

# Report

## 도메인 분석 및 SW 설계

- Elaboration phase 3-

교수님 : 이정태 교수님

제출 일 : 2020.07.12

조 이름 : 아무거나하조

이름	학번	분반
고예준	201820742	D
김민영	201720580	D
송지연	201820748	D
이승현	201720579	D
최정민	201820712	D

### Revision History

버전	일자	설명	저자
Elaboration phase 4-1	20.07.10	품질 개선을 위한 Design Pattern 적용 작성 수정/보완된 Use case Realization 작성 수정/보완된 Source Code 작성 Test 보고서 작성	아무거나하조
Elaboration phase 4-2	20.07.11	수정/보완된 Use case Realization 보완 수정/보완된 Source Code 보완 Test 보고서 보완	아무거나하조
Elaboration phase 4-3	20.07.12	Test 보고서 보완	아무거나하조

## <순서>

1. 품질 개선을 위한 Design Pattern 적용.....	1
1.1 Revision History .....	1
1.2 품질 개선을 위한 Design Pattern 적용 기능 및 이유.....	1
2. 수정/보완된 Use case realization .....	2
2.1 Revision history .....	2
2.2 수정/보완된 Design Interaction diagram .....	3
가) 예약 환자 접수 .....	3
나) 진료비 결제 .....	8
다) Start up Use case .....	12
2.3 수정/보완된 Design Class Diagram .....	13
가) 예약 환자 접수 .....	13
나) 진료비 결제 .....	17
다) Start Up Use Case .....	21
3. 수정/보완된 Source Code.....	24
3.1 Revision history .....	24
3.2 수정/보완된 Source Code 작성 근거.....	24
가) 진료비 결제 .....	24
나) Start up Use Case .....	25
3.3 수정/보완된 Source Code.....	26
가) Register.....	26
나) ReservationInformation.....	27
다) Reception.....	27
라) PatientInformation.....	28
마) DoctorInformation.....	28
바) TreatmentRecord.....	28
사) Payment.....	29
아) Money.....	29
자) payAdapter .....	30

차) 외부 결제 .....	30
카) System_manager .....	31
타) Hospital_manager.....	31
파) manager .....	32
하) Hospital .....	33
4. Test 보고서 .....	35
4.1 Revision history .....	35
4.2 Test 보고서 .....	35
가) 예약 환자 접수 .....	35
나) 진료비 결제 .....	38
다) Start up Use Case .....	40

## < 표 순서 >

표 1 Adapter_GRASP 적용 이유 .....	1
표 2 예약 환자 접수_RDD_makeNewReservationReceipt .....	3
표 3 예약 환자 접수_GRASP_makeNewReservationReceipt.....	3
표 4 예약 환자 접수_RDD_enterReservationNumber(reservationNumber).....	4
표 5 예약 환자 접수_GRASP_enterReservationNumber(reservationNumber) .....	4
표 6 예약 환자 접수_RDD_confirmReservationNumber .....	5
표 7 예약 환자 접수_GRASP_confirmReservationNumber .....	5
표 8 예약 환자 접수_RDD_confirmReservationInformation.....	6
표 9 예약 환자 접수_GRASP_confirmReservationInformation .....	6
표 10 예약 환자 접수_RDD_requestReceipt .....	7
표 11 예약 환자 접수_GRASP_requestReceipt .....	7
표 12 진료비 결제_RDD_makeNewPayment .....	8
표 13 진료비 결제_GRASP_makeNewPayment .....	8
표 14 진료비 결제_RDD_enterPatientInformation(name,securityNumber, address) .....	9
표 15 진료비 결제_GRASP_enterPatientInformation(name,securityNumber, address) .....	9
표 16 진료비 결제_RDD_confirmPatientInformation.....	10
표 17 진료비 결제_GRASP_confirmPatientInformation .....	10
표 18 진료비 결제_RDD_makePayment(amount) .....	11
표 19 진료비 결제_GRASP_makePayment(amount).....	11
표 20 Start Up Use Case _RDD .....	12
표 21 Start Up Use Case _GRASP.....	13

표 22 예약 환자 접수_Register Class 설계 근거 .....	13
표 23 예약 환자 접수_Reception Class 설계 근거 .....	14
표 24 예약 환자 접수_ReservationInformation Class 설계 근거 .....	14
표 25 예약 환자 접수_DoctorInformation Class 설계 근거 .....	15
표 26 예약 환자 접수_PatientInformation Class 설계 근거 .....	15
표 27 예약 환자 접수_relation 설계 근거 .....	15
표 28 예약 환자 접수_role 설계 근거 .....	16
표 29 진료비 결제_Register Class 설계 근거 .....	17
표 30 진료비 결제_TreatmentRecord Class 설계 근거 .....	18
표 31 진료비 결제_Payment Class 설계 근거 .....	18
표 32 진료비 결제_PayAdapter Class 설계 근거 .....	19
표 33 진료비 결제_relation 설계 근거 .....	19
표 34 진료비 결제_role 설계 근거 .....	20
표 35 Start Up Use Case_Hospital Class 설계 근거 .....	21
표 36 Start Up Use Case _Register Class 설계 근거 .....	21
표 37 Start Up Use Case _Manager Class 설계 근거 .....	21
표 38 Start Up Use Case_Hospital manager Class 설계 근거 .....	21
표 39 Start Up Use Case_System_manager Class 설계 근거 .....	22
표 40 Start Up Use Case_relation 설계 근거 .....	22
표 41 Start Up Use Case_role 설계 근거 .....	23
표 42 Test 보고서_예약 환자 접수 .....	35
표 43 Test 보고서_진료비 결제 .....	38
표 44 Test 보고서_Start up Use case .....	40

## <그림 순서>

그림 1 Adapter 적용 이유 .....	1
그림 2. 예약환자접수_ Sequence Diagram_makeNewReservationReceipt .....	4
그림 3. 예약환자접수_ Sequence Diagram_enterReservationNumber(reservationNumber) .....	5
그림 4. 예약환자접수_ Sequence Diagram_confirmReservationNumber .....	6
그림 5. 예약환자접수_ Sequence Diagram_confirmReservationInformation .....	7
그림 6. Sequence Diagram_예약접수_requestReceipt .....	8
그림 7 진료비 결제_Sequence Diagram_makeNewPayment .....	9
그림 8 진료비 결제_Sequence Diagram_enterPatientInformation(name,securityNumber, address) .....	10
그림 9 진료비 결제_Sequence Diagram_confirmPatientInformation .....	11
그림 10 진료비 결제_Sequence Diagram_makePayment(amount) .....	12
그림 11 Start Up Use Case_Communication Diagram .....	13
그림 12 예약 환자 접수_Class Diagram .....	17
그림 13 진료비 결제_Class Diagram .....	20

그림 14 Start Up Use Case_Class Diagram .....	23
그림 15 Source Code-Class Diagram mapping_진료비 결제.....	24
그림 16 Source Code-Class Diagram mapping_start up use case .....	25
그림 17 SourceCode_Register .....	26
그림 18 SourceCode_ReservationInformation .....	27
그림 19 SourceCode_Reception .....	27
그림 20 SourceCode_PatientInformation .....	28
그림 21 SourceCode_DoctorInformation .....	28
그림 22 SourceCode TreatmentRecord.....	28
그림 23 SourceCode_Payment.....	29
그림 24 SourceCode_Money .....	29
그림 25 SourceCode_InterfacePayAdapter .....	30
그림 26 SourceCode_ClassPayAdapter .....	30
그림 27 SourceCode_외부 결제.....	30
그림 28 SourceCode_System_manager .....	31
그림 29 SourceCode_Hospital_manager .....	31
그림 30 SourceCode_manager.....	32
그림 31 SourceCode_Hospital1 .....	33
그림 32 SourceCode_Hospital2 .....	34

## **<Revision History 순서>**

Revision History 1_품질 개선을 위한 Design Pattern 적용 .....	1
Revision History 2_수정/보완된 Use case realization.....	2
Revision History 3_수정/보완된 Source Code .....	24
Revision History 4_Test 보고서 .....	35

# 1. 품질 개선을 위한 Design Pattern 적용

## 1.1 Revision History

Revision History 1\_품질 개선을 위한 Design Pattern 적용

버전	일자	설명	저자
Elaboration phase 4-1	20.07.10	1.2 품질 개선을 위한 Design Pattern 적용 기능 및 이유 작성	아무거나하조

## 1.2 품질 개선을 위한 Design Pattern 적용 기능 및 이유

- Design Pattern 을 통하여 시스템 품질 개선이 가능하다고 판단한 부분  
진료비 결제 Use Case 에서 외부 시스템과 직접 연결하는 부분을 중간 접속자인 Adapter 를  
이용하여 시스템 품질 개선이 가능하다.

### - Adapter 적용 이유

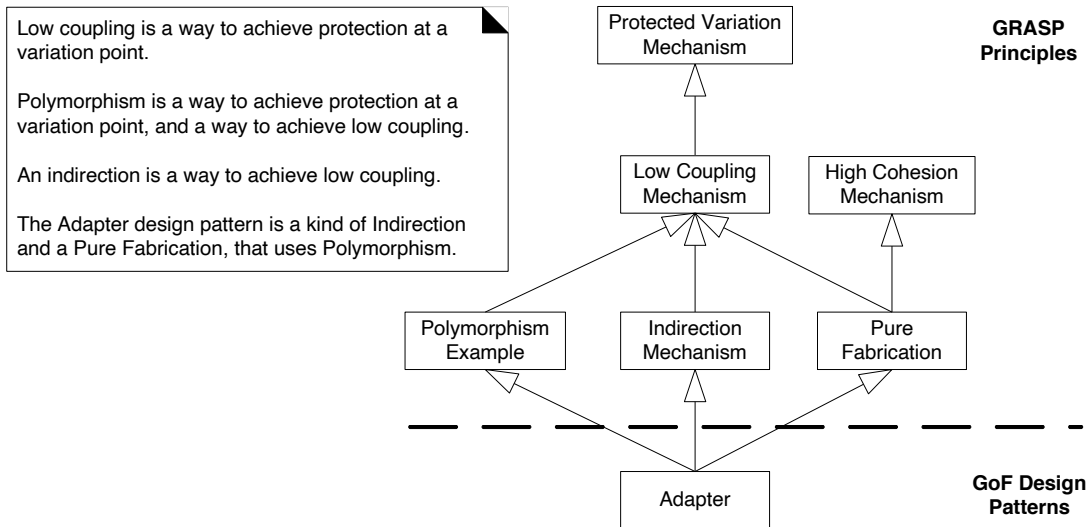


그림 1 Adapter 적용 이유

표 1 Adapter\_GRASP 적용 이유

More GRASP	적용 이유
<b>Polymorphism</b>	nonfunctional requirement 중 reliability 1 번인 “외부 서비스(전자 서명 장치, 처방전 출력 장치, 네트워크)에 오류가 발생하더라도 계속해서 접수/결제를 진행할 수 있도록 지원해야 한다.”에 의하여, 외부 결제 시스템 A 와의 연결이 끊어지면 바로 B 결제 시스템과의 연결을 허용해야 한다. 외부 시스템이 바뀔에 따라,

	외부 시스템과 연결되는 부분이 다 바뀌게 된다면, 이는 적응성이 굉장히 떨어진다. 따라서 Polymorphism pattern 을 적용하여 중간 접속자인 Adapter 를 이용한다면 변화에 대한 적응성을 높일 수 있다.
<b>Pure Fabrication</b>	Payment class 는 System event 에 대한 입력을 받아오는 Register, Patient Information, 외부 결제시스템과의 연결이 필요하다. 이 경우 과도한 연결로 low coupling 을 위배하므로, 결제 시스템과의 직접적인 연결이 아닌, 현실 세계에서는 없지만 SW 상에서 존재하는 중간 접속자인 Adapter 를 이용한다면, coupling 이 낮아져 재사용성을 높일 수 있다.
<b>Indirection</b>	Payment 가 외부시스템인 결제 시스템과 연결되어 있어야 하는데 이는 direct coupling 이다. 하지만 결제 시스템의 경우 바뀔 가능성이 존재하는 외부 시스템이다. 따라서 중간에 adapter 라는 객체를 추가하여, adapter 를 통해 연결하도록 하여 direct coupling 을 방지할 수 있다.
<b>Protected Variation</b>	외부 시스템의 변화에 대해 내부 시스템이 변화를 받으면 안된다. 이 때 결제시스템은 외부 시스템이고 변동 가능성이 있기 때문에 중간 연결자인 adapter 를 이용하여 새로운 외부 시스템과 연결을 해야할 경우가 발생하더라도 쉽게 확장이 가능하고 내부 시스템을 보호할 수 있다.

따라서, 외부 시스템인 결제 시스템과 연결하기 위하여 중간 접속자인 Adapter 를 이용하면, 위와 같이 GRASP 의 Polymorphism, Pure Fabrication, Indirection, Protected Variation Pattern 을 적용할 수 있으므로, 변화에 대한 적응성을 높일 수 있고, 유지 보수가 편리해지며, 재사용성이 증가하므로, 무인 접수 및 결제 시스템의 품질 개선이 가능하다.

## 2. 수정/보완된 Use case realization

### 2.1 Revision history

Revision History 2\_수정/보완된 Use case realization

버전	일자	설명	저자
Elaboration phase 4-1	20.07.10	2.2 수정/보완된 Design Interaction diagram 작성	아무거나하조



		2.3 수정/ 보완된 Design Class diagram 작성	
Elaboration phase 4-2	20.07.11	2.2 수정/보완된 Design Interaction diagram 보완 2.3 수정/ 보완된 Design Class diagram 보완	아무거나하조

## 2.2 수정/보완된 Design Interaction diagram

### 가) 예약 환자 접수

#### 1) makeNewReservationReceipt

- RDD
  - reservation receipt 를 처리할 환경 구축
  - reservation receipt 를 담당할 객체 생성

**표 2 예약 환자 접수\_RDD\_makeNewReservationReceipt**

<b>Doing responsibilities</b>	0. Register 는 makeNewReservationReceipt system event 를 받을 responsibility 가 있다. 1. Register 는 Reception 을 생성할 responsibility 가 있다.
<b>Knowing responsibilities</b>	없음.

- GRASP pattern

**표 3 예약 환자 접수\_GRASP\_makeNewReservationReceipt**

<b>0</b>	Who will be responsible for handling the "makeNewReservationReceipt" system event? - Register 가 전체 시스템을 represent 하므로 <b>controller pattern</b> 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
<b>1</b>	Who will be responsible for creating Reception? - Register 가 Reception 을 initialize 하기 위한 데이터를 모두 가지고 있을 예정이기 때문에 <b>creator pattern</b> 을 적용하여 register 에게 reception 을 생성할 responsibility 를 준다.

- Sequence Diagram

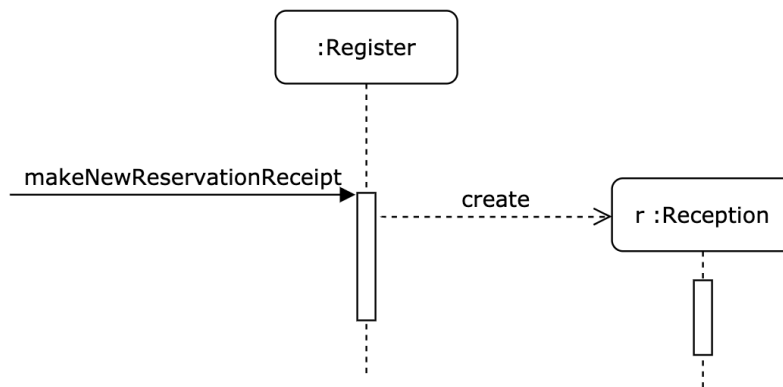


그림 2. 예약환자접수\_ Sequence Diagram\_makeNewReservationReceipt

2) enterReservationNumber(reservationNumber)

- RDD

- 입력한 reservation number 를 보여준다.

표 4 예약 환자 접수\_RDD\_enterReservationNumber(reservationNumber)

<b>Doing responsibilities</b>	0. Register 는 enterReservationNumber system event 를 받을 responsibility 가 있다.
<b>Knowing responsibilities</b>	1a. Reception 은 입력된 reservation number 에 대한 정보를 알 필요가 있다. 1b. Register 는 reservation number 에 대한 정보를 알 필요가 있다.

- GRASP pattern

표 5 예약 환자 접수\_GRASP\_enterReservationNumber(reservationNumber)

<b>0</b>	Who will be responsible for handling the "enterReservationNumber" system event? - Register 가 전체 시스템을 represent 하므로 <b>controller pattern</b> 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
<b>1</b>	Who will be responsible for knowing reservation number? a. Reception 이 reservation number 를 알 responsibility 가 있다. - Reception 은 접수를 목적으로 하는 객체이므로 reservation number 를 알 responsibility 를 부여하면 이는 high cohesion, low coupling, SOC 를 만족한다. b. Register 가 reservation number 를 알 responsibility 가 있다. - Register 는 reservation number 를 받아서 알고 있는 상태이므로 <b>knowledge expert pattern</b> 을 적용하여 Register 에게 reservation number 를 알 responsibility 를 부여할 수 있다. 하지만 Register 는 사용자의 input system event 를 handle 을 목적으로 하는 객체이므로 reservation number 를 알

	<p>responsibility 를 부여하면 이는 low cohesion, high coupling 이므로 register 에 부여하는 것은 올바르지 않다.</p> <p>--&gt; Reception 가 <b>high cohesion pattern, low coupling pattern</b> 을 적용하여 Reception 에게 reservation number 를 알 responsibility 를 준다.</p>
--	--

- Sequence Diagram

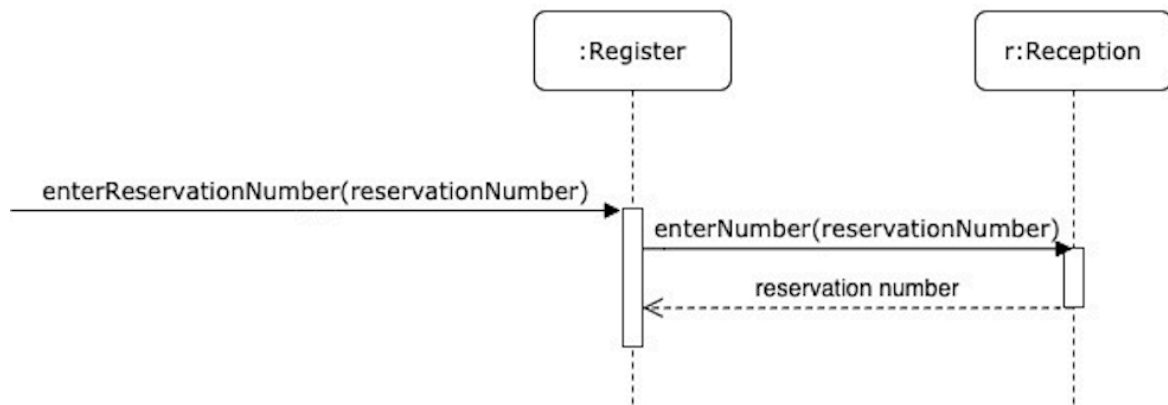


그림 3. 예약환자접수\_ Sequence Diagram\_enterReservationNumber(reservationNumber)

### 3) confirmReservationNumber

- RDD

- reservation number 에 해당하는 reservation 정보를 가져온다.

표 6 예약 환자 접수\_RDD\_confirmReservationNumber

<b>Doing responsibilities</b>	<p>0. Register 는 confirmReservationNumber system event 를 받을 responsibility 가 있다.</p> <p>2. 예약 시스템은 reservation number 에 해당하는 reservation information 을 알려줄 responsibility 가 있다.</p>
<b>Knowing responsibilities</b>	<p>1. Reception 은 reservation number 를 알 responsibility 가 있다.</p>

- GRASP pattern

표 7 예약 환자 접수\_GRASP\_confirmReservationNumber

<b>0</b>	<p>Who will be responsible for handling the "confirmReservationNumber" system event?</p> <ul style="list-style-type: none"> <li>- Register 가 전체 시스템을 represent 하므로 <b>controller pattern</b> 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.</li> </ul>
<b>1</b>	<p>Who will be responsible for knowing reservation Information?</p> <ul style="list-style-type: none"> <li>- Reception 은 접수를 목적으로 하고 접수를 하기 위해 reservation information 이 필요하다. 그러므로 <b>information expert pattern, high</b></li> </ul>

	<b>cohesion pattern, low coupling pattern</b> 을 적용하여 Reception 에게 reservation information 을 알 responsibility 를 준다.
2	Who will be responsible for getting reservation information? <ul style="list-style-type: none"> <li>- 예약 시스템은 reservation information 을 알아내는 일을 처리하는 데에 전문적인 정보를 가장 많이 가지고 있으므로 <b>information expert pattern</b> 을 적용하여 예약 시스템에게 reservation information 을 알아내는 responsibility 를 준다.</li> </ul>

- Sequence Diagram

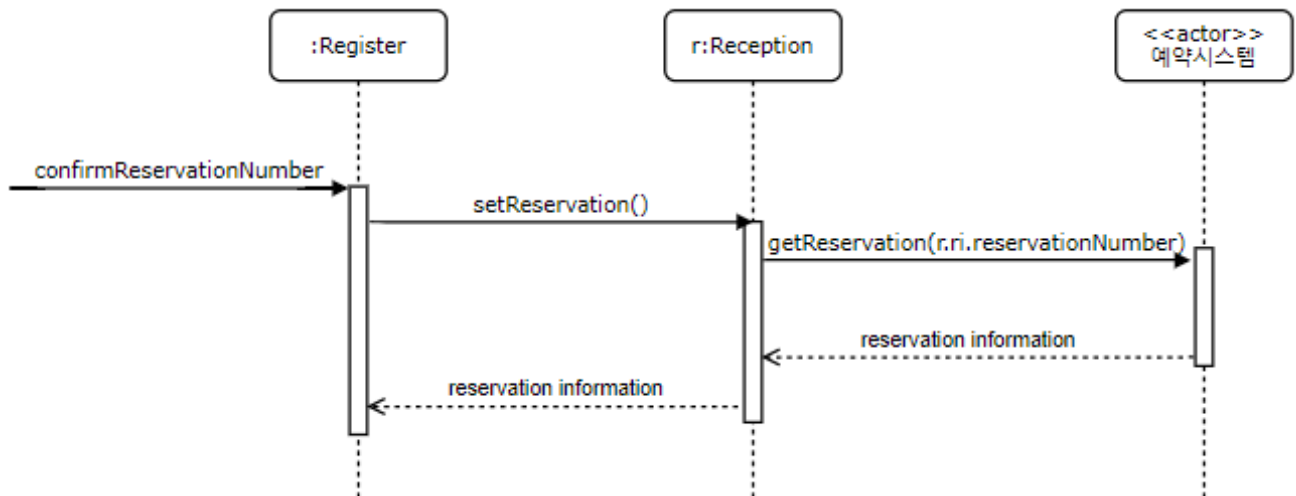


그림 4. 예약환자접수\_ Sequence Diagram\_confirmReservationNumber

#### 4) confirmReservationInformation

- RDD

표 8 예약 환자 접수\_RDD\_confirmReservationInformation

<b>Doing responsibilities</b>	0. Register 는 confirmReservationInformation system event 를 받을 responsibility 가 있다.
<b>Knowing responsibilities</b>	1. Reception 은 해당 ReservationInformation 이 맞다는 정보를 알 responsibility 가 있다.

- GRASP pattern

표 9 예약 환자 접수\_GRASP\_confirmReservationInformation

0	Who will be responsible for handling the "confirmReservationInformation" system event? <ul style="list-style-type: none"> <li>- Register 가 전체 시스템을 represent 하므로 <b>controller pattern</b> 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.</li> </ul>
1	Who will be responsible for knowing that the Reservation Information is correct?

	<ul style="list-style-type: none"> <li>- Reception 은 접수가 목적인 객체이므로 해당 정보를 가지고 있을 responsibility 가 있다. <b>High cohesion pattern</b> 과 <b>knowledge expert pattern</b> 을 적용하여 Reception 에게 reservation information 이 맞음을 알 responsibility 를 준다.</li> </ul>
--	--

- Sequence Diagram

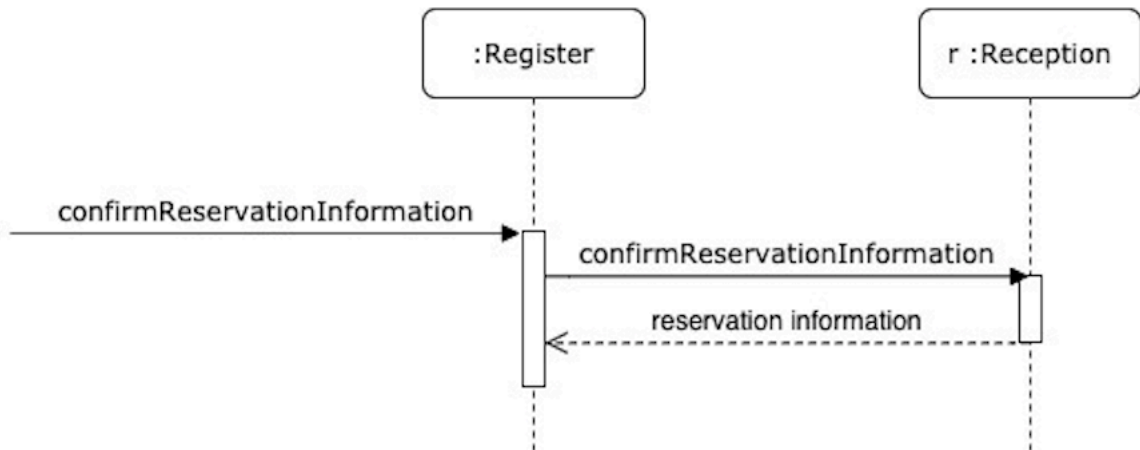


그림 5. 예약환자접수\_ Sequence Diagram\_confirmReservationInformation

#### 5) requestReceipt

- RDD

표 10 예약 환자 접수\_RDD\_requestReceipt

<b>Doing responsibilities</b>	0. Register 는 requestReceipt system event 를 받을 responsibility 가 있다.
<b>Knowing responsibilities</b>	1. Reception 은 접수를 처리할 responsibility 가 있다.

- GRASP pattern

표 11 예약 환자 접수\_GRASP\_requestReceipt

<b>0</b>	Who will be responsible for handling the "confirmReservationNumber" system event? <ul style="list-style-type: none"> <li>- Register 가 전체 시스템을 represent 하므로 <b>controller pattern</b> 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.</li> </ul>
<b>1</b>	Who will be responsible for process reception? <ul style="list-style-type: none"> <li>- Reception 은 접수가 목적인 객체이므로 접수에 관한 모든 정보를 가지고 있다. <b>High cohesion pattern</b> 과 <b>knowledge expert pattern</b> 을 적용하여 Reception 에게 접수를 처리할 responsibility 를 준다.</li> </ul>

- Sequence Diagram

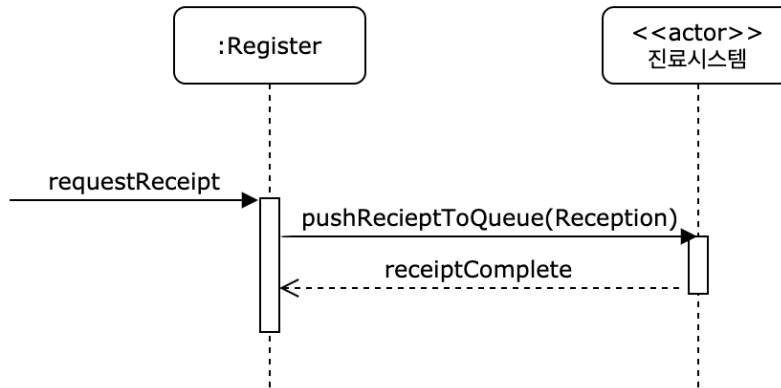


그림 6. Sequence Diagram\_예약접수\_requestReceipt

#### 나) 진료비 결제

- 1) makeNewPayment
  - RDD

표 12 진료비 결제\_RDD\_makeNewPayment

<b>Doing responsibilities</b>	0. Register 는 makeNewPayment system event 를 받을 responsibility 가 있다. 1. Register 는 Payment 를 생성할 responsibility 가 있다.
<b>Knowing responsibilities</b>	X

- GRASP pattern

표 13 진료비 결제\_GRASP\_makeNewPayment

<b>0</b>	Who will be responsible for handling the "makeNewPayment" system event? - Register 가 전체 시스템을 represent 하고 제일 처음 받아서 처리하기 때문에 <b>controller pattern</b> 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
<b>1</b>	Who will be responsible for creating Payment? - Register 가 Payment 를 initialize 하기 위한 데이터를 모두 가지고 있고 Payment 를 record 할 예정이기 때문에 <b>creator pattern</b> 을 적용하여 Register 에게 Payment 를 생성할 responsibility 를 준다.

- Sequence Diagram

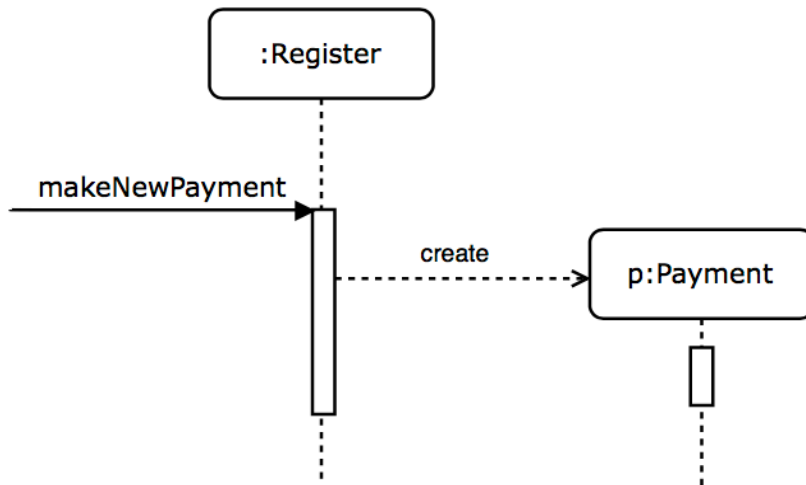


그림 7 진료비 결제\_Sequence Diagram\_makeNewPayment

2) enterPatientInformation(name, securityNumber, address)

- RDD

표 14 진료비 결제\_RDD\_enterPatientInformation(name,securityNumber, address)

<b>Doing responsibilities</b>	0. Register 는 enterPatientInformation system event 를 받을 responsibility 가 있다. 1. Payment 는 Patient Information 의 인스턴스 pi 를 생성할 responsibility 가 있다.
<b>Knowing responsibilities</b>	2. Payment 는 Patient Information 을 알 responsibility 가 있다. 3. Register 는 Patient Information 을 알 responsibility 가 있다.

- GRASP pattern

표 15 진료비 결제\_GRASP\_enterPatientInformation(name,securityNumber, address)

<b>0</b>	Who will be responsible for handling the "enterPatientInformation" system event? - Register 가 전체 시스템을 represent 하고 제일 처음 받아서 처리하기 때문에 <b>controller pattern</b> 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
<b>1</b>	Who will be responsible for creating PatientInformation? - Payment 는 Patient Information 을 initialize 하기 위한 정보를 많이 가지고 있기 때문에 <b>Information Expert pattern</b> 을 적용하여 Payment 에게 Patient Information 을 생성할 responsibility 를 준다.

- Sequence Diagram

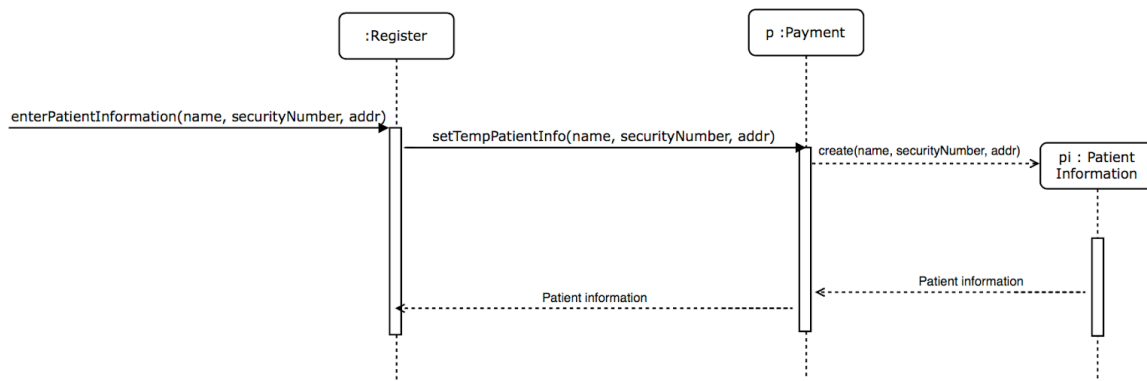


그림 8 진료비 결제\_Sequence Diagram\_enterPatientInformation(name,securityNumber, address)

3) confirmPatientInformation

- RDD

표 16 진료비 결제\_RDD\_confirmPatientInformation

<b>Doing responsibilities</b>	0. Register 는 confirmPatientInformation system event 를 받을 responsibility 가 있다. 1. Payment 는 외부 시스템 EMR 을 제어할 responsibility 가 있다.
<b>Knowing responsibilities</b>	2. Payment 는 Patient Information, total treatment fee 을 알 responsibility 가 있다. 3. Register 는 total treatment fee 을 알 responsibility 가 있다.

- GRASP pattern

표 17 진료비 결제\_GRASP\_confirmPatientInformation

<b>0</b>	Who will be responsible for handling the “confirmPatientInformation” system event? - Register 가 전체 시스템을 represent 하고 제일 처음 받아서 처리하기 때문에 <b>controller pattern</b> 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
<b>1</b>	Who will be responsible for getting patientInformation from EMR and knowing patientInformation? - Payment 는 결제가 목적인 객체이므로 결제에 관한 모든 정보를 가지고 있어야 한다. 그리고 결제를 하기 위해서는 patientInformation 을 알 필요가 있다. 그러므로 <b>High cohesion pattern</b> 과 <b>knowledge expert pattern</b> 을 적용하여 Payment 에게 patientInformation 을 가져오고 알 responsibility 를 준다.



- Sequence Diagram

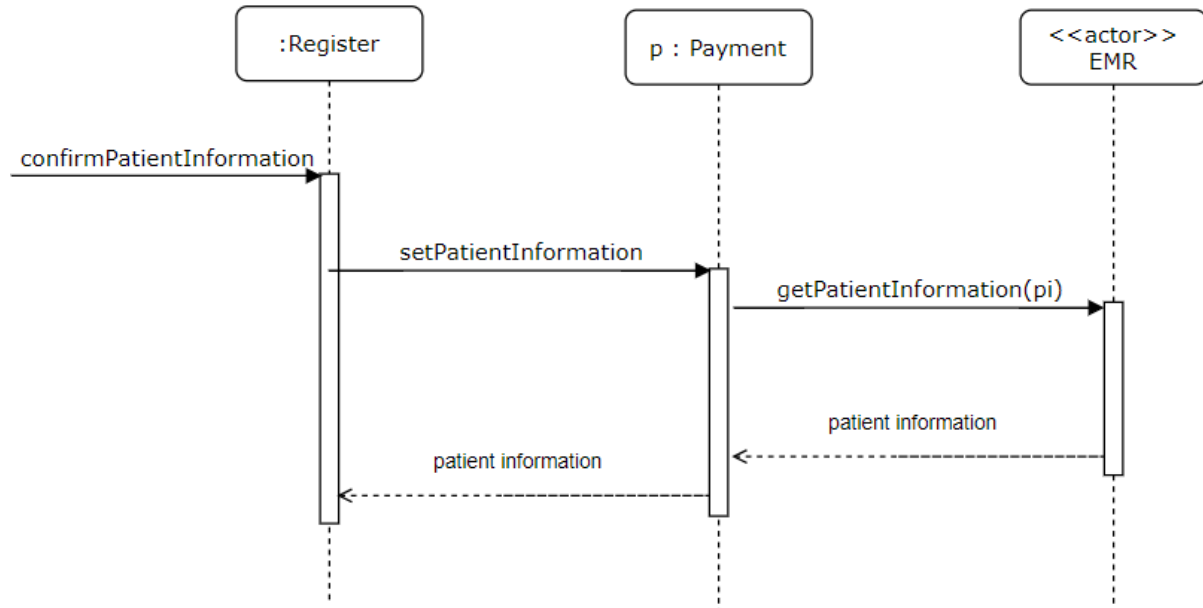


그림 9 진료비 결제\_Sequence Diagram\_confirmPatientInformation

4) makePayment(amount)

- RDD

표 18 진료비 결제\_RDD\_makePayment(amount)

<b>Doing responsibilities</b>	0. Register 는 makePayment(amount) system event 를 받을 responsibility 가 있다. 1. Payment 는 외부 시스템 결제시스템을 제어할 responsibility 가 있다. 2. Payment 는 TreatmentRecord 에 결제 내역을 기록할 responsibility 가 있다.
<b>Knowing responsibilities</b>	0. Payment 는 PaymentComplete 를 알 responsibility 가 있다. 1. Register 는 PaymentComplete 를 알 responsibility 가 있다. 2. TreatmentRecord 는 해당 Payment 를 알 responsibility 가 있다.

- GRASP pattern

표 19 진료비 결제\_GRASP\_makePayment(amount)

<b>0</b>	Who will be responsible for handling the "makePayment(amount)" system event? - Register 가 전체 시스템을 represent 하고 제일 처음 받아서 처리하기 때문에 <b>controller pattern</b> 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
<b>1</b>	Who will be responsible for handling Payment?

	<ul style="list-style-type: none"> <li>- 결제 시스템은 외부 시스템이기 때문에 변화가능성이 크다. 따라서 다양한 결제 시스템을 처리할 responsibility 가 있다. <b><u>Polymorphism, pure fabrication, indirection, protected variations pattern</u></b> 을 적용해서 PayAdapter interface 를 기반으로 pay 함수를 다형적으로 활용한다.</li> </ul>
2	<p>Who will responsible for knowing Payment for treatment is finished?</p> <ul style="list-style-type: none"> <li>- TreatmentRecord 는 진료에 대한 정보를 저장하는 것이 목적이다. 그러므로 <b><u>knowledge expert pattern, high cohesion pattern</u></b> 을 적용해서 TreatmentRecord 가 해당 진료의 결제에 대한 내용을 저장할 responsibility 를 준다.</li> </ul>

- Sequence Diagram

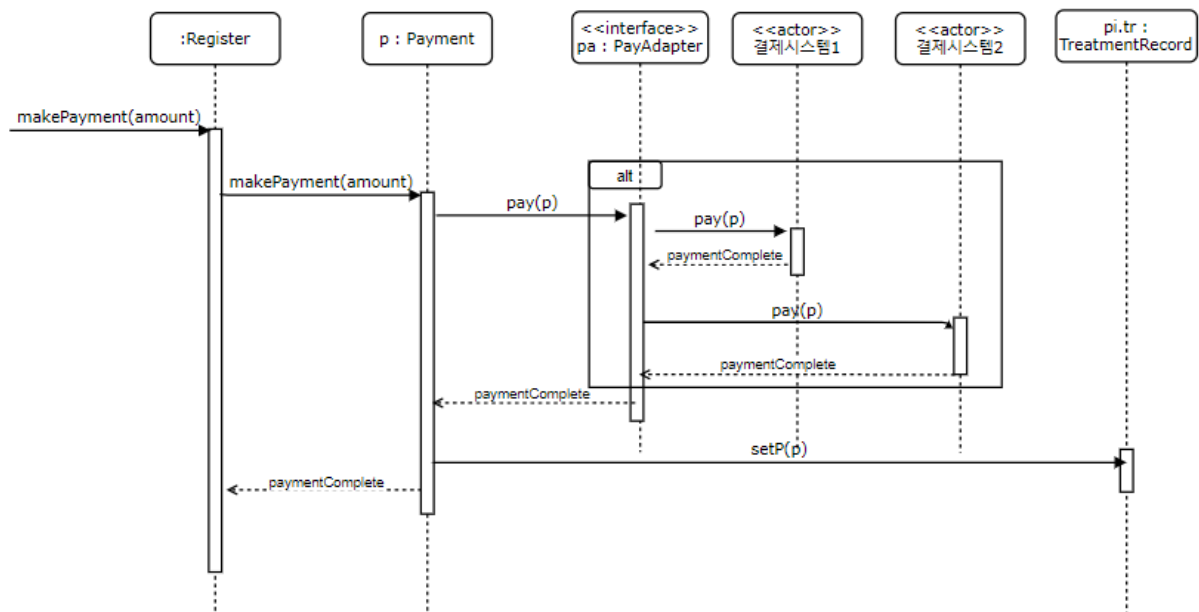


그림 10 진료비 결제\_Sequence Diagram\_makePayment(amount)

다) Start up Use case

- RDD

표 20 Start Up Use Case \_RDD

<b>Doing responsibilities</b>	0. 병원은 Register, Hospital_manager, System_manager 를 생성할 responsibility 가 있다.
<b>Knowing responsibilities</b>	1. 병원은 해당 Hospital_manager 배열과, System_manager 배열을 알 responsibility 가 있다.

- GRASP pattern

표 21 Start Up Use Case \_GRASP

0	모든 과정을 수행하기 위해 Hospital 을 생성해야 하므로, <b>creator pattern</b> 을 적용하여 start up use case 에서 Hospital 을 생성한다.
1	Register class 를 통해 모든 과정들이 시작되기 때문에 <b>creator pattern</b> 을 적용하여 start up use case 에서 Register 객체를 생성해야 한다.
2	진료비 결제 use case 기능을 제공하기 위해서 병원 관리자가 등록되어야 한다. 따라서 <b>creator pattern</b> 을 적용하여 start up use case 에서 병원 관리자 객체를 만들고 병원 관리자 객체 배열에 병원 관리자 목록을 저장한다.
3	진료비 결제 use case 기능을 제공하기 위해서 시스템 관리자가 등록되어야 한다. 따라서 <b>creator pattern</b> 을 적용하여 start up use case 에서 시스템 관리자 객체를 만들고 시스템 관리자 객체 배열에 시스템 관리자 목록을 저장한다.

- Communication Diagram

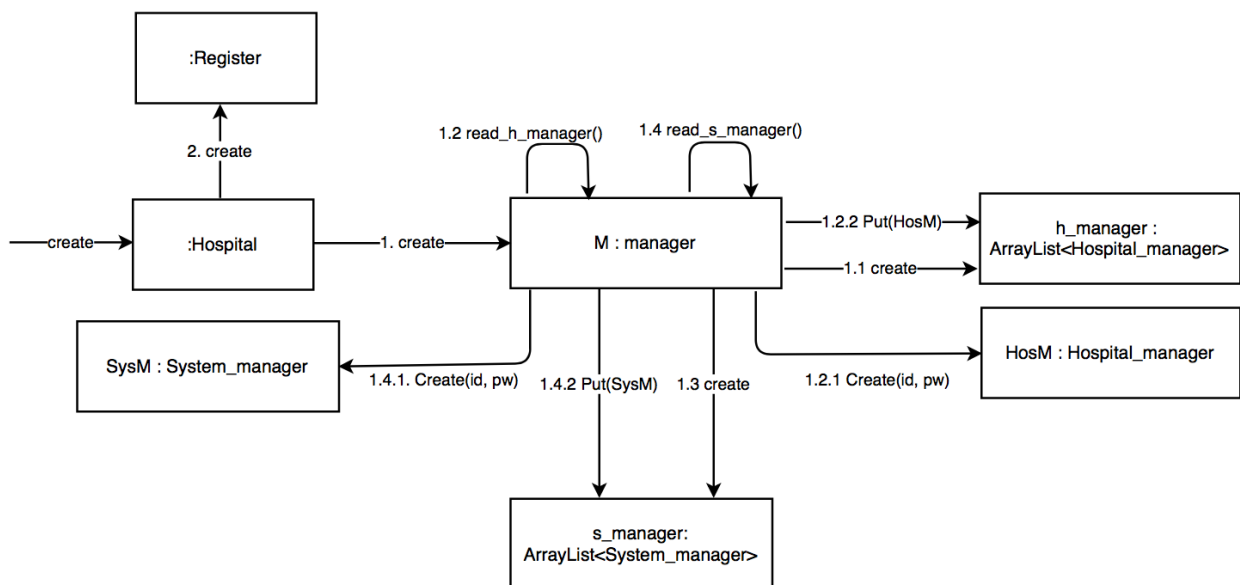


그림 11 Start Up Use Case\_Communication Diagram

## 2.3 수정/보완된 Design Class Diagram

### 가) 예약 환자 접수

- class name, attribute, operation 설계 근거

표 22 예약 환자 접수\_Register Class 설계 근거

Register	
name	해당 class 는 input system event 를 처리하는 controller 로 전체 시스템을 설명할 수 있는 것을 이름으로 하는 것이 좋다. Register 는 병원에서 접수

	및 결제를 처리하는 데스크라는 뜻이 있다. 그러므로 register 는 접수 및 결제를 처리하는 시스템을 전체 설명할 수 있다고 생각하므로 이를 선택했다.
<b>attributes</b>	X
<b>operation</b>	<p>makeNewReservationReceipt() : 예약 환자의 접수를 시작해야 하므로 이를 operation 으로 선정하였다.</p> <p>enterReservationNumber(ReservationNumber) : 예약 환자의 접수를 위해 예약 번호를 입력받아야 하므로 이를 operation 으로 선정하였다.</p> <p>confirmReservationNumber() : 예약 환자의 접수를 위해 입력된 예약 번호가 맞는지 확인해야 하므로 이를 operation 으로 선정하였다.</p> <p>confirmReservationInformation() : 예약 환자의 접수를 위해 예약 정보가 맞는지 확인해야 하므로 이를 operation 으로 선정하였다.</p> <p>requestReceipt() : 환자가 입력한 접수 내용을 외부시스템에 보내서 대기줄에 넣을 수 있어야 하므로 이를 operation 으로 선정하였다.</p> <p>이는 sequence diagram 에서 가져왔다.</p>

표 23 예약 환자 접수\_Reception Class 설계 근거

Reception	
<b>name</b>	해당 class 는 접수에 필요한 정보를 저장하고 접수를 처리하는 데 사용된다. 목적이 접수이므로 이의 영문인 reception 을 선택했다.
<b>attributes</b>	<p>IsComplete : 해당 접수가 끝났다는 정보가 있어야 register 에서 다음 일을 처리할 수 있음을 알 수 있다. 이는 접수에 관한 내용이므로 이를 Reception 의 attribute 로 선정했다.</p> <p>이의 근거로는 domain model 을 들 수 있다.</p>
<b>operation</b>	<p>enterNumber(reservationNumber) : 예약 번호를 입력받아야 하므로 이를 operation 으로 선정하였다.</p> <p>setReservation() : 예약 정보를 외부 시스템에서 찾아와야 하므로 이를 operation 으로 선정하였다..</p> <p>confirmReservationInformation() : 해당 예약 정보가 맞는지 사용자에게 확인 받아야 하므로 이를 operation 으로 선정하였다.</p> <p>이는 sequence diagram 에서 가져왔다.</p>

표 24 예약 환자 접수\_ReservationInformation Class 설계 근거

ReservationInformation	
<b>name</b>	해당 class 는 환자가 예약한 정보를 담고 있다. 따라서 환자의 예약 정보를 나타내는 이름인 ReservationInformation 을 선택했다.
<b>attributes</b>	<p>reservationNumber : 예약 정보에 예약 번호가 있어야 예약 접수를 진행할 수 있다. 예약 번호는 예약 정보에 포함되므로, 이를 ReservationInformation 의 attribute 로 선정했다.</p>

	<p>date : 예약 정보에서 예약한 날짜를 저장하는 것은 필수이므로, 이를 ReservationInformation 의 attribute 로 선정했다.</p> <p>time : 예약 정보에서 예약한 시간을 저장하는 것은 필수이므로 이를 ReservationInformation 의 attribute 로 선정했다.</p> <p>이의 근거로는 domain model 을 들 수 있다.</p>
<b>operation</b>	X

**표 25 예약 환자 접수\_DoctorInformation Class 설계 근거**

<b>DoctorInformation</b>	
<b>name</b>	해당 class 는 의사 이름과 진료 날짜를 저장하고 있다. 따라서 의사 정보를 나타내는 이름인 DoctorInformation 을 선택했다.
<b>attributes</b>	<p>name : 의사 이름은 의사의 정보이므로, 이를 DoctorInformation 의 attribute 로 선정했다.</p> <p>workingDate : 의사마다 진료 날짜 정보를 담고 있으므로, 이를 DoctorInformation 의 attribute 로 선정했다.</p> <p>이의 근거로는 domain model 을 들 수 있다.</p>
<b>operation</b>	X

**표 26 예약 환자 접수\_PatientInformation Class 설계 근거**

<b>PatientInformation</b>	
<b>name</b>	해당 class 는 환자 정보를 저장하는 데 사용된다. 그러므로 환자 정보를 영문으로 한 PatientInformation 을 선택했다.
<b>attributes</b>	<p>name : 환자 이름은 환자의 정보에서 필수적이므로 이를 PatientInformation 의 attribute 로 선정했다.</p> <p>securityNumber : 환자 주민 번호는 환자의 정보에서 필수적이므로 이를 PatientInformation 의 attribute 로 선정했다.</p> <p>addr : 환자 주소는 환자의 정보에서 필수적이므로 이를 PatientInformation 의 attribute 로 선정했다.</p> <p>이의 근거로는 domain model 을 들 수 있다.</p>
<b>operation</b>	X

- relation 설계 근거

**표 27 예약 환자 접수\_relation 설계 근거**

<b>Register-Reception</b>	Register 는 예약 접수를 처리하므로 Register 와 Reception 은 association 을 가져야 한다. 이의 근거로는 domain model 을 들 수 있다.
<b>Reception-ReservationInformation</b>	예약 접수가 처리될 때 예약 번호를 이용하여 해당 예약 정보가 반환되어야 하기 때문에 Reception 과

	ReservationInformation 은 association 을 가져야 한다. 이의 근거로는 domain model 을 들 수 있다.
<b>ReservationInformation-DoctorInformation</b>	예약 정보는 의사정보를 포함하며, 예약 정보가 반환될 때, 예약 진료의 의사 정보가 포함되기 때문에 ReservationInformation 과 DoctorInformation 은 association 을 가져야 한다. 이의 근거로는 domain model 을 들 수 있다.
<b>ReservationInformation-PatientInformation</b>	예약 정보는 환자 정보를 포함하며, 예약 정보가 반환될 때, 예약 환자의 정보가 포함되기 때문에 ReservationInformation 과 PatientInformation 은 association 을 가져야 한다. 이의 근거로는 domain model 을 들 수 있다.

- role 설계 근거

**표 28 예약 환자 접수\_role 설계 근거**

<b>Register-Reception</b>	Register 는 예약 접수를 처리하며 Reception 객체는 환자의 예약 접수를 수행하는 역할을 하므로 Reception 을 r 의 role 을 주었다.
<b>Reception-ReservationInformation</b>	예약 접수가 처리될 때 예약 번호를 이용하기 때문에 Reception 에 해당 예약 정보가 반환되므로 ReservationInformation 을 줄인 ri 의 role 을 주었다.
<b>ReservationInformation-DoctorInformation</b>	예약 진료를 담당하는 의사 정보인 DoctorInformation 객체는 ReservationInformation 에서 예약 진료를 담당하는 의사 정보를 가지고 있는 역할을 하여 doctorInformation 을 줄인 di 의 role 을 주었다.
<b>ReservationInformation-PatientInformation</b>	예약 진료를 받는 환자 정보인 PatientInformation 객체는 ReservationInformation 에서 예약 진료를 받는 환자 정보를 가지고 있는 역할을 하여 PatientInformation 을 줄인 Pi 의 role 을 주었다.

- Class Diagram

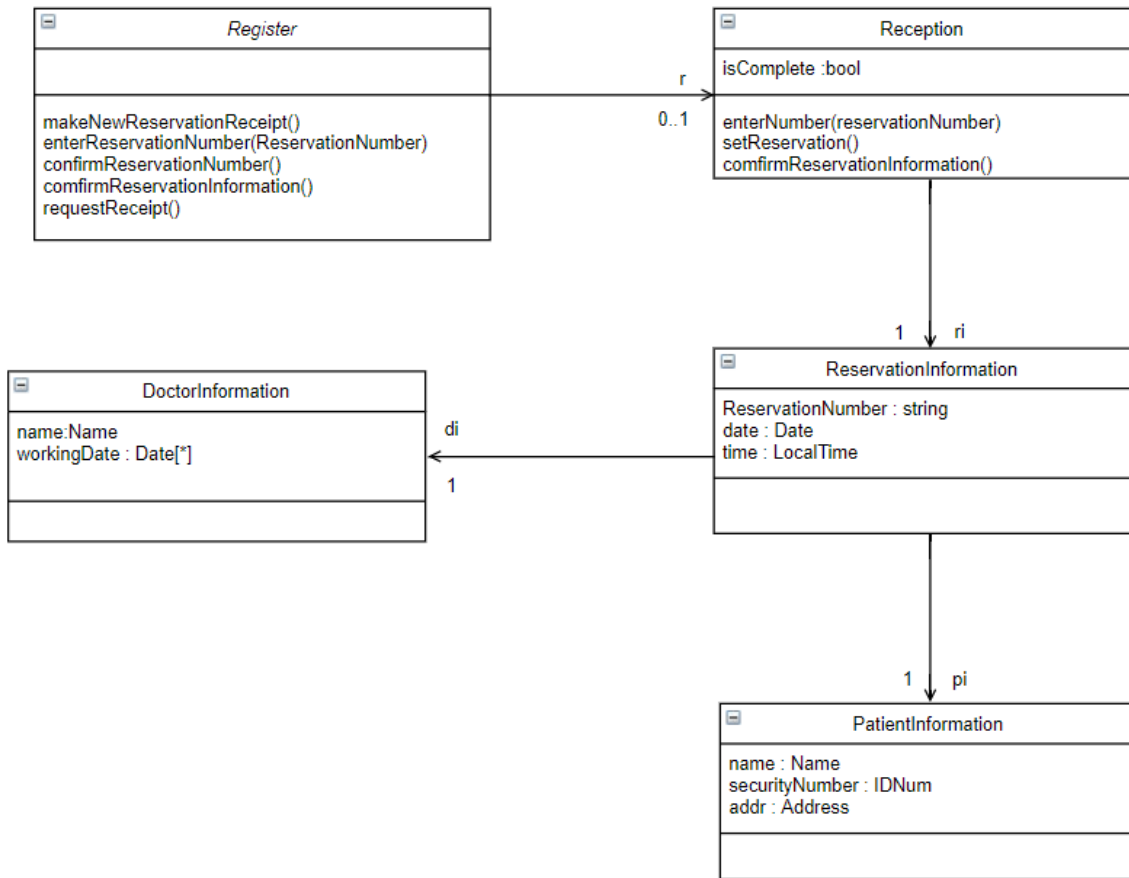


그림 12 예약 환자 접수\_Class Diagram

나) 진료비 결제

- class name, attribute, operation 설계 근거
  - 예약 환자 접수와 진료비 결제에서 PatientInformation class 의 역할이 같으므로 진료비 결제에서는 PatientInformation 설계 근거를 명시하지 않았다.

표 29 진료비 결제\_Register Class 설계 근거

Register	
name	해당 class 는 input system event 를 처리하는 controller 로 전체 시스템을 설명할 수 있는 것을 이름으로 하는 것이 좋다. Register 는 병원에서 접수 및 결제를 처리하는 데스크라는 뜻이 있다. 그러므로 register 는 접수 및 결제를 처리하는 시스템을 전체 설명할 수 있다고 생각하므로 이를 선택했다.
attributes	X

<b>operation</b>	<p>makeNewPayment() : 결제를 시작할 때 payment 객체를 만들기 위해서 필요하므로, 이를 operation 으로 선정하였다.</p> <p>enterPatientInformation(name,securityNumber, addr) : 결제를 시작할 때 사용자의 정보를 입력 받아야 하므로, 이를 operation 으로 선정하였다.</p> <p>confirmPatientInformation() : 환자로부터 결제할 금액을 입력 받고 결제를 처리해야 하므로, 이를 operation 으로 선정하였다.</p> <p>makePayment(amount) : 환자로부터 결제할 금액을 입력 받고 결제를 처리해야 하므로, 이를 operation 으로 선정하였다.</p> <p>이의 근거로는 system sequence diagram 과 sequence diagram 를 들 수 있다.</p>
------------------	---

**표 30 진료비 결제\_TreatmentRecord Class 설계 근거**

<b>TreatmentRecord</b>	
<b>name</b>	해당 class 는 진료 비용을 저장하기 위한 class 이므로, 진료 정보를 담고 있다는 의미를 가진 TreatmentRecord 을 선택했다.
<b>attributes</b>	<p>fee : TreatmentRecord 는 진료의 정보를 가지고 있으므로 해당 진료비를 저장하는 것이 적합하므로, 이를 TreatmentRecord 의 attribute 로 선정했다.</p> <p>이의 근거로는 domain model 을 들 수 있다.</p>
<b>operation</b>	<p>setP(p) : TreatmentRecord 에 해당 진료비가 결제되었는지 기록해야 하기 때문에 이를 operation 으로 선정하였다.</p> <p>이의 근거로는 sequence diagram 을 들 수 있다.</p>

**표 31 진료비 결제\_Payment Class 설계 근거**

<b>Payment</b>	
<b>name</b>	해당 class 는 진료 정보와, 환자 정보, 결제가 완료 되었는지를 저장하기 위한 class 이므로, 결제의 의미를 가진 Payment 를 선택했다.
<b>attributes</b>	<p>amount : Payment 는 진료비용을 저장해야 하므로, 이를 Payment 의 attribute 로 선정했다.</p> <p>paymentComplete : Payment 는 진료가 정상적으로 완료되었는지를 반환해야 하므로, 이를 Payment 의 attribute 로 선정했다.</p>
<b>operation</b>	<p>setTempPatientInfo(name, securityNumber, addr) : 사용자에게 개인 정보를 입력받아야 하므로, 이를 operation 으로 선정하였다.</p> <p>setPatientInformation() : 환자의 정보가 저장되어 있는 외부 시스템의 기록과 환자가 입력한 정보가 일치하는지 확인해야 하므로, 이를 operation 으로 선정하였다.</p>



	makePayment(amount) : 지불할 금액을 환자로부터 입력받아서 외부 시스템과 연결해야 하므로, 이를 operation 으로 선정하였다. 이는 sequence diagram 에서 가져왔다.
--	---

표 32 진료비 결제\_PayAdapter Class 설계 근거

PayAdapter	
name	해당 class 는 외부 결제 시스템이 변동 가능성이 있기 때문에 내부에서 인터페이스를 구현해서 외부 시스템과 연동시켜야 하는 class 이므로, 결제를 돕는다는 의미를 가진 PayAdapter 를 선택했다.
attributes	X
operation	pay() : 오이부 결제 시스템과 내부시스템이 연동해야 하므로, 이를 operation 으로 선정하였다. 이는 sequence diagram 에서 가져왔다.

- relation 설계 근거

표 33 진료비 결제\_relation 설계 근거

TreatmentRecord - PatientInformation	진료를 완료한 환자 정보를 담을 수 있는 PatientInformation 객체는 TreatmentRecord 에서 해당 환자의 진료비 정보를 읽어와서 배열 형태로 저장해야 하기 때문에 PatientInformation 과 TreatmentRecord 는 association 을 가져야 한다. 이의 근거로는 domain model 을 들 수 있다.
PatientInformation - Payment	결제는 환자 정보를 포함하며, 결제가 반환할 때 예약 환자의 정보가 포함되기 때문에, payment 와 PatientInformation 은 association 을 가져야 한다. 이의 근거로는 domain model 을 들 수 있다.
Register-Payment	Register 는 진료비 결제를 처리하므로 결제에 해당하는 payment 와 register 는 association 을 가져야 한다. 이의 근거로는 domain model 을 들 수 있다.
Payment-PayAdapter	Payment 는 진료비 결제를 목적으로 하고 이를 하기 위해서는 결제 시스템에 서비스를 요청해야 한다. 요청하는데 중간다리 역할을 하는 PayAdapter 를 사용한다. 그러므로 Payment 와 PayAdapter 는 association 을 가져야 한다.
TreatmentRecord-Payment	Payment 에서 처리한 내용을 기록해야 한다. 따라서 TreatmentRecord class 를 만들어서 Payment 가 해당 형식에 맞게 내용을 넘겨주면서 기록할 수 있다. 그러므로 TreatmentRecord 와 Payment 사이의 association 이 필요하다.

- role 설계 근거

표 34 진료비 결제\_role 설계 근거

<b>TreatmentRecord - PatientInformation</b>	환자 정보에는 환자가 받은 진료의 기록이 포함된다. 그러므로 TreatmentRecord 에게 TreaatmentRecord 를 줄인 tr 의 role 을 주었다.
<b>PatientInformation - Payment</b>	진료비 결제가 처리될 때 환자 정보를 이용하기 때문에 Payment 에 해당 환자 정보가 반환되므로 PatientInformation 을 줄인 pi 의 role 을 주었다.
<b>Register-Payment</b>	Register 는 결제를 처리하며 Payment 는 환자의 진료비 결제를 수행하는 역할을 하므로 Payment p 의 role 을 주었다.
<b>Payment-PayAdapter</b>	Payment 는 진료비 결제를 목적으로 하고 결제를 하기 위해서는 결제 시스템에 서비스를 요청해야 한다. 요청하는데 중간 다리 역할을 하는 PayAdapter 를 사용하므로, PayAdapter 를 줄인 pa 의 role 을 주었다.
<b>TreatmentRecord-Payment</b>	Payment 가 처리한 내용을 TreatmentRecord 에 전달해주면서 기록해야 한다. 따라서 Payment class 의 instance 인 p 의 role 을 주었다.

- Class Diagram

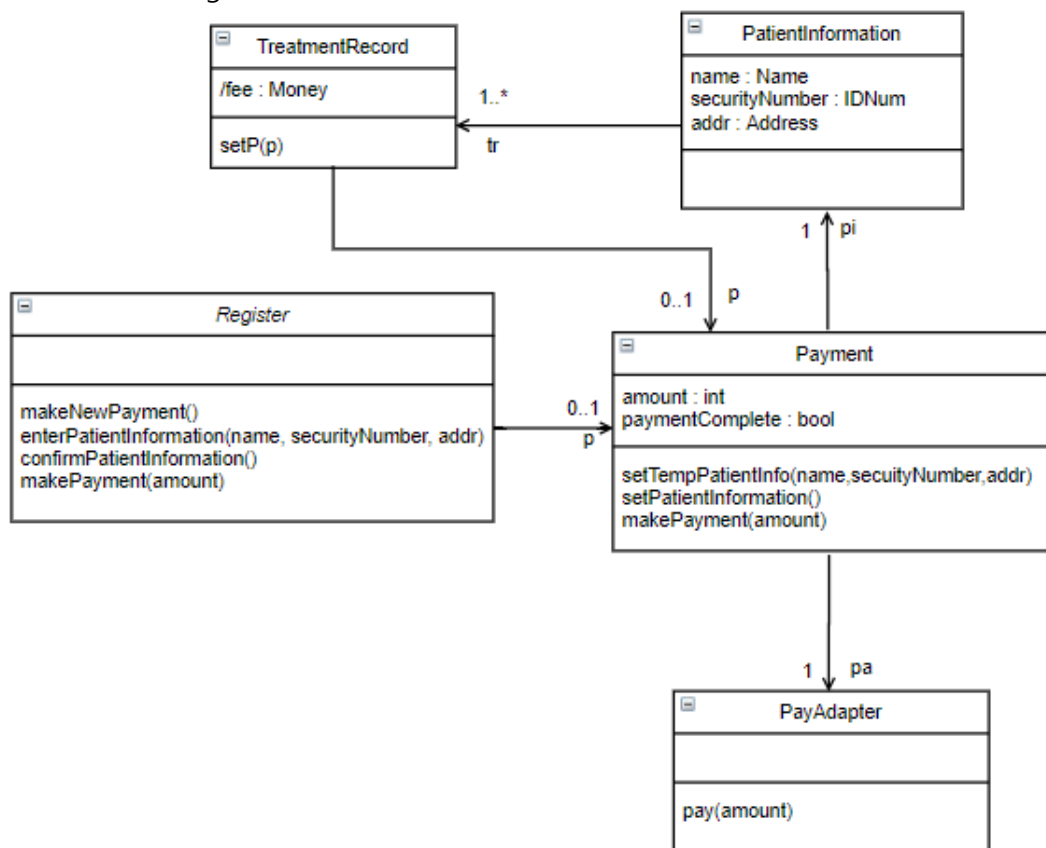


그림 13 진료비 결제\_Class Diagram

#### 다) Start Up Use Case

- class name, attribute, operation 설계 근거

**표 35 Start Up Use Case\_Hospital Class 설계 근거**

Hospital	
<b>name</b>	해당 class 는 register 및 manager class 를 생성하며, 이는 병원 전체를 나타낼 수 있는 이름으로 하는 것이 좋다. 따라서 병원을 나타내는 Hospital 을 이름으로 선택했다.
<b>attributes</b>	X
<b>operation</b>	X

**표 36 Start Up Use Case \_Register Class 설계 근거**

Register	
<b>name</b>	해당 class 는 input system event 를 처리하는 controller 로 전체 시스템을 설명할 수 있는 것의 이름으로 하는 것이 좋다. Register 는 병원에서 접수 및 결제를 처리하는 데스크라는 뜻이 있으므로, Register 을 이름으로 선택했다. 이는 Hospital class 에 의해서 만들어져야 한다.
<b>attributes</b>	X
<b>operation</b>	X

**표 37 Start Up Use Case\_Manager Class 설계 근거**

Manager	
<b>name</b>	해당 class 는 병원 관리자와 시스템 관리자의 정보를 contain 한다. 따라서 관리자를 나타내는 Manager 를 이름으로 선택했다.
<b>attributes</b>	X
<b>operation</b>	<p>read_h_manager() : 병원 관리자의 id 와 password 에 대한 정보를 불러와야 하므로, 이를 operation 으로 선정하였다.</p> <p>put_h_manager(HosM) : 병원 관리자의 id 와 password 에 대한 정보를 입력하여 Hospital_manager 객체를 만들어 arraylist 에 넣어야 하므로, 이를 operation 으로 선정하였다.</p> <p>read_s_manager() : 시스템 관리자의 id 와 password 에 대한 정보를 불러와야 하므로, 이를 operation 으로 선정하였다.</p> <p>put_s_manager(SysM) : 시스템 관리자의 id 와 password 에 대한 정보를 입력하여 System_manager 객체를 만들어서 arraylist 에 넣어야 하므로, 이를 Operation 으로 선정하였다.</p>

**표 38 Start Up Use Case\_Hospital manager Class 설계 근거**

Hospital Manager	
<b>name</b>	해당 class 는 병원 관리자의 ID 와 Password 를 저장하는데 사용된다. 따라서 병원 관리자를 나타내는 Hospital_manager 를 이름으로 선택했다.

<b>attributes</b>	id : 사용자 모드에 대한 병원 관리자의 접근 권한 정보에는 id 가 필수적이므로, id 를 Hospital_manager 의 attribute 로 선정했다. password : 사용자 모드에 대한 병원 관리자의 접근 권한 정보에는 password 가 필수적이므로, password 를 Hospital_manager 의 attribute 로 선정했다.
<b>operation</b>	X

**표 39 Start Up Use Case\_System\_manager Class 설계 근거**

<b>System_manager</b>	
<b>name</b>	해당 class 는 시스템 관리자의 ID 와 Password 를 저장하는 데 사용된다. 따라서 시스템 관리자를 나타내는 System_manager 를 이름으로 정했다.
<b>attributes</b>	id : 사용자 모드에 대한 시스템 관리자의 접근 권한 정보에는 id 가 필수적이므로, id 를 System_manager 의 attribute 로 선정했다. password : 사용자 모드에 대한 시스템 관리자의 접근 권한 정보에는 password 가 필수적이므로, password 를 System_manager 의 attribute 로 선정했다.
<b>operation</b>	X

- relation 설계 근거

**표 40 Start Up Use Case\_relation 설계 근거**

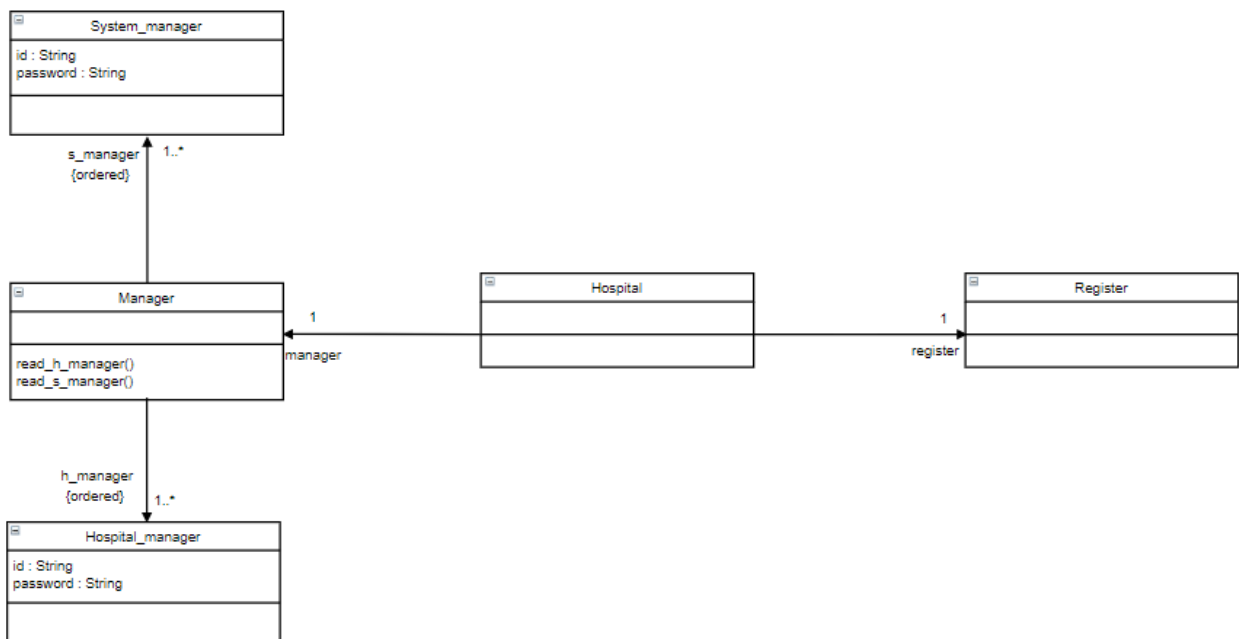
<b>Hospital-Register</b>	사용자에게 입력을 받으면 처리하기 위하여 Register class 필요하기 때문에, Hospital class 가 Register class 를 생성할 필요가 있으므로 Hospital 과 Register 는 association 을 가져야 한다.
<b>Hospital-Manager</b>	진료 관리 시스템에 등록될 관리자 객체를 만들어야 하기 때문에, Hospital class 가 Manager class 를 생성할 필요가 있으므로 Hospital 과 Manager 는 association 을 가져야 한다.
<b>Manager-Hospital_manager</b>	진료 관리 시스템에 등록될 병원 관리자 객체를 만들어야 하기 때문에 Hospital class 가 Hospital_manager class 를 생성할 필요가 있으므로 Manager 과 Hospital_manager 는 association 을 가져야 한다.
<b>Manager-System_manager</b>	진료 관리 시스템에 등록될 시스템 관리자 객체를 만들어야 하기 때문에 Hospital class 가 System_manager class 를 생성할 필요가 있으므로 Manager 와 System_manager 는 association 을 가져야 한다.

- role 설계 근거

**표 41 Start Up Use Case\_role 설계 근거**

<b>Hospital-Register</b>	Hospital 에는 system 초기화 시 Register 가 생성될 필요가 있으므로 Register 를 줄인 rg 의 role 을 주었다.
<b>Hospital-Manager</b>	Hospital 에는 system 초기화 시 Manager 가 생성될 필요가 있으므로 Manager 를 줄인 M 의 role 을 주었다.
<b>Manager-Hospital_manager</b>	Manager 에는 system 초기화 시 Hospital_manager 가 생성될 필요가 있으므로, Hospital_manager 를 줄인 h_manager 의 role 을 주었다.
<b>Manager-System_manager</b>	Manager 에는 system 초기화 시 System_manager 가 생성될 필요가 있으므로 System_manager 를 줄인 s_manager 의 role 을 주었다.

- Class Diagram



**그림 14 Start Up Use Case\_Class Diagram**

3. 수정/보완된 Source Code

3.1 Revision history

Revision History 3\_수정/보완된 Source Code

버전	일자	설명	저자
Elaboration phase 4-1	20.07.10	3.2 수정/보완된 Source Code 작성 근거 작성 3.3 수정/보완된 Source Code 작성	아무거나하조
Elaboration phase 4-2	20.07.11	3.2 수정/보완된 Source Code 작성 근거 보완 3.3 수정/보완된 Source Code 보완	아무거나하조

3.2 수정/보완된 Source Code 작성 근거

가) 진료비 결제

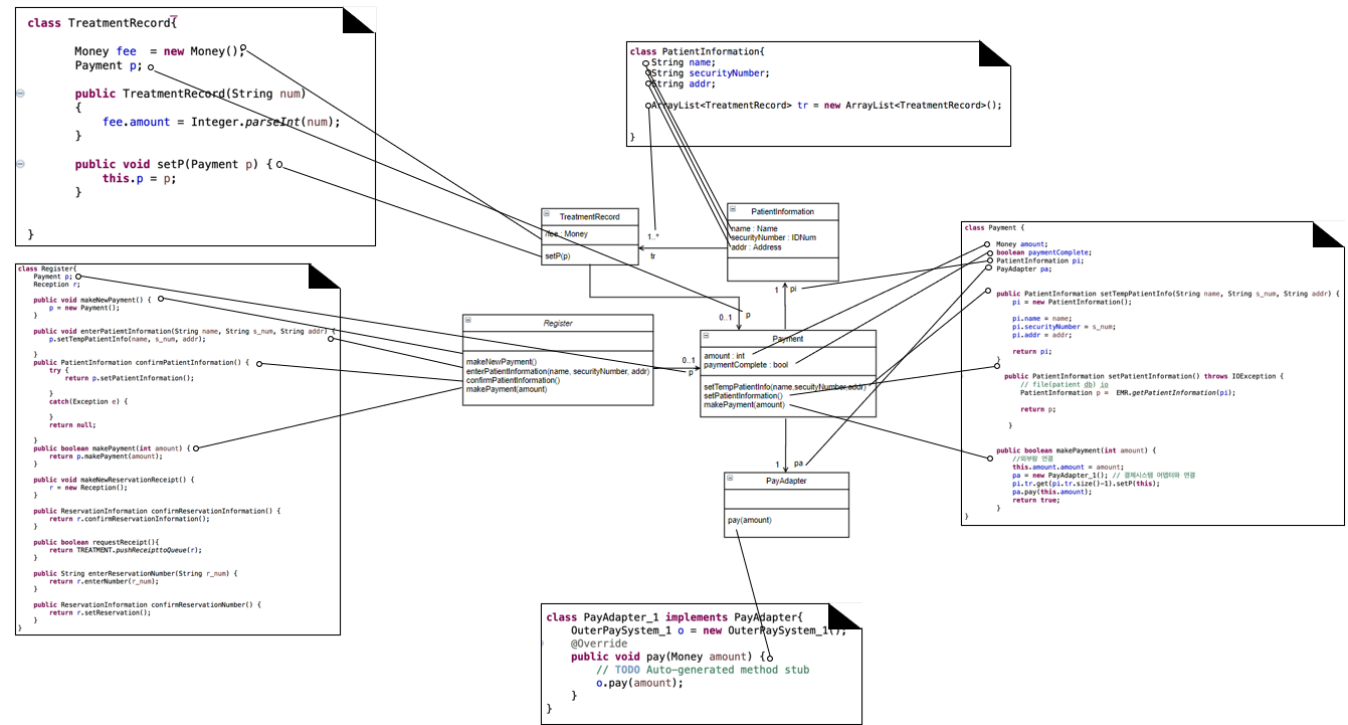


그림 15 Source Code-Class Diagram mapping\_진료비 결제

## 나) Start up Use Case

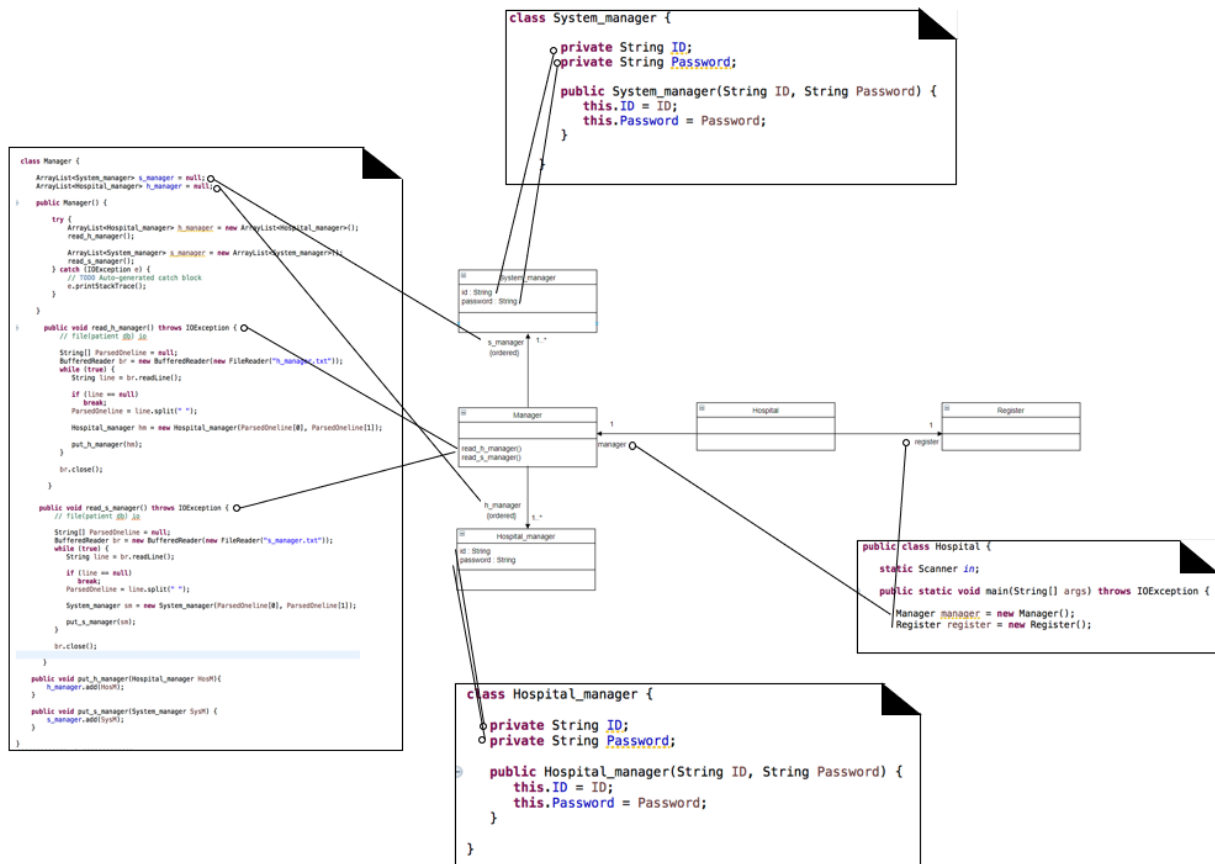


그림 16 Source Code-Class Diagram mapping\_start up use case

### 3.3 수정/보완된 Source Code

가) Register

```
class Register{
    Payment p;
    Reception r;

    public void makeNewPayment() {
        p = new Payment();
    }

    public void enterPatientInformation(String name, String s_num, String addr) {
        p.setTempPatientInfo(name, s_num, addr);
    }

    public PatientInformation confirmPatientInformation() {
        try {
            return p.setPatientInformation();
        }
        catch(Exception e) {
        }
        return null;
    }

    public boolean makePayment(int amount) {
        return p.makePayment(amount);
    }

    public void makeNewReservationReceipt() {
        r = new Reception();
    }

    public ReservationInformation confirmReservationInformation() {
        return r.confirmReservationInformation();
    }

    public boolean requestReceipt(){
        return TREATMENT.pushReceipttoQueue(r);
    }

    public String enterReservationNumber(String r_num) {
        return r.enterNumber(r_num);
    }

    public ReservationInformation confirmReservationNumber() {
        return r.setReservation();
    }
}
```

그림 17 SourceCode\_Register



나) ReservationInformation

```
class ReservationInformation{
    DoctorInformation di;
    PatientInformation pi;
    String reservationNumber;
    Date date;
    LocalTime time;
}
```

그림 18 SourceCode\_ReservationInformation

다) Reception

```
class Reception{
    ReservationInformation ri;
    boolean isComplete ;

    public String enterNumber(String rNum) {

        ri.reservationNumber = rNum;

        return ri.reservationNumber;
    }

    public ReservationInformation setReservation() {
        //외부에서 정보 가져오기
        try {
            ReservationSystem.getReservation(ri.reservationNumber);
        }
        catch(Exception e) {
            System.out.println(e.getMessage());
            return null;
        }

        return ri;
    }

    public ReservationInformation confirmReservationInformation() {
        return ri;
    }
}
```

그림 19 SourceCode\_Reception

라) PatientInformation

```
class PatientInformation{
    String name;
    String securityNumber;
    String addr;

    ArrayList<TreatmentRecord> tr = new ArrayList<TreatmentRecord>();

}
```

그림 20 SourceCode\_PatientInformation

마) DoctorInformation

```
class DoctorInformation{
    ArrayList<Date> workingDate = new ArrayList<Date>();
    String name;

}
```

그림 21 SourceCode\_DoctorInformation

바) TreatmentRecord

```
class TreatmentRecord{

    Money fee = new Money();
    Payment p;

    public TreatmentRecord(String num)
    {
        fee.amount = Integer.parseInt(num);
    }

    public void setP(Payment p) {
        this.p = p;
    }

}
```

그림 22 SourceCode TreatmentRecord

#### 사) Payment

```
class Payment {  
  
    Money amount;  
    boolean paymentComplete;  
    PatientInformation pi;  
    PayAdapter pa;  
  
    public PatientInformation setTempPatientInfo(String name, String s_num, String addr) {  
        pi = new PatientInformation();  
  
        pi.name = name;  
        pi.securityNumber = s_num;  
        pi.addr = addr;  
  
        return pi;  
    }  
  
    public PatientInformation setPatientInformation() throws IOException {  
        // file(patient db) io  
        PatientInformation p = EMR.getPatientInformation(pi);  
  
        return p;  
    }  
  
    public boolean makePayment(int amount) {  
        //외부랑 연결  
        this.amount.amount = amount;  
        pa = new PayAdapter_1(); // 결제시스템 어댑터와 연결  
        pi.tr.get(pi.tr.size()-1).setP(this);  
        pa.pay(this.amount);  
        return true;  
    }  
}
```

그림 23 SourceCode\_Payment

#### 아) Money

```
class Money{  
    int amount;  
    String unit;  
}
```

그림 24 SourceCode\_Money

자) payAdapter

```
interface PayAdapter{  
    public void pay(Money amount);  
}
```

그림 25 SourceCode\_InterfacePayAdapter

```
class PayAdapter_1 implements PayAdapter{  
    OuterPaySystem_1 o = new OuterPaySystem_1();  
    @Override  
    public void pay(Money amount) {  
        // TODO Auto-generated method stub  
        o.pay(amount);  
    }  
}
```

그림 26 SourceCode\_ClassPayAdapter

차) 외부 결제

```
class OuterPaySystem_1 {  
    public void pay(Money amount) {  
        // TODO Auto-generated method stub  
        try {  
            System.out.println("카드를 넣어 주세요. \n");  
            TimeUnit.SECONDS.sleep(1);  
  
            for (int i = 0; i < 3; i++) {  
                System.out.println("Processing...");  
                TimeUnit.SECONDS.sleep(1);  
            }  
            System.out.println("카드 결제 성공. \n");  
        } catch (Exception e) {  
            System.out.println(e.getMessage());  
            //return false;  
        }  
    }  
}
```

그림 27 SourceCode\_외부 결제

ㄱ) System\_manager

```
class System_manager {  
  
    private String ID;  
    private String Password;  
  
    public System_manager(String ID, String Password) {  
        this.ID = ID;  
        this.Password = Password;  
    }  
  
}
```

그림 28 SourceCode\_System\_manager

ㄴ) Hospital\_manager

```
class Hospital_manager {  
  
    private String ID;  
    private String Password;  
  
    public Hospital_manager(String ID, String Password) {  
        this.ID = ID;  
        this.Password = Password;  
    }  
  
}
```

그림 29 SourceCode\_Hospital\_manager

#### 파) manager

```
class Manager {

    ArrayList<System_manager> s_manager = null;
    ArrayList<Hospital_manager> h_manager = null;

    public Manager() {

        try {
            ArrayList<Hospital_manager> h_manager = new ArrayList<Hospital_manager>();
            read_h_manager();

            ArrayList<System_manager> s_manager = new ArrayList<System_manager>();
            read_s_manager();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            System.out.println("파일이 존재하지 않는다. ");
        }

        public void read_h_manager() throws IOException {
            String[] ParsedOnewline = null;
            BufferedReader br = new BufferedReader(new FileReader("h_manager.txt"));

            while (true) {
                String line = br.readLine();
                if (line == null)
                    break;
                ParsedOnewline = line.split(" ");
                Hospital_manager hm = new Hospital_manager(ParsedOnewline[0], ParsedOnewline[1]);
                put_h_manager(hm);
            }
            br.close();
        }

        public void read_s_manager() throws IOException {
            String[] ParsedOnewline = null;
            BufferedReader br = new BufferedReader(new FileReader("s_manager.txt"));

            while (true) {
                String line = br.readLine();
                if (line == null)
                    break;
                ParsedOnewline = line.split(" ");
                System_manager sm = new System_manager(ParsedOnewline[0], ParsedOnewline[1]);
                put_s_manager(sm);
            }
            br.close();
        }

        public void put_h_manager(Hospital_manager HosM){
            h_manager.add(HosM);
        }

        public void put_s_manager(System_manager SysM) {
            s_manager.add(SysM);
        }
    }
}
```

그림 30 SourceCode\_manager

#### 하) Hospital

```
public class Hospital {
    static Scanner in;
    public static void main(String[] args) throws IOException {

        Manager manager = new Manager();
        Register register = new Register();

        while (true) {
            String opt, name, addr, security_number;
            Person temp_person = null;

            System.out.println("Please select the menu.");
            System.out.println("-----");
            System.out.println("1. 예약 환자 접수");
            System.out.println("2. 진료비 결제");

            int menu = in.nextInt();
            in.nextLine();

            if (menu == 1) { // 예약환자접수(reservation);

                register.makeNewReservationReceipt();

                String reservation_number;
                ReservationInformation rii;

                do {
                    System.out.println("예약 번호 입력 : ");
                    reservation_number = in.nextLine();
                    reservation_number = register.enterReservationNumber(reservation_number);

                    System.out.println("\n" + reservation_number + "\n" + "정보가 맞습니까?(y,n) : ");

                    opt = in.nextLine();
                } while (opt.equals("n"));

                rii = register.confirmReservationNumber();

                if (rii != null) {
                    System.out.println("이름 : " + rii.pi.name);
                    System.out.println("주민번호 : " + rii.pi.securityNumber);
                    System.out.println("담당 의사 : " + rii.di.name);
                    System.out.println("해당 예약 정보가 맞습니까?(y,n) : ");
                    opt = in.nextLine();

                    if (opt.equals("y")) {
                        register.confirmReservationInformation();
                        if (register.requestReceipt())
                            System.out.println("접수 완료\n");
                        else
                            System.out.println("접수 실패 \n");
                    } else {
                        System.out.println("관리자에게 문의 하세요 \n");
                    }
                } else {
                    System.out.println("해당 예약 정보가 없습니다. \n");
                }
            }
        }
    }
}
```

그림 31 SourceCode\_Hospital1

```

else if (menu == 2) {
    // 진료비 결제
    register.makeNewPayment();

    do {
        System.out.println("이름을 입력하세요 : ");
        name = in.nextLine();
        System.out.println("주민번호를 입력하세요 : ");
        security_number = in.nextLine();
        System.out.println("주소를 입력하세요 : ");
        addr = in.nextLine();
        register.enterPatientInformation(name, security_number, "");
        System.out.println("\n" + name + " " + security_number + "\n" + addr + " "+"정보가 맞습니까?(y,n) : ");
        opt = in.nextLine();

        if (opt.equals("y")) {
            PatientInformation tp = register.confirmPatientInformation();
            if (tp != null)
            {
                Iterator<TreatmentRecord> it = tp.tr.iterator();
                while(it.hasNext()) {

                    System.out.println(it.next());
                    System.out.println("액수를 입력하세요 : ");
                    int amount = in.nextInt();
                    in.nextLine();

                    if(amount != 0) {
                        if(register.makePayment(amount)) {
                            System.out.println("결제완료 ");
                        }
                        else {
                            System.out.println("실패 ");
                        }
                    }
                    else
                    {
                        System.out.println("결제할 진료비가 없습니다 ");
                    }
                }
                break;
            }
            else {
                System.out.println("등록된 정보가 없습니다. \n");
                opt = "n";
            }
        }
    } while (opt.equals("n"));
}

```

그림 32 SourceCode\_Hospital2



## 4. Test 보고서

### 4.1 Revision history

Revision History 4\_Test 보고서

버전	일자	설명	저자
Elaboration phase 4-1	20.07.10	4.2 Test 보고서 작성	아무거나하조
Elaboration phase 4-2	20.07.11	4.2 Test 보고서 보완	아무거나하조
Elaboration phase 4-3	20.07.12	4.2 Test 보고서 보완	아무거나하조

### 4.2 Test 보고서

아래와 같은 보고서를 통해, 수정 보완된 부분인 진료비 결제 use case 에서 외부 시스템과의 연결을 adapter 를 이용한 방안은 기존의 Source Code 구현에 영향을 미치지 않았음을 알 수 있다.

#### 가) 예약 환자 접수

표 42 Test 보고서\_예약 환자 접수

Test Case ID	1	Test Name	예약환자 접수		Use Case	예약 환자 접수
Description	예약 번호를 통해 진료를 접수하는 과정을 테스트한다.					
Precondition	<ul style="list-style-type: none"><li>- 시스템이 정상 작동하고 있어야 한다.</li><li>- 시스템이 환자 정보 시스템과 정상적으로 연결되어 있어야 한다.</li><li>- 환자는 예약 번호를 가지고 있다.</li></ul>					
Step	System Operation	Test Steps	Test Data	Expected Result	Actual Result	Status(P/F)
1	makeNew Reservation Receipt	원하는 transaction 을 선택한다.	1(예약 환자 접수)	사용자는 예약 환자 접수 transaction 을 시작한다.	예약 번호 입력 :	P
2	enterReservation Number(reservati o nNumber)	예약 번호를 입력한다.	s123	시스템은 사용자에게 입력한 예약번호가	s123 정보가 맞습니까? (y,n)	P

				맞는지 물어본다.		
2-1		잘못된 예약 번호를 입력한다.	123S	시스템은 에러를 출력한다.	해당 예약 정보가 없습니다.	P
3	confirmReservationNumber	입력한 예약 번호가 맞는지 확인한다.	Y + valid 예약 번호	시스템은 입력한 예약 번호에 해당하는 예약 정보를 사용자에게 보여준다.	이름 : 홍길동 주민번호 : 200710-1234567 담당의사 : 이정태 해당 예약 정보가 맞습니까?(y/n) :	P
			Y + invalid 예약 번호	시스템은 사용자에게 입력한 예약 번호에 해당하는 예약이 없음을 알린다.	해당 예약 정보가 없습니다.	P
			N	사용자에게 다시 예약 번호 입력을 요청한다.	예약 번호 입력 :	P
4	confirmReservationInformation	예약 정보가 맞는지 확인한다.	Y + 진료 시스템 정상 작동	시스템은 사용자에게 접수가 완료되었음을 알려준다.	접수 완료	P
			Y+진료 시스템 오작동	시스템은 사용자에게 접수가 실패하였음을 알려준다.	접수 실패	P

			N	시스템은 사용자에게, 관리자에게 문의하라고 알려준다.	관리자에게 문의하세요	P
<b>Postcondition</b>		<ul style="list-style-type: none"> <li>- 중간에 접수가 중단되지 않았다면, 접수 정보가 정상적으로 원하는 의사의 대기줄에 등록된다.</li> </ul>				

나) 진료비 결제

표 43 Test 보고서\_진료비 결제

Test Case ID	2	Test Name	진료비 결제		Use Case	진료비 결제	
Description	개인 정보를 통해 지불할 진료비를 알아내고 해당 진료비를 결제하는 과정을 테스트한다.						
Precondition	<div>- 시스템이 정상 작동하고 있어야 한다.</div> <div>- 시스템이 환자 정보 시스템과 정상적으로 연결되어 있어야 한다.</div> <div>- 시스템이 결제 시스템과 정상적으로 연결되어 있어야 한다.</div> <div>- 환자는 이름, 주민등록번호, 주소를 가지고 있어야 한다.</div>						
Step	System Operation	Test Steps	Test Data	Expected Result	Actual Result	Status(P/F)	
1	makeNewPayment	원하는 transaction 을 선택한다.	2(진료비 결제)	사용자는 진료비 결제 transaction 을 시작한다.	이름을 입력하세요 :	P	
2	enterPatient Information	이름, 주민등록번호, 주소를 입력한다.	홍길동, 200710 - 1234567, 경기도 수원시 영통구 원천동 월드컵로 206	시스템은 사용자에게 입력한 이름, 주민등록번호, 주소를 보여주고 이 정보가 맞는지 물어본다.	홍길동 200710-1234567 경기도 수원시 영통구 원천동 월드컵로 206 정보가 맞습니까? (y,n) :	P	
3	confirmPatientInformation	입력한 이름, 주민등록번호, 주소가 맞는지 확인한다.	Y + valid 이름, valid 주민등록번호, valid 주소 + 결제되지 않은 진료비 O	시스템은 입력한 이름, 주민등록번호, 주소에 해당하는 환자의 진료비를 사용자에게 보여준다.	10000 50000 30000	P	

			Y + valid 이름, valid 주민등록번호, valid 주소 + 결제되지 않은 진료비 X	시스템은 입력한 이름, 주민등록번호, 주소에 해당하는 환자의 진료비를 사용자에게 보여준다. 결제할 진료비가 없으므로 결제할 진료비가 없음을 사용자에게 알린다.	결제할 진료비가 없습니다.	P
			Y + invalid 이름, 주민등록번호, 주소	시스템은 사용자에게 입력한 이름, 주민등록번호, 주소에 해당하는 환자가 없음을 알린다.	등록된 정보가 없습니다.	P
			N	사용자에게 다시 이름, 주민등록번호, 주소 입력을 요청한다.	이름을 입력하세요.	P
4	makePayment(amount)	진료비를 지불한다.	90000	시스템은 사용자에게 결제 시스템에게 결제가 정상 처리 됐는지 알린다. 해당 진료에 대한 결제가 일어났음을 저장한다.	결제 완료	P
5		1,2,3 과정을 반복한다.	홍길동, 200710 - 1234567, 경기도 수원시 영통구 원천동 월드컵로 206 Y	사용자가 결제를 완료하였으므로 시스템은 사용자에게 결제할 진료비가 없음을 알린다.	결제할 진료비가 없습니다.	P
Postcondition		- 중간에 결제가 중단되지 않았다면, 진료에 대한 진료비 결제가 완료되었음을 저장한다.				

다) Start up Use Case

표 44 Test 보고서\_Start up Use case

Test Case ID		3	Test Name	StartUp		Use Case	StartUp
Description		시스템을 시작할 때 초기화하는 과정을 테스트한다.					
Precondition		<div>- 시스템이 정상 작동하고 있어야 한다.</div> <div>- 시스템은 초기화 단계여야 한다.</div>					
Step	System Operation	Test Steps	Test Data	Expected Result	Actual Result	Status (P/F)	
1	initiateSystem	manager 를 생성한다.		manager 객체가 생성된다.	manager	P	
		register 를 생성한다.		register 객체가 생성된다.	register	P	
		병원 관리자 목록을 생성한다.		ArrayList<Hospital_manager>가 생성된다.	h_manager	P	
		병원 관리자의 정보를 불러온다.	hongkildong1, pAsSwOrd11	병원 관리자의 id, password 정보 하나를 읽어온다.	honkildong1, pAsSwOrd11	P	
			hongkildong2, pAsSwOrd22				
			hongkildong3, pAsSwOrd33				
		해당 파일이 존재하지 않음	해당 파일이 존재하지 않아 에러가 발생한다.	파일이 존재하지 않는다.	P		
		병원 관리자 목록에 병원 관리자의 정보를 넣는다.		병원 관리자 목록에 id, password 정보가 담긴다.		P	
		시스템 관리자의 목록을 생성한다.		ArrayList<System_manager>가 생성된다.	s_manager	P	
시스템 관리자의	admin1, adminpW111 admin2,	시스템 관리자의 id, password 정보 하나를 읽어온다.	admin1, adminpW111	P			

		정보를 불러온다.	adminpW222			
			해당 파일이 존재하지 않음	해당 파일이 존재하지 않아 에러가 발생한다.	파일이 존재하지 않는다.	p
		시스템 관리자 목록에 시스템 관리자의 정보를 넣는다.		시스템 관리자 목록에 id, password 정보가 담긴다.		P
2	makeNewPayment	원하는 transaction 을 선택한다.	2(진료비 결제)	사용자는 진료비 결제 transaction 을 시작한다.	이름을 입력하세요 :	P
<b>Postcondition</b>		- 병원 관리자와 시스템 관리자의 목록이 정상적으로 생성되고 초기화된다.				