentAdd == EStudentRepository.eTrue.getCheck() &&
            findStudentList.size() >= EStudentRepository.eOne.getNumber()) {
        throw new ExistingDataException(EStudentRepository.eAlreadyStudentNumberExceptionMessage.getContent());}
    else if(checkForStudentAdd == EStudentRepository.eTrue.getCheck() &&
            findStudentList.size() == EStudentRepository.eZero.getNumber()) { return null;}
    return findStudentList.get(EStudentRepository.eZero.getNumber());
}
```

[예외흐름4]

쿼리 문법을 잘못입력했을 경우, QueryException이 발생하는데 이는 '1. 학생 정보를 조회한다'의 QueryException 예외처리와 같이 처리한다.

[기본흐름6]

문법에 이상이 없다면 데이터베이스에 있는 모든 학생 데이터를 가져온다.

[예외흐름5]

또한, 학생 정보를 추가하는 로직이 파라미터 checkForStudentAdd에 true가 들어온다. 따라서, if문 마지막 두개만 로직을 탈 가능성이 있다. CheckForStudentAdd가 true이고 학생리스트가 1이상이라는건 이미 존재하는 학생번호를 추가하려는 것을 의미한다. 만약, 이 로직을 탈 경우 아래와 같은 예외가 발생한다.

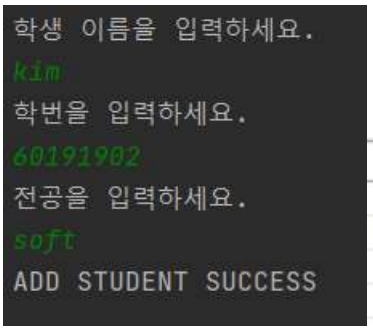
```
INFO: ExistingDataException : THIS STUDENT NUMBER ALREADY EXISTS.
```

[기본흐름7]

추가하려는 학생 정보가 없을 경우 null을 반환하며 예외가 발생하지 않는다. 이후, addStudentData 메서드에서 StudentRepository클래스의 createStudent메서드를 호출하고 데이터베이스에 저장한다. 서버는 저장이 완료되었으면 response에 true를 넣고 클라이언트에 반환하고, 마지막으로 클라이언트의 StudentView클래스의 addStudentDataReponse

메서드에서 성공 알림을 보여준다.

<'학생을 등록한다' 기능의 실행결과>



student_id	major	studentName	studentNumber
87	CS	Kim JungMi	20100131
88	CS	Jang Goyoung	20130094
89	CS	Kim Soyoung	20130095
90	soft	kim	60191902

위 그림의 왼쪽 그림은 콘솔창 입력화면, 출력화면이고 위 그림의 오른쪽 화면은 student테이블의 데이터 일부이다. 입력한 데이터가 student테이블에 들어가 있는 것을 확인할 수 있다.

4. 학생을 지운다.

[기본흐름1]

```
case four: studentView.deleteStudentDataResponse(objReader);break;
```

사용자는 'four'를 클라이언트 콘솔창에 입력해 학생을 지우는 메서드를 실행한다.

[대안흐름1]

만약, 사용자가 'four'가 아닌 오타를 입력했다면 아래와 같은 메시지와 함께 메뉴값을 다시 입력받을 수 있게 한다.

```
fourr
메뉴 입력이 잘못되었습니다. 메뉴를 다시 입력해 주세요.

**************************************************
******************** 학사관리 시스템******************
**************************************************
********************번호를 영어로 입력하세요****************
```

이때 사용하는 검증 메서드는 아래와 같다.

```
private boolean validationCorrectMenuInput(String inputMenu) {
    if(inputMenu.equals(EMainClient.one.getContent()) || inputMenu.equals(EMainClient.two.getContent())
            || inputMenu.equals(EMainClient.three.getContent()) || inputMenu.equals(EMainClient.four.getContent())
            || inputMenu.equals(EMainClient.five.getContent()) || inputMenu.equals(EMainClient.six.getContent())
            || inputMenu.equals(EMainClient.seven.getContent()) || (inputMenu.equals(EMainClient.eight.getContent()))){
        return true; } else{ System.out.println(EMainClient.eEnterTheMenuAgain.getContent());
        return false; } }
```

[예외흐름1]

만약, 서버가 닫혀있는데 'four'를 입력했다면 '1. 학생 정보를 조회한다'의 io Exception 예외처리와 같이 처리한다.

[기본흐름2]

```
public void deleteStudentDataResponse(BufferedReader objReader) throws IOException {
        boolean isCompleted = studentServerBlockingStub.deleteStudentData(receiveDeleteStudentData(objReader))
                .getIsCompleted();
        if(isCompleted) System.out.println(EStudentView.eDeleteStudentSuccessMessage.getContent());
        else System.out.println(EStudentView.eDeleteStudentFailMessage.getContent());
}
```

정상적인 흐름이라면, StudentView 클래스의 deleteStudentDataResponse 메서드를 호출한다.

[기본흐름3]

이후, receiveDeleteStudentData를 호출한다. 메서드는 아래 그림과 같다.

```
private DeleteStudentRequest receiveDeleteStudentData(BufferedReader objReader) throws IOException {
    System.out.println(EStudentView.eMenuStar.getContent());
    System.out.println(EStudentView.eMenuDeleteStudentGuide.getContent());
    System.out.println(EStudentView.eMenuStudentNumberGuide.getContent());
    String studentNumber = objReader.readLine().trim();
    validationStudentNumber(studentNumber);
    DeleteStudentRequest request = DeleteStudentRequest.newBuilder()
            .setStudentNumber(studentNumber)
            .build();
    return request;
}
```

이를 통해 사용자로부터 지우고자 하는 학번을 전달받는다.

[기본흐름4]

이후, validationStudentNumber 메서드를 통해 숫자만 입력되었는지 검증한다.

```
private void validationStudentNumber(String studentNumber) {
    if(studentNumber.matches(EStudentView.eMatchOnlyNumber.getContent()) == false){
        throw new IllegalArgumentException(EStudentView.eStudentNumberMatchMessage.getContent()); }
}
```

[예외흐름2]

숫자만 입력되어있지 않으면 아래 그림과 같은 예외가 발생한다.

```
INFO: IllegalArgumentException : 학생번호에 숫자만 포함할 수 있습니다.
```

[기본흐름5]

정상적으로 입력되었다면 서버의 deleteStudentData를 호출한다.

서버의 StudentServiceImpl 클래스의 deleteStudentData메서드는 아래 그림과 같다.

```java
@Override
public void deleteStudentData(DeleteStudentRequest request, StreamObserver<IsCompletedResponse> responseObserver) {
    try {
        validationStudentNumber(request);
        Student findStudent = studentRepository.findStudentByStudentNumber(request.getStudentNumber(),
                EStudentServiceImpl.eFalse.getCheck());
        List<StudentCourse> findStudentCourses = studentCourseRepository.findStudentCourseByStudent(findStudent);
        if (findStudentCourses.size() != EStudentServiceImpl.eZero.getNumber()) {
            studentCourseRepository.deleteStudentCourse(findStudentCourses); }
        boolean isCompletedDelete = studentRepository.deleteStudentByFindStudent(findStudent);

        IsCompletedResponse isCompleted = IsCompletedResponse.newBuilder()
                .setIsCompleted(isCompletedDelete).build();

        responseObserver.onNext(isCompleted);
        responseObserver.onCompleted();
    } catch (Exception e) {
        logger.info( msg: e.getClass().getSimpleName() + EStudentServiceImpl.eColon.getContent() + e.getMessage());
        Status status = Status.INTERNAL.withDescription(e.getMessage());
        responseObserver.onError(status.asRuntimeException());
        return;
    }
}
```

[기본흐름6]

먼저 요청값을 validationStudentNumber를 통해 검증한다.

```java
private void validationStudentNumber(DeleteStudentRequest request) {
    if (request.getStudentNumber().equals(EStudentServiceImpl.eEmpty.getContent())) {
        throw new IllegalArgumentException(EStudentServiceImpl.eEmptyRequestStudentNumberExceptionMessage.getContent()); } }
```

[예외흐름3]

빈 값이 들어온다면 IllegalArugmentException의 예외가 발생한다.

[기본흐름7]

이후, 정상적인 흐름이라면 StudentRepository의 findStudentByStudentNumber가 호출된다.

```
public Student findStudentByStudentNumber(String studentNumber, Boolean checkForStudentAdd)
        throws NullDataException, ExistingDataException {
    List<Student> findStudentList = em.createQuery(EStudentRepository.eFindStudentByStudentNumberQuery.getContent(),
            Student.class)
            .setParameter(EStudentRepository.eStudentNumber.getContent(), studentNumber)
            .getResultList();

    if (checkForStudentAdd == EStudentRepository.eFalse.getCheck() &&
            findStudentList.size() == EStudentRepository.eZero.getNumber()) {
        throw new NullDataException(EStudentRepository.eNoStudentDataByStudentNumberExceptionMessage.getContent()); }
    else if (checkForStudentAdd == EStudentRepository.eFalse.getCheck() &&
            findStudentList.size() > EStudentRepository.eOne.getNumber()) {
        throw new ExistingDataException(EStudentRepository.eManyStudentsDataByStudentNumberExceptionMessage.getContent()); }
    else if (checkForStudentAdd == EStudentRepository.eTrue.getCheck() &&
            findStudentList.size() >= EStudentRepository.eOne.getNumber()) {
        throw new ExistingDataException(EStudentRepository.eAlreadyStudentNumberExceptionMessage.getContent());}
    else if(checkForStudentAdd == EStudentRepository.eTrue.getCheck() &&
            findStudentList.size() == EStudentRepository.eZero.getNumber()) { return null;}
    return findStudentList.get(EStudentRepository.eZero.getNumber());
}
```

[예외흐름4]

쿼리 문법을 잘못입력했다면 마찬가지로 QueryException예외가 발생하고 이는 위에서 설명했던 내용과 같다.

[예외흐름5]

이후, checkForStudentAdd 파라미터가 false로 들어오기 때문에 4개의 if문중 위의 두개만 탈 가능성이 있다. 첫번째 if문을 탄다는건 입력받은 학생번호로 학생을 찾을 수 없다는 것을 의미한다. 아래 그림과 같다.

```
INFO: NullDataException : NO STUDENT DATA FOUND BY STUDENT_NUMBER
```

[예외흐름6]

두번째 if문은 이미 데이터베이스에 해당 학생번호가 두개이상 존재한다는 것이므로 아래 그림과 같은 예외를 던진다.

```
INFO: ExistingDataException : THERE ARE SEVERAL STUDENTS WITH THE SAME STUDENT_NUMBER
```

[기본흐름8]

이후, deleteStudentData메서드에서 찾은 findStudent로 student_course Table의 정보를 찾는다. StudentCourseRepository클래스의 findStudentCourseByStudent메서드를 이용한다.

```
public List<StudentCourse> findStudentCourseByStudent(Student student) {
    List<StudentCourse> studentList = em.createQuery(EStudentCourseRepository.eFindStudentCourseByStudentQuery.getContent(),
            StudentCourse.class)
            .setParameter(EStudentCourseRepository.eStudent.getContent(), student)
            .getResultList();
    return studentList;
}
```

[기본흐름9]

이후, null이 아니라면 deleteStudentData메서드에서 학생번호에 해당하는 student_course Table 정보를 모두 지우고,

```
public void deleteStudentCourse(List<StudentCourse> findStudentCourses) {
    EntityTransaction tx = em.getTransaction();
    tx.begin();
    for (StudentCourse findStudentCourse : findStudentCourses) { em.remove(findStudentCourse); }
    tx.commit();
}
```

[기본흐름10]

마지막으로 deleteStudentByStudentNumber메서드를 이용해 해당하는 학생정보를 지운다.

```
public boolean deleteStudentByFindStudent(Student findStudent) {
    EntityTransaction tx = em.getTransaction();
    tx.begin();
    em.remove(findStudent);
    tx.commit();
    return true;
}
```

위의 메서드에서 검증을 하지 않는 이유는, 이미 앞 단계에서 검증을 마쳤기 때문이다.

최종적으로 삭제를 완료했으면 response에 true를 넣고 클라이언트에 전달한다. 클라이언트는, StudentView 클래스의 deleteStudentDataResponse에서 true를 확인한 후 사용자에게 콘솔창에 메시지를 전달한다.

<'학생을 지운다' 기능의 실행결과>

<'학생을 지운다' 기능 실행전 테이블 상태>

student_id	major	studentName	studentNumber
86	CS	Kim Minsu	20100128
87	CS	Kim JungMi	20100131
88	CS	Jang Goyoung	20130094
89	CS	Kim Soyoung	20130095
90	soft	kim	60191902

student_course_id	course_id	student_id
67	2	88
68	5	88
69	1	89
70	2	89
71	3	89
72	4	89

<콘솔창 화면, '학생을 지운다' 기능 실행후 테이블 상태>

```
학번을 입력하세요.
20130095
DELETE STUDENT SUCCESS
```

student_id	major	studentName	studentNumber
84	CS	Kim Chulmin	20130091
85	EE	Park Kiyoung	20110876
86	CS	Kim Minsu	20100128
87	CS	Kim JungMi	20100131
88	CS	Jang Goyoung	20130094
90	soft	kim	60191902

student_course_id	course_id	student_id
63	4	87
64	5	87
65	7	87
66	1	88
67	2	88
68	5	88
NULL	NULL	NULL

맨 위의 그림은 기능을 실행하기 위해 콘솔창에서 학번을 입력받는 화면이고, 두번째 화

면은 student테이블, 세번째 사진은 student_course테이블 화면이다. 학번 20130095의 정보가 student테이블, student_course테이블에서 사라졌음을 확인할 수 있다.

5. 강좌를 등록한다.

[기본흐름1]

```
case five: courseView.addCourseDataResponse(objReader);break;
```

사용자는 'five'를 클라이언트 콘솔창에 입력해 강좌를 등록하는 메서드를 실행한다.

[대안흐름1]

만약, 사용자가 'five'가 아닌 오타를 입력했다면 아래와 같은 메시지와 함께 메뉴값을 다시 입력받을 수 있게 한다.

```
fivee
메뉴 입력이 잘못되었습니다. 메뉴를 다시 입력해 주세요.

**************************************************
********** 학사관리 시스템**********
**************************************************
**********번호를 영어로 입력하세요**********
```

이때 사용하는 검증 메서드는 아래와 같다.

```
private boolean validationCorrectMenuInput(String inputMenu) {
    if(inputMenu.equals(EMainClient.one.getContent()) || inputMenu.equals(EMainClient.two.getContent())
            || inputMenu.equals(EMainClient.three.getContent()) || inputMenu.equals(EMainClient.four.getContent())
            || inputMenu.equals(EMainClient.five.getContent()) || inputMenu.equals(EMainClient.six.getContent())
            || inputMenu.equals(EMainClient.seven.getContent()) || (inputMenu.equals(EMainClient.eight.getContent()))){
        return true; } else{ System.out.println(EMainClient.eEnterTheMenuAgain.getContent());
        return false; } }
```

[예외흐름1]

만약, 서버가 닫혀있는데 'five'를 입력했다면 '1. 학생 정보를 조회한다'의 io Exception 예외처리와 같이 처리한다

[기본흐름2]

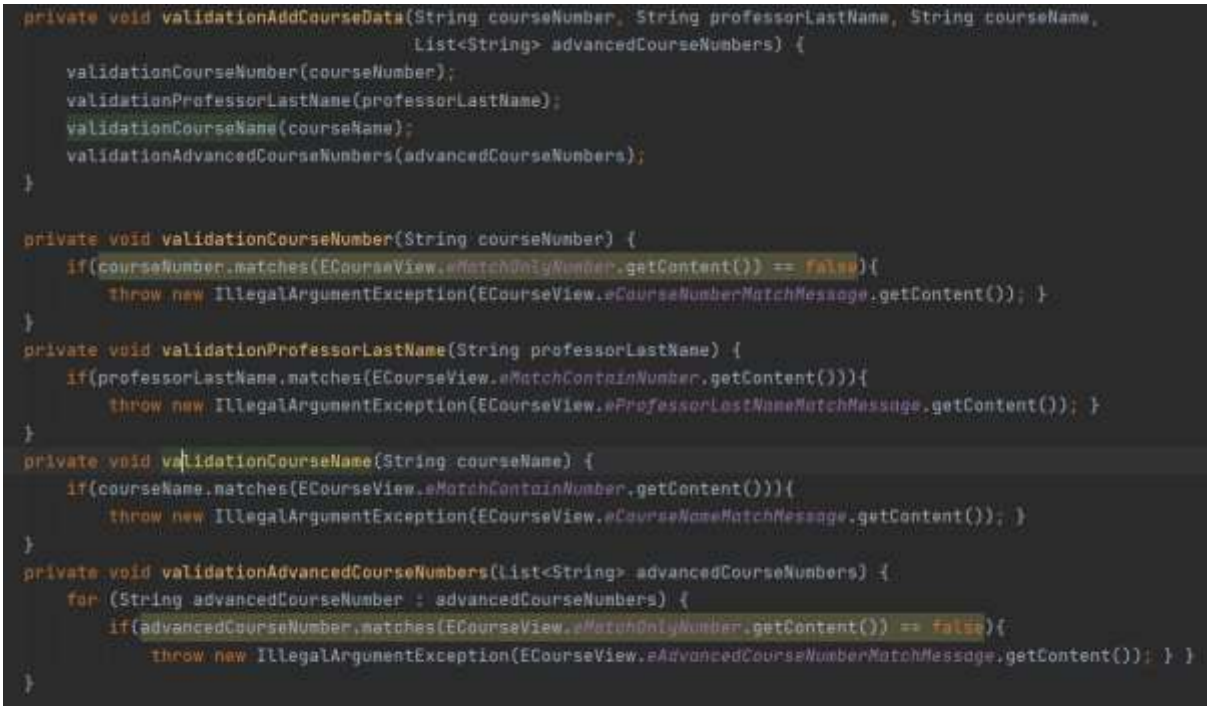
```
public void addCourseDataResponse(BufferedReader objReader) throws IOException {
    boolean isCompleted = this.courseServerBlockingStub.addCourseData(receiveAddCourseData(objReader)).getIsCompleted();
    if(isCompleted) System.out.println(ECourseView.eAddCourseSuccessMessage.getContent());
    else System.out.println(ECourseView.eAddCourseFailMessage.getContent());
}
```

CourseView 클래스의 addCourseDataReponse메서드이다. 먼저 receiveAddCourseData메서드를 통해 사용자로부터 추가하고자 하는 강좌 정보를 입력받는다.

```
private AddCourseRequest receiveAddCourseData(BufferedReader objReader) throws IOException {
    System.out.println(ECourseView.eMenuStar.getContent());
    System.out.println(ECourseView.eMenuAddCourseGuide.getContent());
    System.out.println(ECourseView.eMenuCourseNumberGuide.getContent()); String courseNumber = objReader.readLine().trim();
    System.out.println(ECourseView.eMenuProfessorLastNameGuide.getContent()); String professorLastName = objReader.readLine().trim();
    System.out.println(ECourseView.eMenuCourseNameGuide.getContent()); String courseName = objReader.readLine().trim();
    List<String> advancedCourseNumbers = new ArrayList<>();
    while (true) {
        System.out.println(ECourseView.eMenuAdvancedCourseNumberGuide.getContent());
        String advancedCourseNumber = objReader.readLine().trim();
        if (advancedCourseNumber.equals(ECourseView.eX.getContent())) { break; }
        advancedCourseNumbers.add(advancedCourseNumber);
    }
    validationAddCourseData(courseNumber, professorLastName, courseName, advancedCourseNumbers);
    AddCourseRequest request = AddCourseRequest.newBuilder()
            .setCourseNumber(courseNumber)
            .setProfessorLastName(professorLastName)
            .setCourseName(courseName)
            .addAllAdvancedCourseNumber(advancedCourseNumbers)
            .build();
    return request;
}
```

[기본흐름3]

사용자로부터 입력받은 데이터를 validationAddCourseData메서드를 통해 검증한다.



[예외흐름2]

만약, 입력받은 courseNumber가 숫자가 아니라면,

```
INFO: IllegalArgumentException : 강좌번호에 숫자만 포함되어야 합니다.
```

[예외흐름3]

위와 같은 예외를 던진다. 입력받은 professorLastName이 숫자를 포함한다면,

```
INFO: IllegalArgumentException : 교수 성에 숫자가 포함되서는 안됩니다.
```

위와 같은 예외를 던진다. 입력받은 courseName이 숫자를 포함한다면,

```
INFO: IllegalArgumentException : 강좌 이름에 숫자가 포함되서는 안됩니다.
```

위와 같은 예외를 던진다. 입력받은 advancedCourseNumbers가 숫자가 아니라면,

```
INFO: IllegalArgumentException : 선이수 강좌번호에 숫자만 포함되어야 합니다.
```

위와 같은 예외를 던진다.

[기본흐름4]

이상이 없으면 서버의 addCourseData를 호출한다.

```
@Override
public void addCourseData(AddCourseRequest request, StreamObserver<IsCompletedResponse> responseObserver) {
    try {
        validationCourse(request);
        List<Course> coursesResult;
        if(request.getAdvancedCourseNumberList().size() != ECourseServiceImpl.eZero.getNumber()) {
            coursesResult = courseRepository.findCoursesByCourseNumber(request.getAdvancedCourseNumberList());
            courseRepository.addCourseWithAdvancedCourse(request, coursesResult);
        } else{ courseRepository.createCourse(request); }

        IsCompletedResponse isCompleted = IsCompletedResponse.newBuilder()
                .setIsCompleted(ECourseServiceImpl.eTrue.getCheck()).build();

        responseObserver.onNext(isCompleted);
        responseObserver.onCompleted();
    }
     catch(Exception e){
            logger.info( msg: e.getClass().getSimpleName() + ECourseServiceImpl.eColon.getContent() + e.getMessage());
            Status status = Status.INTERNAL.withDescription(e.getMessage());
            responseObserver.onError(status.asRuntimeException());
            return; } }
```

CourseServiceImpl 클래스의 addCourseData메서드이다. 먼저, 빈 값이 넘어왔는지 validationCourse메서드를 통해 검증한다.

```
private void validationCourse(AddCourseRequest request) {
    if (request.getCourseNumber().equals(ECourseServiceImpl.eEmpty.getContent()) ||
            request.getProfessorLastName().equals(ECourseServiceImpl.eEmpty.getContent()) ||
            request.getCourseName().equals(ECourseServiceImpl.eEmpty.getContent())) {
        throw new IllegalArgumentException(ECourseServiceImpl.eEmptyRequestCourseExceptionMessage.getContent());
    }
}
```

[예외흐름4]

만약, 빈 값이 넘어온다면 아래와 같은 예외를 던진다.

```
INFO: IllegalArgumentException : THE COURSE INPUT IS A EMPTY VALUE.
```

[기본흐름5]

이후, addCourseData메서드에서 선강좌 정보를 입력받았는지 아닌지에 대해 나뉜다. 입력받았다면, CourseRepository클래스의 findCoursesByCourseNumber메서드를 호출한다.

```
public List<Course> findCoursesByCourseNumber(ProtocolStringList courseNumberList){
    List<Course> coursesResult = new ArrayList<>();
    for (String courseNumber : courseNumberList) {
        Course singleResult = em.createQuery(ECourseRepository.eFindCourseByCourseNumberQuery.getContent(), Course.class)
                .setParameter(ECourseRepository.eCourseNumber.getContent(), courseNumber)
                .getSingleResult();
        coursesResult.add(singleResult); }
    return coursesResult;
}
```

[예외흐름5]

입력받은 선강좌번호가 데이터베이스에 없는 강좌 정보라면 NoResultException예외를 던진다.

```
INFO: NoResultException : No entity found for query
```

[기본흐름6]

정상적인 정보라면 강좌정보를 배열로 반환한다. 이후 CourseRepository클래스의 addCourseWithAdvancedCourse메서드를 호출하고, advanced_course Table에 데이터를 추가, course Table에 데이터를 추가한다. 만약, 선강좌정보가 없다면 course Table에만 데이터를 추가한다.

```
public Course addCourseWithAdvancedCourse(AddCourseRequest request, List<Course> coursesResult) {
    EntityTransaction tx = em.getTransaction();
    tx.begin();
    Course createdCourse = Course.createCourse(request.getCourseNumber(), request.getProfessorLastName(),
            request.getCourseName(), coursesResult);
    em.persist(createdCourse);
    tx.commit();
    return createdCourse;
}
```

```
public Course createCourse(AddCourseRequest request) {
    Course course = Course.createCourse(request.getCourseNumber(), request.getProfessorLastName(), request.getCourseName());
    em.persist(course);
    return course;
}
```

이후, 완료했으면 response에 true를 넣고 클라이언트에 전달한다. 마지막으로 클라이언트는 addCourseDataResponse메서드에서 완료여부를 콘솔창으로 알려준다.

<'강좌를 등록한다' 기능의 실행결과>

<'강좌를 등록한다' 기능 실행전 테이블 상태>

course_id	courseName	courseNumber	professorLastName
1	Java_Programming	12345	Park
2	C++_Programming	23456	Park
3	Models_of_Software_Systems	17651	Kim
4	Methods_of_Software_Development	17652	Ko
5	Managing_Software_Development	17653	Kim
6	Analysis_of_Software_Artifacts	17654	Ahn
7	Architectures_of_Software_Systems	17655	Lee
NULL	NULL	NULL	NULL

course_id	advanced_course_id
3	1
4	2
6	3
7	1
7	3

위의 첫번째 화면은 course테이블이고 두번째 화면은 advanced_course테이블의 화면이다.

<콘솔창 입력, '강좌를 등록한다' 기능 실행후 테이블 상태>

```
five
**********************************************************
**********************강좌 정보 입력하기*****************
강좌 번호를 입력하세요.
11
교수 성을 입력하세요.
kim
강좌 이름을 입력하세요.
soft
선이수 강좌 번호를 입력하세요. 입력을 마치셨으면 x를 입력하세요.
12345
선이수 강좌 번호를 입력하세요. 입력을 마치셨으면 x를 입력하세요.
23456
선이수 강좌 번호를 입력하세요. 입력을 마치셨으면 x를 입력하세요.
x
ADD COURSE SUCCESS
```

course_id	courseName	courseNumber	professorLastName
1	Java_Programming	12345	Park
2	C++_Programming	23456	Park
3	Models_of_Software_Systems	17651	Kim
4	Methods_of_Software_Development	17652	Ko
5	Managing_Software_Development	17653	Kim
6	Analysis_of_Software_Artifacts	17654	Ahn
7	Architectures_of_Software_Systems	17655	Lee
91	soft	11	kim

course_id	advanced_course_id
3	1
4	2
6	3
7	1
7	3
91	1
91	2

3개의 그림중 첫번째 그림은 콘솔창 입력화면이고, 두번째 그림은 course테이블, 세번째 그림은 advanced_course테이블 화면이다. 사용자가 입력한 강좌정보가 course테이블에, 선이수강좌정보가 advanced_course테이블에 들어가 있는 것을 확인할 수 있다.

6. 강좌를 삭제한다.

[기본흐름1]

```
case six: courseView.deleteCourseDataResponse(objReader);break;
```

사용자는 'six'를 클라이언트 콘솔창에 입력해 강좌를 등록하는 메서드를 실행한다.

[대안흐름1]

만약, 사용자가 'six'가 아닌 오타를 입력했다면 아래와 같은 메시지와 함께 메뉴값을 다시 입력받을 수 있게 한다.

```
sixx
메뉴 입력이 잘못되었습니다. 메뉴를 다시 입력해 주세요.

**************************************************
******************** 학사관리 시스템*****************
**************************************************
********************번호를 영어로 입력하세요****************
```

이때 사용하는 검증 메서드는 아래와 같다.

```
private boolean validationCorrectMenuInput(String inputMenu) {
    if(inputMenu.equals(EMainClient.one.getContent()) || inputMenu.equals(EMainClient.two.getContent())
            || inputMenu.equals(EMainClient.three.getContent()) || inputMenu.equals(EMainClient.four.getContent())
            || inputMenu.equals(EMainClient.five.getContent()) || inputMenu.equals(EMainClient.six.getContent())
            || inputMenu.equals(EMainClient.seven.getContent()) || (inputMenu.equals(EMainClient.eight.getContent()))){
        return true; } else{ System.out.println(EMainClient.eEnterTheMenuAgain.getContent());
        return false; } }
```

[예외흐름1]

만약, 서버가 닫혀있는데 'six'를 입력했다면 '1. 학생 정보를 조회한다'의 io Exception 예외처리와 같이 처리한다.

[기본흐름2]

```
public void deleteCourseDataResponse(BufferedReader objReader) throws IOException {
    boolean isCompleted = this.courseServerBlockingStub.deleteCourseData(receiveDeleteCourseData(objReader)).getIsCompleted();
    if(isCompleted) System.out.println(ECourseView.eDeleteCourseSuccessMessage.getContent());
    else System.out.println(ECourseView.eDeleteCourseFailMessage.getContent());
}
```

위의 그림은 CourseView클래스의 deleteCourseDataResponse메서드이다. 먼저 receiveDeleteCourseData메서드를 통해 사용자로부터 정보를 입력받는다.

```
private DeleteCourseRequest receiveDeleteCourseData(BufferedReader objReader) throws IOException {
    System.out.println(ECourseView.eMenuStar.getContent());
    System.out.println(ECourseView.eMenuDeleteCourseGuide.getContent());
    System.out.println(ECourseView.eMenuCourseNumberGuide.getContent());
    String courseNumber = objReader.readLine().trim();
    validationCourseNumber(courseNumber);
    DeleteCourseRequest request = DeleteCourseRequest.newBuilder()
            .setCourseNumber(courseNumber)
            .build();
    return request;
}
```

[기본흐름3]

이후, validationCourseNumber메서드를 통해 강좌 번호를 검증한다.

```
private void validationCourseNumber(String courseNumber) {
    if(courseNumber.matches(ECourseView.eMatchOnlyNumber.getContent()) == false){
        throw new IllegalArgumentException(ECourseView.eCourseNumberMatchMessage.getContent()); }
}
```

[예외흐름2]

강좌 번호가 숫자만 포함하고 있지 않다면 예외를 던진다.

```
INFO: IllegalArgumentException : 강좌번호에 숫자만 포함되어야 합니다.
```

[기본흐름4]

문제없다면, deleteCourseDataResponse메서드에서 서버의 deleteCourseData를 호출한다.

```
@Override
public void deleteCourseData(DeleteCourseRequest request, StreamObserver<IsCompletedResponse> responseObserver) {
    try {
        validationCourseNumber(request);
        boolean isCompletedDelete = courseRepository.deleteCourseByCourseNumber(request.getCourseNumber());
        IsCompletedResponse isCompleted = IsCompletedResponse.newBuilder()
                .setIsCompleted(isCompletedDelete).build();

        responseObserver.onNext(isCompleted);
        responseObserver.onCompleted();
    } catch (Exception e) {
        logger.info( msg: e.getClass().getSimpleName() + ECourseServiceImpl.eColon.getContent() + e.getMessage());
        Status status = Status.INTERNAL.withDescription(e.getMessage());
        responseObserver.onError(status.asRuntimeException());
        return;
    }
}
```

위의 그림은 CourseServiceImpl클래스의 deleteCourseData메서드이다.

[기본흐름5]

먼저 요청을 validationCourseNumber메서드를 통해 검증한다.

```
private void validationCourseNumber(DeleteCourseRequest request) {
    if (request.getCourseNumber().equals(ECourseServiceImpl.eEmpty.getContent())) {
        throw new IllegalArgumentException(ECourseServiceImpl.eEmptyRequestCourseNumberExceptionMessage.getContent());
    }
}
```

[예외흐름3]

빈 값이 들어왔다면 아래와 같은 예외를 던진다.

```
INFO: IllegalArgumentException : THE COURSE INPUT IS A EMPTY VALUE.
```

[기본흐름6]

```
public boolean deleteCourseByCourseNumber(String courseNumber) throws NullDataException, ExistingDataException {

    Course findCourse = findCourseByCourseNumber(courseNumber);
    List<StudentCourse> studentCourses = this.studentCourseRepository.findStudentCourseByCourse(findCourse);
    deleteStudentCourseByCourse(studentCourses);
    deleteAdvancedCourseByCourse(findCourse);

    EntityTransaction tx = em.getTransaction();
    tx.begin();
    em.remove(findCourse);
    tx.commit();
    return true;
}
```

문제없다면, deleteCourseData메서드에서 deleteCourseByCourseNumber메서드를 호출한다.

[기본흐름7]

이후 findCourseByCourseNumber메서드를 호출한다.

```
public Course findCourseByCourseNumber(String courseNumber) throws NullDataException, ExistingDataException {
    List<Course> findCourseList = em.createQuery(ECourseRepository.eFindCourseListByCourseNumberQuery.getContent(),
            Course.class)
            .setParameter(ECourseRepository.eCourseNumber.getContent(), courseNumber)
            .getResultList();
    if (findCourseList.size() == ECourseRepository.eZero.getNumber()) {
        throw new NullDataException(ECourseRepository.eNoCourseDataByCourseNumberExceptionMessage.getContent()); }
    else if (findCourseList.size() > ECourseRepository.eOne.getNumber()) {
        throw new ExistingDataException(ECourseRepository.eManyCoursesByCourseNumberExceptionMessage.getContent());}
    return findCourseList.get(ECourseRepository.eZero.getNumber());
}
```

[예외흐름4]

입력받은 강좌정보로 강좌정보를 찾을 수 없으면 NullDataException예외를 던진다.

```
INFO: NullDataException : NO COURSE DATA FOUND BY COURSE_NUMBER
```

[예외흐름5]

입력받은 강좌정보로 강좌정보를 찾을 때 찾아진 값이 2개 이상이라면 ExistingDataException예외를 던진다.

```
INFO: ExistingDataException : THERE ARE SEVERAL COURSES WITH THE SAME COURSE_NUMBER
```

[기본흐름8]

```
public List<StudentCourse> findStudentCourseByCourse(Course findCourse) {
    List<StudentCourse> studentCourses = em.createQuery(EStudentCourseRepository.eFindStudentCourseByCourseQuery.getContent(),
            StudentCourse.class)
            .setParameter(EStudentCourseRepository.eCourse.getContent(), findCourse)
            .getResultList();
    return studentCourses;
}
```

문제 없으면 deleteCourseByCourseNumber메서드에서 findStudentCourseByCourse메서드를 호출하고 advanced_course Table에 있는 정보를 강좌 번호와 일치하면 지우고, student_course Table에 있는 정보도 강좌 번호와 일치하면 지운다. 이때, 사용자가 수강신청한 강좌 정보, 지우고자 하는 강좌 정보가 선이수강좌에 포함될 수 있는데 이 경우 경고 메시지를 사용자에게 알려준다.

```
INFO: WARNING! DELETE COURSE'S ADVANCED_COURSE DATA
```

```
INFO: WARNING! DELETE STUDENT'S COURSE DATA
```

위의 그림은 경고 메시지를 보여준다.

문제가 없다면, 강좌를 지우고 response에 true를 넣고 클라이언트에 전달한다. 최종적으로 클라이언트는 삭제가 완료됐으면 메시지를 콘솔창에 출력한다.

<'강좌를 삭제한다' 기능의 실행결과>

<'강좌를 삭제한다' 기능 실행전 테이블 상태>

course_id	courseName	courseNumber	professorLastName
1	Java_Programming	12345	Park
2	C++_Programming	23456	Park
3	Models_of_Software_Systems	17651	Kim
4	Methods_of_Software_Development	17652	Ko
5	Managing_Software_Development	17653	Kim
6	Analysis_of_Software_Artifacts	17654	Ahn
7	Architectures_of_Software_Systems	17655	Lee
NULL	NULL	NULL	NULL

course_id	advanced_course_id
3	1
4	2
6	3
7	1
7	3

student_course_id	course_id	student_id
20	1	76
21	3	76
22	4	76
23	5	76
24	1	77
25	2	77
26	3	77
27	6	77
28	1	78
29	2	78
30	3	78

위의 첫번째 화면은 course테이블이고 두번째 화면은 advanced_course테이블의 화면, 세번째 화면은 student_course테이블 화면이다.

<콘솔창 입력, '강좌를 삭제한다' 기능 실행후 테이블 상태>

```
강좌 번호를 입력하세요.
17654
DELETE COURSE SUCCESS
```

course_id	courseName	courseNumber	professorLastName
1	Java_Programming	12345	Park
2	C++_Programming	23456	Park
3	Models_of_Software_Systems	17651	Kim
4	Methods_of_Software_Development	17652	Ko
5	Managing_Software_Development	17653	Kim
7	Architectures_of_Software_Systems	17655	Lee
NULL	NULL	NULL	NULL

course_id	advanced_course_id
3	1
4	2
7	1
7	3

student_course_id	course_id	student_id
24	1	77
25	2	77
26	3	77
28	1	78
29	2	78
30	3	78
31	7	78
32	1	79
33	3	79
34	4	79
36	1	80

맨위의 화면은 콘솔창에 강좌번호17654번을 입력받는 화면, 두번째는 course테이블, 세번째는 advanced_course테이블, 네번째는 student_course테이블 화면이다. 메서드가 실행되면, course테이블의 정보, advanced_course테이블의 정보, student_course테이블의 정보가 삭제되는 것을 확인할 수 있다.

7. 수강신청을 한다.

[기본흐름1]

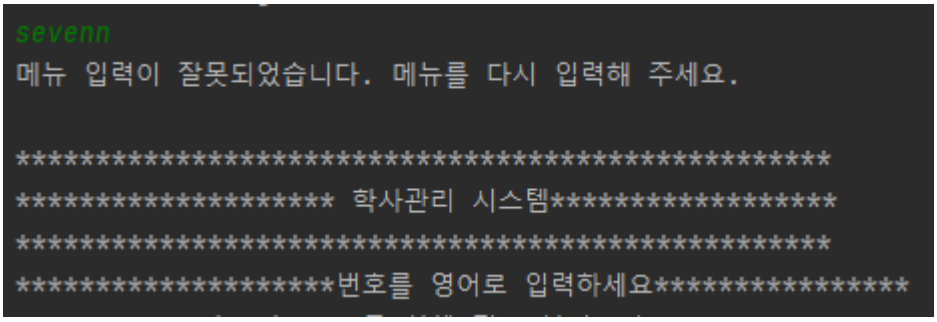
```
case seven: studentCourseView.applicationForCourse(objReader);break;
```

사용자는 'seven'을 클라이언트 콘솔창에 입력해 강좌를 등록하는 메서드를 실행한다.

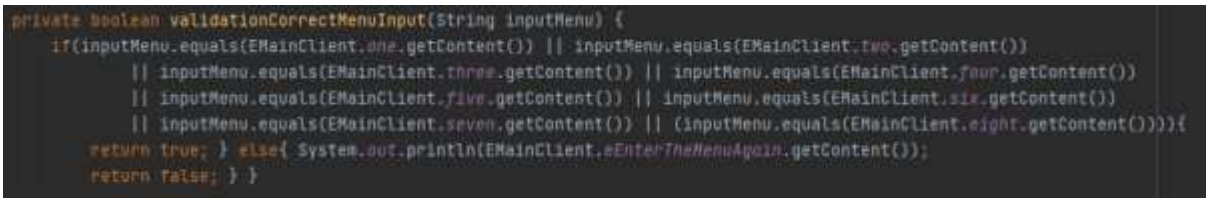
[대안흐름1]

만약, 사용자가 'seven'가 아닌 오타를 입력했다면 아래와 같은 메시지와 함께 메뉴값

을 다시 입력받을 수 있게 한다.



이때 사용하는 검증 메서드는 아래와 같다.



[예외흐름1]

만약, 서버가 닫혀있는데 'seven'을 입력했다면 '1. 학생 정보를 조회한다'의 io Exception 예외처리와 같이 처리한다.

[기본흐름2]

위의 그림은 StudentCourseView클래스의 applicationForCourse메서드이다. 먼저 receiveApplicationForCourseRequest메서드를 통해 사용자로부터 수강신청 정보를 입력받는다.

[기본흐름3]

정보를 입력받고 validationApplicationForCourseData메서드를 통해 검증한다.

```
private void validationApplicationForCourseData(String studentNumber, String courseNumber) {
    validationStudentNumber(studentNumber);
    validationCourseNumber(courseNumber); }
private void validationStudentNumber(String studentNumber) {
    if(studentNumber.matches(EStudentCourseView.eMatchOnlyNumber.getContent()) == false){
        throw new IllegalArgumentException(EStudentCourseView.eStudentNumberMatchMessage.getContent()); } }
private void validationCourseNumber(String courseNumber) {
    if(courseNumber.matches(EStudentCourseView.eMatchOnlyNumber.getContent()) == false){
        throw new IllegalArgumentException(EStudentCourseView.eCourseNumberMatchMessage.getContent()); } }
```

[예외흐름2]

StudentNumber가 숫자만 포함되지 않았다면 예외처리, CourseNumber가 숫자만 포함되지 않았다면 예외처리를 한다.

```
INFO: IllegalArgumentException : 학생번호에 숫자만 포함할 수 있습니다.
```

```
INFO: IllegalArgumentException : 강좌번호에 숫자만 포함되어야 합니다.
```

[기본흐름4]

문제가 없으면 서버의 applicationForCourse메서드를 호출한다.

```
@Override
public void applicationForCourse(ApplicationForCourseRequest request, StreamObserver<IsCompletedResponse> responseObserver) {
    try {
        validationStudentId(request);
        validationCourseId(request);
        Student findStudent = studentRepository.findStudentByStudentNumber(request.getStudentNumber(),
                EStudentCourseServiceImpl.eFalse.getCheck());
        Course findCourse = courseRepository.findCourseByCourseNumber(request.getCourseNumber());
        List<StudentCourse> studentCourses = studentCourseRepository.findStudentCourseByStudent(findStudent);
        if(studentCourses.size() != EStudentCourseServiceImpl.eZero.getNumber()){
            validationExistingCourse(findCourse, studentCourses); }
        validationAdvancedCourse(findCourse, studentCourses);

        StudentCourse studentCourse = studentCourseRepository.createStudentCourse(findCourse);
        studentRepository.addStudentCourse(findStudent, studentCourse);

        IsCompletedResponse isCompleted = IsCompletedResponse.newBuilder()
                .setIsCompleted(EStudentCourseServiceImpl.eTrue.getCheck()).build();

        responseObserver.onNext(isCompleted);
        responseObserver.onCompleted();
    } catch (Exception e) {
        logger.info( msg: e.getClass().getSimpleName() + EStudentCourseServiceImpl.eColon.getContent() + e.getMessage());
        Status status = Status.INTERNAL.withDescription(e.getMessage());
        responseObserver.onError(status.asRuntimeException());
        return ;
    }
}
```

[기본흐름5]

이후 사용자가 입력한 정보가 빈 스트링인지 validationStudentId, validationCourseId메서드를 통해 검증한다.

```
private void validationCourseId(ApplicationForCourseRequest request) {
    if (request.getCourseNumber().equals(EStudentCourseServiceImpl.eEmpty.getContent())) {
        throw new IllegalArgumentException(EStudentCourseServiceImpl.eEmptyRequestCourseIdExceptionMessage.getContent());
    }
}
private void validationStudentId(ApplicationForCourseRequest request) {
    if (request.getStudentNumber().equals(EStudentCourseServiceImpl.eEmpty.getContent())) {
        throw new IllegalArgumentException(EStudentCourseServiceImpl.eEmptyRequestStudentIdExceptionMessage.getContent());
    }
}
```

[기본흐름6]

문제가 없으면 StudentRepository클래스의 findStudentByStudentNumber메서드를 호출한다.

```
public Student findStudentByStudentNumber(String studentNumber, Boolean checkForStudentAdd)
        throws NullDataException, ExistingDataException {
    List<Student> findStudentList = em.createQuery(EStudentRepository.eFindStudentByStudentNumberQuery.getContent(),
            Student.class)
            .setParameter(EStudentRepository.eStudentNumber.getContent(), studentNumber)
            .getResultList();

    if (checkForStudentAdd == EStudentRepository.eFalse.getCheck() &&
            findStudentList.size() == EStudentRepository.eZero.getNumber()) {
        throw new NullDataException(EStudentRepository.eNoStudentDataByStudentNumberExceptionMessage.getContent()); }
    else if (checkForStudentAdd == EStudentRepository.eFalse.getCheck() &&
            findStudentList.size() > EStudentRepository.eOne.getNumber()) {
        throw new ExistingDataException(EStudentRepository.eManyStudentsDataByStudentNumberExceptionMessage.getContent()); }
    else if (checkForStudentAdd == EStudentRepository.eTrue.getCheck() &&
            findStudentList.size() >= EStudentRepository.eOne.getNumber()) {
        throw new ExistingDataException(EStudentRepository.eAlreadyStudentNumberExceptionMessage.getContent());}
    else if(checkForStudentAdd == EStudentRepository.eTrue.getCheck() &&
            findStudentList.size() == EStudentRepository.eZero.getNumber()) { return null;}
    return findStudentList.get(EStudentRepository.eZero.getNumber());
}
```

[예외흐름3]

CheckForStudentAdd가 False이기 때문에 if문 위의 두개만 검증한다. 첫번째 if문은 입력한 학생 번호로 학생 정보를 찾을 수 없는 경우의 NullDataException예외이고, 두번째 if문은 입력한 학생 번호로 여러 개의 학생 정보가 데이터베이스에서 찾아진 경우 ExistingDataException예외이다.

```
INFO: NullDataException : NO STUDENT DATA FOUND BY STUDENT_NUMBER
```

```
INFO: ExistingDataException : THERE ARE SEVERAL STUDENTS WITH THE SAME STUDENT_NUMBER
```

[기본흐름7]

문제가 없으면 학생 정보를 반환한다. 이후 applicationForCourse메서드에서 찾은 입력한 강좌 번호로 강좌 정보를 찾고, 찾은 학생 정보로 student_course Table에 있는 일치하는 정보를 찾는다. 각각 CourseRepository클래스의 findCourseByCourseNumber메서드, StudentCourseRepository클래스의 findStudentCourseByStudent메서드이다.

```
public Course findCourseByCourseNumber(String courseNumber) throws NullDataException, ExistingDataException {
    List<Course> findCourseList = em.createQuery(ECourseRepository.eFindCourseListByCourseNumberQuery.getContent(),
            Course.class)
            .setParameter(ECourseRepository.eCourseNumber.getContent(), courseNumber)
            .getResultList();
    if (findCourseList.size() == ECourseRepository.eZero.getNumber()) {
        throw new NullDataException(ECourseRepository.eNoCourseDataByCourseNumberExceptionMessage.getContent()); }
    else if (findCourseList.size() > ECourseRepository.eOne.getNumber()) {
        throw new ExistingDataException(ECourseRepository.eManyCoursesByCourseNumberExceptionMessage.getContent());}
    return findCourseList.get(ECourseRepository.eZero.getNumber());
}
```

```
public List<StudentCourse> findStudentCourseByStudent(Student student) {
    List<StudentCourse> studentList = em.createQuery(EStudentCourseRepository.eFindStudentCourseByStudentQuery.getContent(),
            StudentCourse.class)
            .setParameter(EStudentCourseRepository.eStudent.getContent(), student)
            .getResultList();
    return studentList;
}
```

[예외흐름4]

입력한 강좌 번호로 강좌 정보를 찾을 때 데이터베이스에 입력한 강좌 번호가 없으면 NullDataException예외, 두 개 이상의 강좌를 찾으면 ExistingDataException예외처리를 한다.

```
INFO: NullDataException : NO COURSE DATA FOUND BY COURSE_NUMBER
```

```
INFO: ExistingDataException : THERE ARE SEVERAL COURSES WITH THE SAME COURSE_NUMBER
```

[기본흐름8]

이후, 학생이 이미 강좌를 듣고 있는지 여부를 체크하기 위해 validationExistingCourse 메서드를 통해 검증, 학생이 수강신청 하려는 강좌를 듣기 위해 선이수강좌를 들었는

지 여부를 검증하기 위해 validationAdvancedCourse메서드를 사용한다.

```
private void validationExistingCourse(Course findCourse, List<StudentCourse> studentCourses) throws ExistingDataException {
    for (StudentCourse studentCourse : studentCourses) {
        if (studentCourse.getCourse().getId() == findCourse.getId()) {
            throw new ExistingDataException(EStudentCourseServiceImpl.eAlreadyTakingCourseExceptionMessage.getContent());
        }
    }
}
private void validationAdvancedCourse(Course findCourse, List<StudentCourse> studentCourses) throws AdvancedCourseException {
    boolean advancedCourseCheck = EStudentCourseServiceImpl.eFalse.getCheck();
    if (findCourse.getAdvancedCourseList().size() != 0) {
        for (Course course : findCourse.getAdvancedCourseList()) {
            for (StudentCourse studentCourse : studentCourses) { if (studentCourse.getCourse().getId() == course.getId()) {
                advancedCourseCheck = EStudentCourseServiceImpl.eTrue.getCheck(); } }
        } if (advancedCourseCheck == EStudentCourseServiceImpl.eFalse.getCheck()) {
            throw new AdvancedCourseException(EStudentCourseServiceImpl.eTakeAdvancedCourseExceptionMessage.getContent()); }
    }
}
```

[예외흐름5]

이미 학생이 듣고있는 강좌를 수강신청 했을 경우, 아래와 같은 예외를 던진다.

```
INFO: ExistingDataException : THIS IS A COURSE YOU ARE ALREADY TAKING
```

[예외흐름6]

학생이 선이수강좌를 듣지 않았는데 선이수강좌가 필요한 강좌를 수강신청 했을 경우 아래와 같은 예외를 던진다.

```
INFO: AdvancedCourseException : YOU DIDN'T TAKE THE ADVANCED COURSE
```

[기본흐름9]

문제가 없으면, student_course Table에 데이터를 추가해서 학생이 강좌를 듣고있는지를 나타낸다. 이후, response에 true를 넣고 클라이언트에 반환한다. 마지막으로 클라이언트는 수강신청이 완료되었으면 메시지를 콘솔창에 출력한다.

<'수강신청을 한다' 기능의 실행결과>

<'수강신청을 한다' 기능 실행전 테이블 상태>

student_id	major	studentName	studentNumber
85	EE	Park Kiyoung	20110876
86	CS	Kim Minsu	20100128
87	CS	Kim JungMi	20100131
88	CS	Jang Goyoung	20130094
89	CS	Kim Soyoung	20130095
NULL	NULL	NULL	NULL

student_course_id	course_id	student_id
63	4	87
64	5	87
65	7	87
66	1	88
67	2	88
68	5	88
69	1	89
70	2	89
71	3	89
72	4	89
NULL	NULL	NULL

위의 첫번째 화면은 student테이블이고 두번째 화면은 student_course테이블의 화면이다.

<콘솔창 입력, '수강신청을 한다' 기능 실행후 테이블 상태>

```
학번을 입력하세요.
20130094
강좌번호를 입력하세요.
17651
APPLICATION FOR COURSE SUCCESS
```

student_course_id	course_id	student_id
61	5	86
62	3	87
63	4	87
64	5	87
65	7	87
66	1	88
67	2	88
68	5	88
69	1	89
70	2	89
71	3	89
72	4	89
90	3	88
NULL	NULL	NULL

첫번째 화면은 수강신청 정보를 입력받는 콘솔창 화면이고, 두번째 화면은 student_course테이블 화면이다. Student_course테이블에 데이터가 추가된 것을 확인할 수가 있다.

Ⅴ. 결론

gRPC를 이용해 클라이언트 서버 프로그래밍을 구현했다. 이 과정을 통해 클라이언트는 서비스를 요청하는 대상, 서버는 서비스를 제공하는 대상임을 알아볼 수 있었고 멀티프로세스를 이용하여 소프트웨어 시스템이 운용되는 방식인 분산 시스템에 대해 명확하게 이해할 수 있었다. 본 보고서는 gRPC를 이용하였지만, 서비스를 주고 받기 위한 통신 방법에는 gRPC뿐만 아니라 여러 가지가 있음을 찾아볼 수 있었다. Google에서 개발한 Remote Procedure Calls로 오픈소스 원격 프로시저 호출 시스템인 gRPC, 상대적으로 적은 리소스를 사용하고, 개방적이며 다양한 포맷의 데이터를 주고 받는 REST API, 불필요한 데이터를 받지 않고, 필요한 데이터만 받을 수 있는 GraphQL등이 있다.[4] 지금까지는 분산 시스템이라는 전체적인 숲을 이해했고, 다음으로는 더욱더 깊은 내용을 이해해보고 싶은 생각이 들었다. 프로세스에서 깊게 들어가 프로세스 안에서 동시에 로직을 실행할 수 있는 단위인 쓰레드를 여러 개 발생시킨 멀티 쓰레드에 대해 명확하게 이해해

[4] 『방구의 개발냄새』, https://bangu4.tistory.com/167

보고 싶어졌고 이를 위해선 본 보고서에서 진행했던 과정과 마찬가지로 명확한 이론과 체계적인 코딩과정이 필요하다고 판단된다.