Report

도메인 분석 및 SW 설계

- Elaboration phase 3-

교수님 : 이정태 교수님

제출 일 : 2020.07.17

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
Elaboration phase 4-4	20.07.16	품질 개선을 위한 Design Pattern 적용 보완 수정/보완된 Use case Realization 보완 수정/보완된 Source Code 보완	아무거나하조

<순서>

<표 순서>

<그림 순서>

<Revision History 순서>

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
Elaboration phase 4-2	20.07.16	1.2 품질 개선을 위한 Design Pattern 적용 기능 및 이유 보완	아무거나하조

1.2 품질 개선을 위한 Design Pattern 적용 기능 및 이유

- **[Adapter]Design Pattern 을 통하여 시스템 품질 개선이 가능하다고 판단한 부분 1**

진료비 결제 Use Case 에서 외부 시스템과 직접 연결하는 부분을 중간 접속자인 Adapter 를 이용하여 시스템 품질 개선이 가능하다.

- **Adapter 적용 이유**

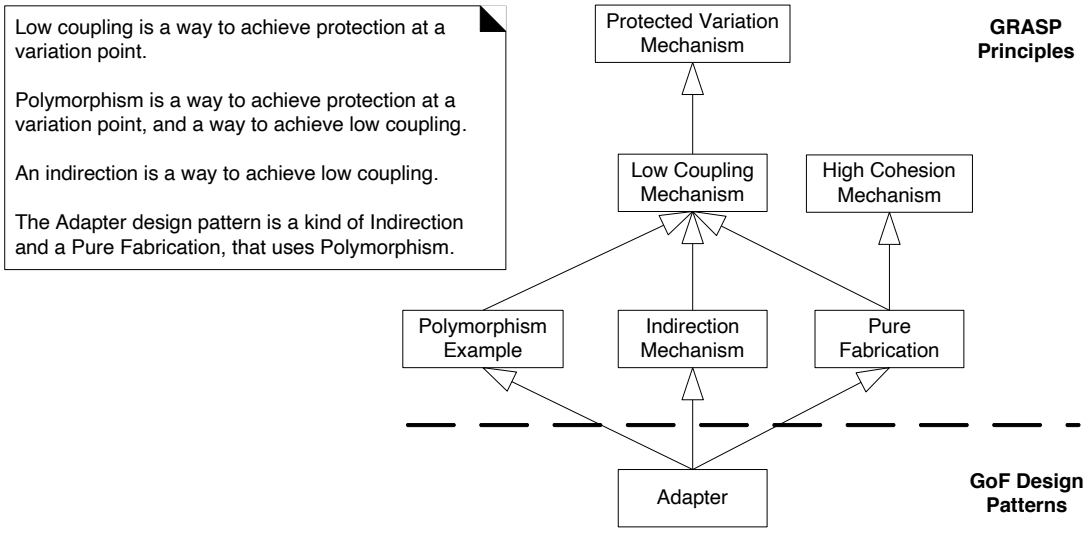


그림 1 Adapter 적용 이유

표 1 Adapter_GRASP 적용 이유

More GRASP	적용 이유
Polymorphism	nonfunctional requirement 중 reliability 1 번인 "외부 서비스(전자서명 장치, 처방전 출력 장치, 네트워크)에 오류가 발생하더라도 계속해서 접수/결제를 진행할 수 있도록 지원해야 한다."에

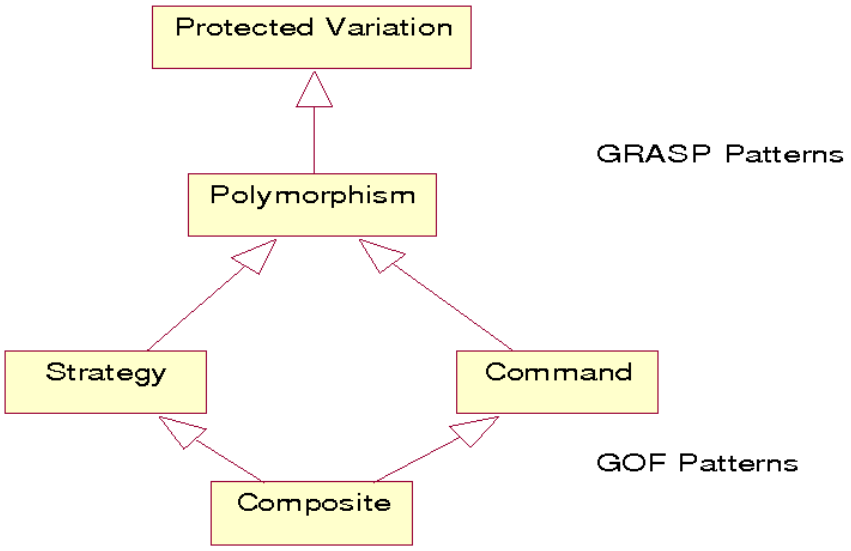
	의하여, 외부 결제 시스템 A 와의 연결이 끊어지면 바로 B 결제 시스템과의 연결을 허용해야 한다. 외부 시스템이 바뀜에 따라, 외부 시스템과 연결되는 부분이 다 바뀌게 된다면, 이는 적응성이 굉장히 떨어진다. 따라서 Polymorphism pattern 을 적용하여 중간 접속자인 Adapter 를 이용한다면 변화에 대한 적응성을 높일 수 있다.
Pure Fabrication	Payment class 는 System event 에 대한 입력을 받아오는 Register, Patient Information, 외부 결제시스템과의 연결이 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.
Indirection	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.
Protected Variation	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.

- **[Strategy]Design Pattern 을 통하여 시스템 품질 개선이 가능하다고 판단한 부분 2**

진료비 결제 Use Case 에서 진료비를 책정하는 부분에 진료비 결제 시 다양한 진료비 할인 정책을 적용할 수 있도록 하기 위해서 Strategy 패턴을 적용해서 각각의 정책을 공통 인터페이스를 가진 독립된 클래스에 정의할 수 있다. 진료비 결제 Use Case 에서는 이와 같은 Strategy 를 이용하여 시스템 품질 개선이 가능하다.

- **Strategy 적용 이유**



-

그림 2 Strategy 적용 이유

표 2 Strategy_GRASP 적용 이유

More GRASP	적용 이유
Polymorphism	nonfunctional requirement 중 supportability 5 번인 "변화하는 business rule 을 적용하기 위해 소프트웨어 업데이트를 할 수 있도록 지원해야 한다." 에 의하여 진료비를 책정하는 정책의 변화에 맞게 시스템을 수정해야 할 수 있다. 따라서 진료비 정책이 바뀌어서 적용되어야 할 때마다 매번 시스템을 수정해야 하는 경우가 발생할 수 있다. Strategy 패턴을 적용한다면, 변화 가능성이 있는 서로 관련된 정책을 설계하고, 정책을 변경하는 데 유연하게 대처할 수 있다.
Protected Variation	Strategy 패턴은 알고리즘의 변경에 대한 Protected Variations 를 제공한다. 진료비를 책정하는 정책이 바뀌더라도 Strategy 는 Polymorphism 패턴을 기반으로 설계되어 있기 때문에 정책 변경에 대한 영향이 적다.

따라서, 외부 시스템인 결제 시스템과 연결하기 위하여 중간 접속자인 PayStrategy 를 이용하면, 위와 같이 GRASP 의 Polymorphism, 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.

- **[Factory] Design Pattern 을 통하여 시스템 품질 개선이 가능하다고 판단한 부분 3**
  진료비 결제 Use Case 에서 외부 시스템과 연결을 제공하는 중간 접속자인 Adapter 및 다양하게 진료비 정책을 결정할 수 있는 Strategy 를 생성해야 한다. 하지만 adapter 와 strategy class 는 복잡하므로 생성하기 어렵다. 따라서 adapter 와 strategy 객체를 생성하는 것을 목적으로 하는 Factory 를 이용하여 시스템 품질 개선이 가능하다.

- **Factory 적용 이유**

표 3 Factory_GRASP 적용 이유

More GRASP	적용 이유
High Cohesion, Low Coupling	Register 는 Adapter 와 Strategy 를 생성할 responsibility 가 있다. 하지만 이는 high coupling 이고 separation of concern 을 위반하므로 바람직하지 않다. 따라서 adapter 와 strategy 를 생성하는 목적을 가진 Factory 라는 객체를 사용하여 high cohesion, low coupling, separation of concern 을 만족할 수 있다.
Pure Fabrication	외부 시스템과의 연결을 제공하는 중간 접속자인 Adapter 및 Strategy 를 생성할 때, Register 와 같은 도메인 객체가 이들을 생성하도록 선택하는 것은 separation of concerns 원칙의 목표에 어긋나며, 그 도메인 객체의 cohesion 을 감소시킬 수 있다. 따라서 현실세계에는 없지만 SW 상에서 존재하는 Factory 라는 객체를 이용하여 Adapter 와 Strategy 객체 생성을 한다면 재사용성을 높일 수 있다.

따라서, 외부 시스템의 중간 접속자인 adapter 및 strategy 를 생성할 때, Factory 를 이용한다면, 위와 같이 GRASP 의 Pure Fabrication pattern 과 high cohesion 및 low coupling 을 적용할 수 있으므로, 변화에 대한 적응성을 높일 수 있고, 유지 보수가 편리해지며, 재사용성이 증가하므로, 무인 접수 및 결제 시스템의 품질 개선이 가능하다.

ServicesFactory, PayAdapter, PayStrategy 는 시스템 당 1 개의 객체만 필요하므로 **singleton pattern** 을 적용한다.

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 2.3 수정/ 보완된 Design Class diagram 작성	아무거나하조
Elaboration phase 4-2	20.07.11	2.2 수정/보완된 Design Interaction diagram 보완 2.3 수정/ 보완된 Design Class diagram 보완	아무거나하조
Elaboration phase 4-3	20.07.16	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 4 예약 환자 접수_RDD_makeNewReservationReceipt

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

- GRASP pattern

표 5 예약 환자 접수_GRASP_makeNewReservationReceipt

0	Who will be responsible for handling the "makeNewReservationReceipt" system event? - Register 가 전체 시스템을 represent 하므로 **controller pattern** 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
1	Who will be responsible for creating Reception?

	- Register 가 Reception 을 initialize 하기 위한 데이터를 모두 가지고 있을 예정이기 때문에 **creator pattern** 을 적용하여 register 에게 reception 을 생성할 responsibility 를 준다.

- Sequence Diagram

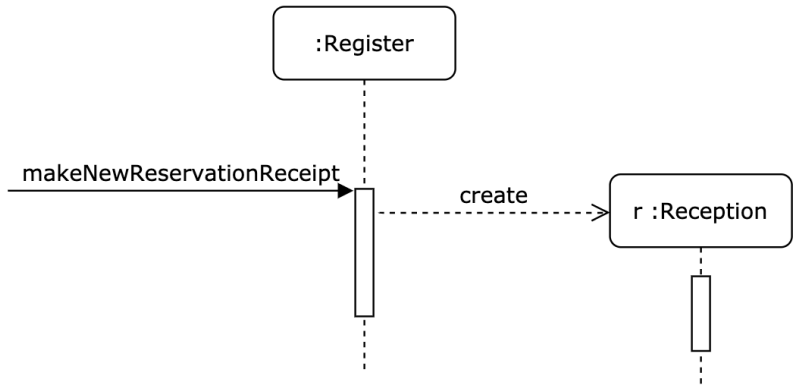


그림 3. 예약환자접수_ Sequence Diagram_makeNewReservationReceipt

2) enterReservationNumber(reservationNumber)

- RDD
  - 입력한 reservation number 를 보여준다.

표 6 예약 환자 접수_RDD_enterReservationNumber(reservationNumber)

Doing responsibilities	0. Register 는 enterReservationNumber system event 를 받을 responsibility 가 있다.
Knowing responsibilities	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 7 예약 환자 접수_GRASP_enterReservationNumber(reservationNumber)

0	Who will be responsible for handling the "enterReservationNumber" system event? - Register 가 전체 시스템을 represent 하므로 **controller pattern** 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
1	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 **knowledge expert pattern** 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 responsibility 를 부여하면 이는 low cohesion, high coupling 이므로 register 에 부여하는 것은 올바르지 않다. --> Reception 가 **high cohesion pattern**, **low coupling pattern** 을 적용하여 Reception 에게 reservation number 를 알 responsibility 를 준다.

- Sequence Diagram

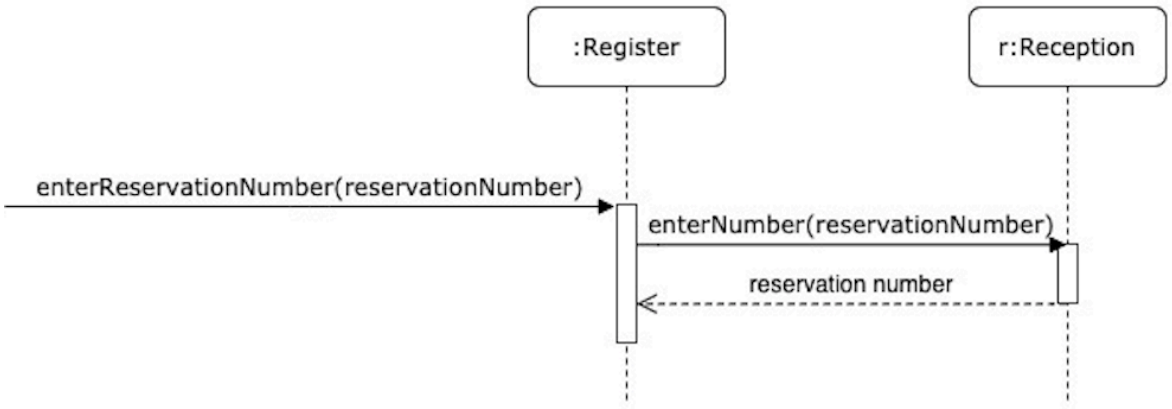


그림 4. 예약환자접수_ Sequence Diagram_enterReservationNumber(reservationNumber)

3) confirmReservationNumber

- RDD
  - reservation number 에 해당하는 reservation 정보를 가져온다.

표 8 예약 환자 접수_RDD_confirmReservationNumber

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

- GRASP pattern

표 9 예약 환자 접수_GRASP_confirmReservationNumber

0	Who will be responsible for handling the "confirmReservationNumber" system event? - Register 가 전체 시스템을 represent 하므로 **controller pattern** 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.

1	Who will be responsible for knowing reservation Information? - Reception 은 접수를 목적으로 하고 접수를 하기 위해 reservation information 이 필요하다. 그러므로 **information expert pattern**, **high cohesion pattern**, **low coupling pattern** 을 적용하여 Reception 에게 reservation information 을 알 responsibility 를 준다.
2	Who will be responsible for getting reservation information? - 예약 시스템은 reservation information 을 알아내는 일을 처리하는 데에 전문적인 정보를 가장 많이 가지고 있으므로 **information expert pattern** 을 적용하여 예약 시스템에게 reservation information 을 알아내는 responsibility 를 준다.

- Sequence Diagram

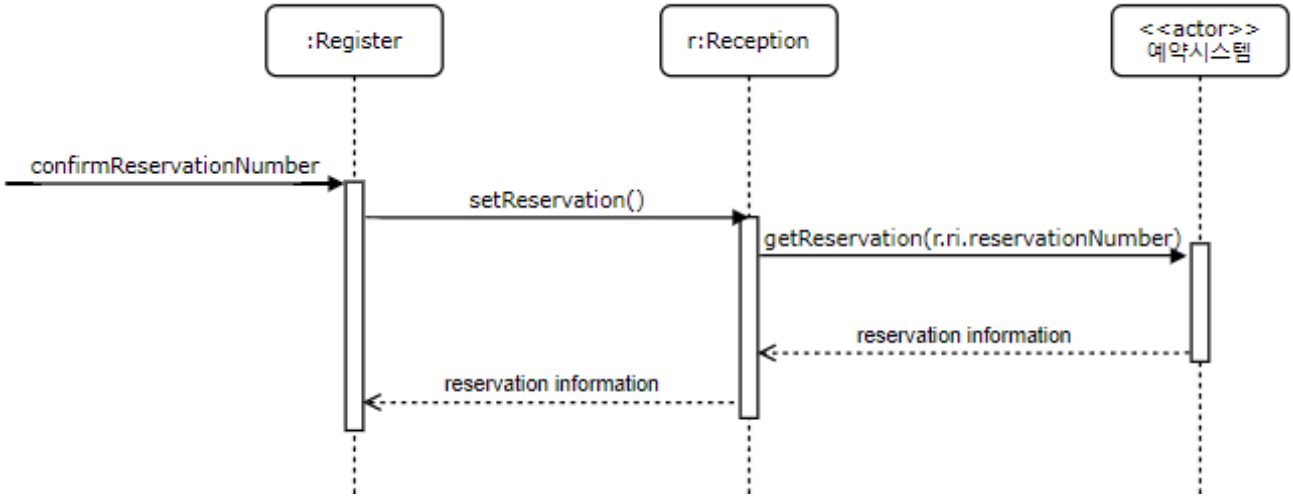


그림 5. 예약환자접수_ Sequence Diagram_confirmReservationNumber

4) confirmReservationInformation

- RDD

표 10 예약 환자 접수_RDD_confirmReservationInformation

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

- GRASP pattern

표 11 예약 환자 접수_GRASP_confirmReservationInformation

0	Who will be responsible for handling the "confirmReservationInformation" system event?

	- Register 가 전체 시스템을 represent 하므로 **controller pattern** 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
1	Who will be responsible for knowing that the Reservation Information is correct? - Reception 은 접수가 목적인 객체이므로 해당 정보를 가지고 있을 responsibility 가 있다**. High cohesion pattern** 과 **knowledge expert pattern** 을 적용하여 Reception 에게 reservation information 이 맞음을 알 responsibility 를 준다.

- Sequence Diagram

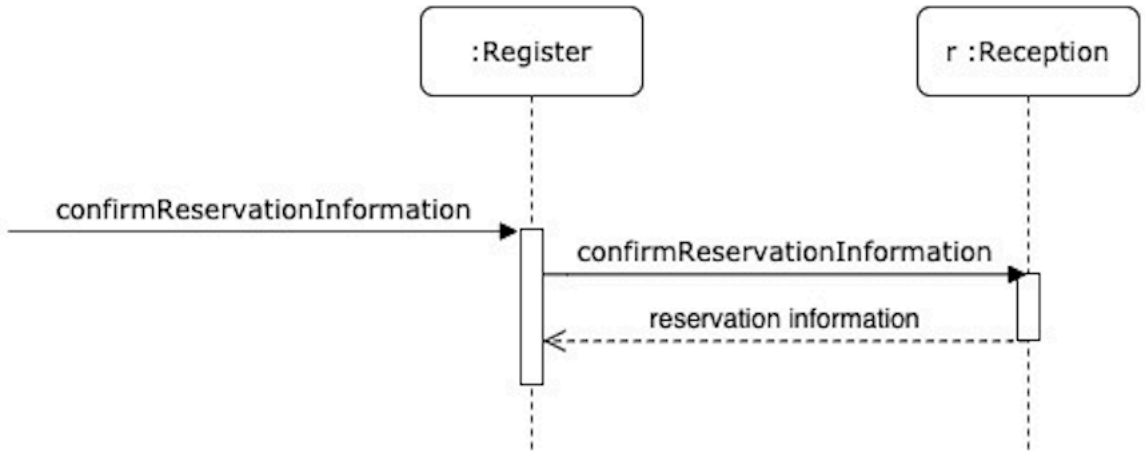


그림 6. 예약환자접수_ Sequence Diagram_confirmReservationInformation

5) requestReceipt

- RDD

표 12 예약 환자 접수_RDD_requestReceipt

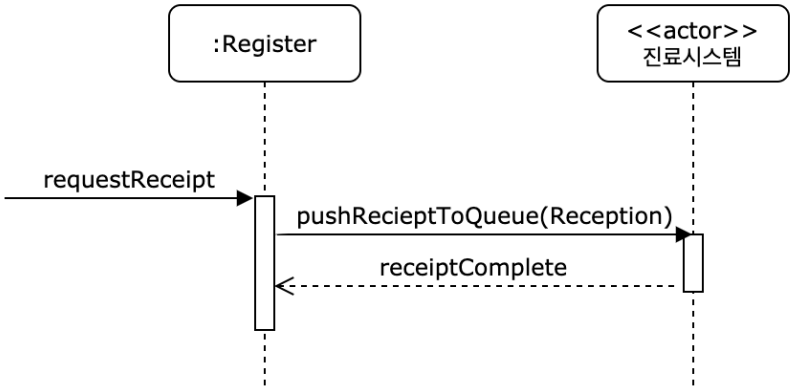
Doing responsibilities	0. Register 는 requestReceipt system event 를 받을 responsibility 가 있다.
Knowing responsibilities	1. Reception 은 접수를 처리할 responsibility 가 있다.

- GRASP pattern

표 13 예약 환자 접수_GRASP_requestReceipt

0	Who will be responsible for handling the "confirmReservationNumber" system event? - Register 가 전체 시스템을 represent 하므로 **controller pattern** 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
1	Who will be responsible for process reception? - Reception 은 접수가 목적인 객체이므로 접수에 관한 모든 정보를 가지고 있다**. High cohesion pattern** 과 **knowledge expert pattern** 을 적용하여 Reception 에게 접수를 처리할 responsibility 를 준다.

- Sequence Diagram



-

그림 7. Sequence Diagram_예약접수_requestReceipt

나) 진료비 결제

1) makeNewPayment

- RDD

표 14 진료비 결제_RDD_makeNewPayment

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

- GRASP pattern

표 15 진료비 결제_GRASP_makeNewPayment

0	Who will be responsible for handling the "makeNewPayment" system event? - Register 가 전체 시스템을 represent 하고 제일 처음 받아서 처리하기 때문에 **controller pattern** 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
1	Who will be responsible for creating Payment? - Register 가 Payment 를 initialize 하기 위한 데이터를 모두 가지고 있고 Payment 를 record 할 예정이기 때문에 **creator pattern** 을 적용하여 Register 에게 Payment 를 생성할 responsibility 를 준다.
2	Who will be responsible for getting payAdapter and payStrategy that is needed to create Payment? - ServicesFactory 는 payAdapter 와 payStrategy 를 생성하고 생성된 객체를 저장한다. 그러므로 **knowledge expert pattern** 을 적용하여

	ServicesFactory 에게 payAdapter 와 payStrategy 을 가져올 responsibility 를 준다.

- Sequence Diagram

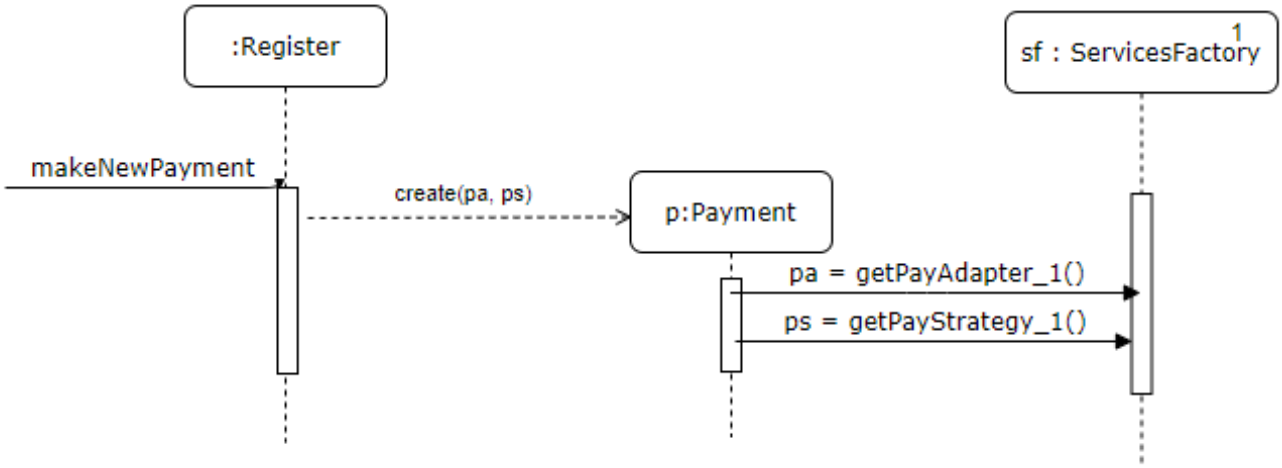


그림 8 진료비 결제_Sequence Diagram_makeNewPayment

2) enterPatientInformation(name, securityNumber, address)

- RDD

표 16 진료비 결제_RDD_enterPatientInformation(name,securityNumber, address)

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

- GRASP pattern

표 17 진료비 결제_GRASP_enterPatientInformation(name,securityNumber, address)

0	Who will be responsible for handling the "enterPatientInformation" system event? - Register 가 전체 시스템을 represent 하고 제일 처음 받아서 처리하기 때문에 **controller pattern** 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
1	Who will be responsible for creating PatientInformation?

	- Payment 는 Patient Information 을 initialize 하기 위한 정보를 많이 가지고 있기 때문에 **Information Expert pattern** 을 적용하여 Payment 에게 Patient Information 을 생성할 responsibility 를 준다.

- Sequence Diagram

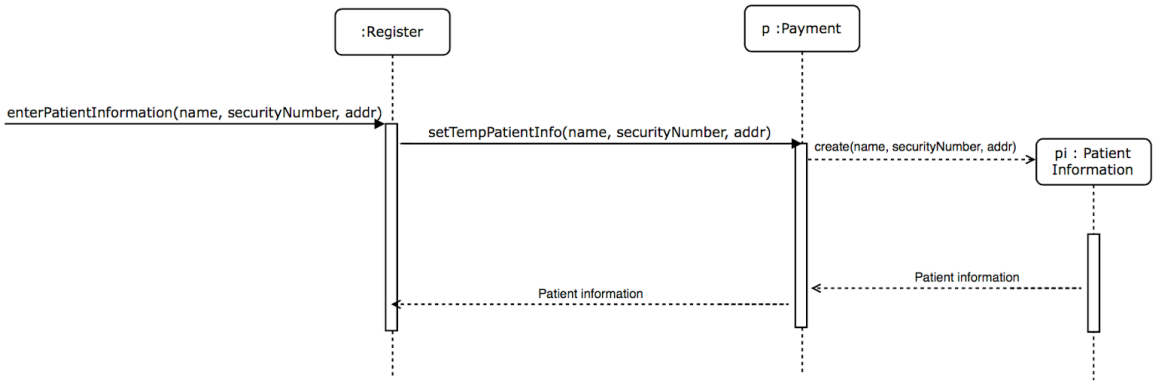


그림 9 진료비 결제_Sequence Diagram_enterPatientInformation(name,securityNumber, address)

3) confirmPatientInformation

- RDD

표 18 진료비 결제_RDD_confirmPatientInformation

Doing responsibilities	0. Register 는 confirmPatientInformation system event 를 받을 responsibility 가 있다. 1. Payment 는 외부 시스템 EMR 을 제어할 responsibility 가 있다.
Knowing responsibilities	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 19 진료비 결제_GRASP_confirmPatientInformation

0	Who will be responsible for handling the "confirmPatientInformation" system event? - Register 가 전체 시스템을 represent 하고 제일 처음 받아서 처리하기 때문에 **controller pattern** 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
1	Who will be responsible for knowing the amount to pay?

	- Payment 는 결제가 목적인 객체이므로 결제에 관한 모든 정보를 가지고 있어야 한다. 그러므로 **High cohesion pattern** 과 **knowledge expert pattern** 을 적용하여 Payment 에게 결제에 필요한 금액을 알 responsibility 를 준다. 결제 금액을 알기 위해서는 진료비를 알아야 하고 진료비에 진료비 책정 정책을 적용한 후, 나온 값을 알아야 한다.
2	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 **High cohesion pattern** 과 **knowledge expert pattern** 을 적용하여 Payment 에게 patientInformation 을 가져오고 알 responsibility 를 준다.
3	Who will be responsible for applying business rule to the treatment fee? - Payment 는 결제가 목적인 객체이므로 **knowledge expert pattern** 을 적용하여 결제비에 진료비 책정 정책을 적용하는 responsibility 를 줄 수 있다. 하지만 business rule 은 변화 가능성이 크기 때문에 이로 인한 code 변경을 최소화 하기 위해 **polymorphism pattern** 을 적용하여 PayStrategy 에게 해당 responsibility 를 준다.

- Sequence Diagram

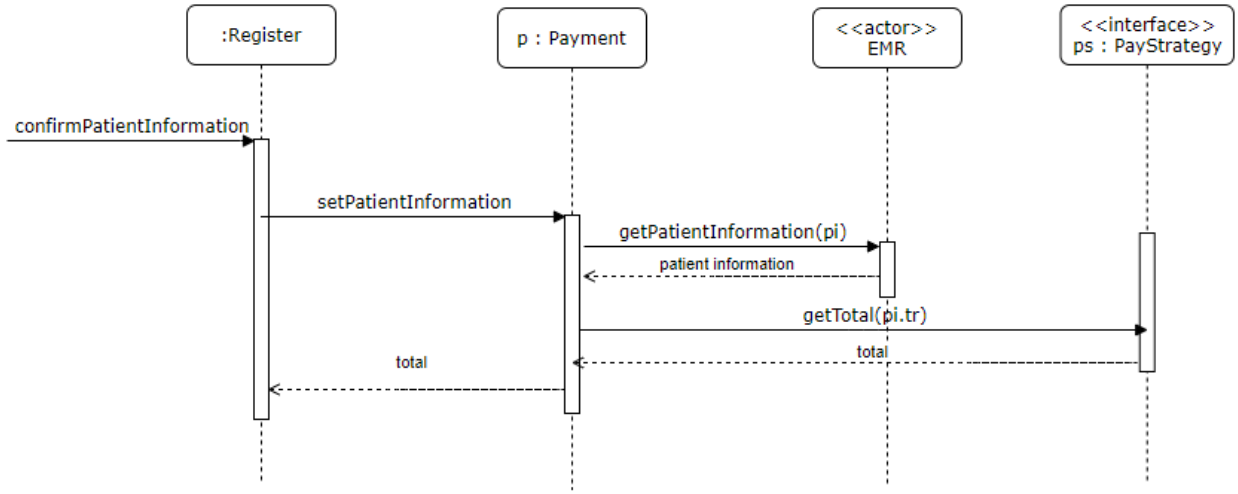


그림 10 진료비 결제_Sequence Diagram_confirmPatientInformation

4) makePayment(amount)

- RDD

표 20 진료비 결제_RDD_makePayment(amount)

Doing responsibilities	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.
Knowing responsibilities	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 21 진료비 결제_GRASP_makePayment(amount)

0	Who will be responsible for handling the "makePayment(amount)" system event? - Register 가 전체 시스템을 represent 하고 제일 처음 받아서 처리하기 때문에 **controller pattern** 을 적용하여 Register 에게 input system event 를 handling 할 responsibility 를 준다.
1	Who will be responsible for handling Payment? - 결제 시스템은 외부 시스템이기 때문에 변화가능성이 크다. 따라서 다양한 결제 시스템을 처리할 responsibility 가 있다. **Polymorphism, pure fabrication, indirection, protected variations pattern** 을 적용해서 PayAdapter interface 를 기반으로 pay 함수를 다형적으로 활용한다.
2	Who will responsible for knowing Payment for treatment is finished? - TreatmentRecord 는 진료에 대한 정보를 저장하는 것이 목적이다. 그러므로 **knowledge expert pattern**, **high cohesion pattern** 을 적용해서 TreatmentRecord 가 해당 진료의 결제에 대한 내용을 저장할 responsibility 를 준다.

- Sequence Diagram

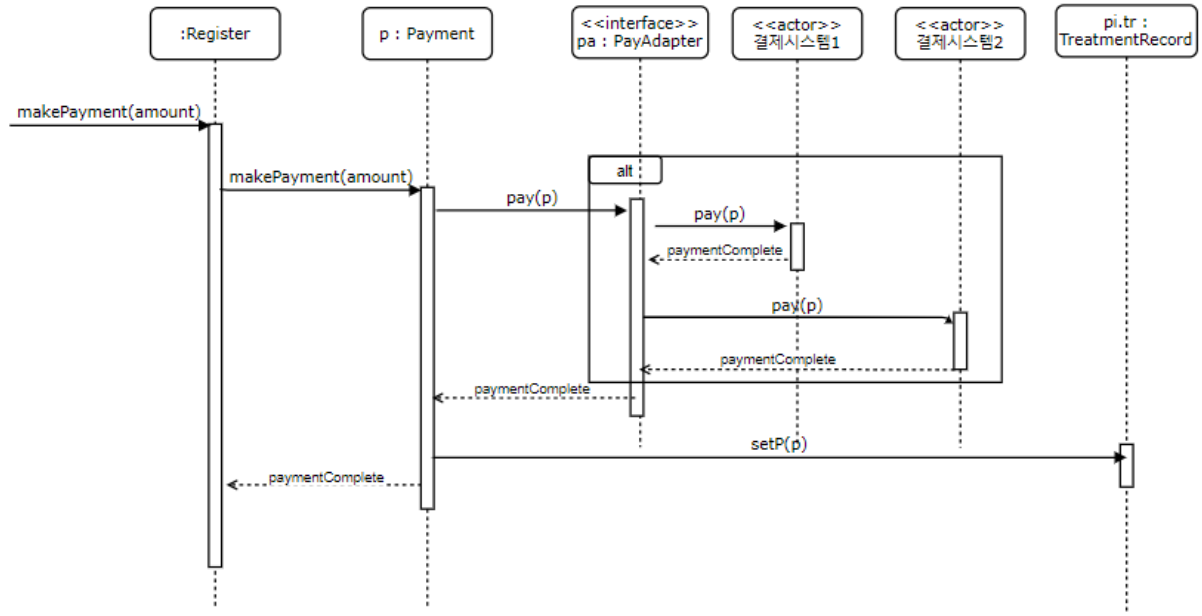


그림 11 진료비 결제_Sequence Diagram_makePayment(amount)

다) Start up Use case

- RDD

표 22 Start Up Use Case _RDD

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

- GRASP pattern

표 23 Start Up Use Case _GRASP

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

4	외부 결제 시스템의 중간 접속자인 adapter 를 생성하기 위해 Factory 를 생성해야 하므로, **creator pattern** 을 적용하여 start up use case 에서 ServicesFactory 객체를 생성해야 한다.

- Communication Diagram

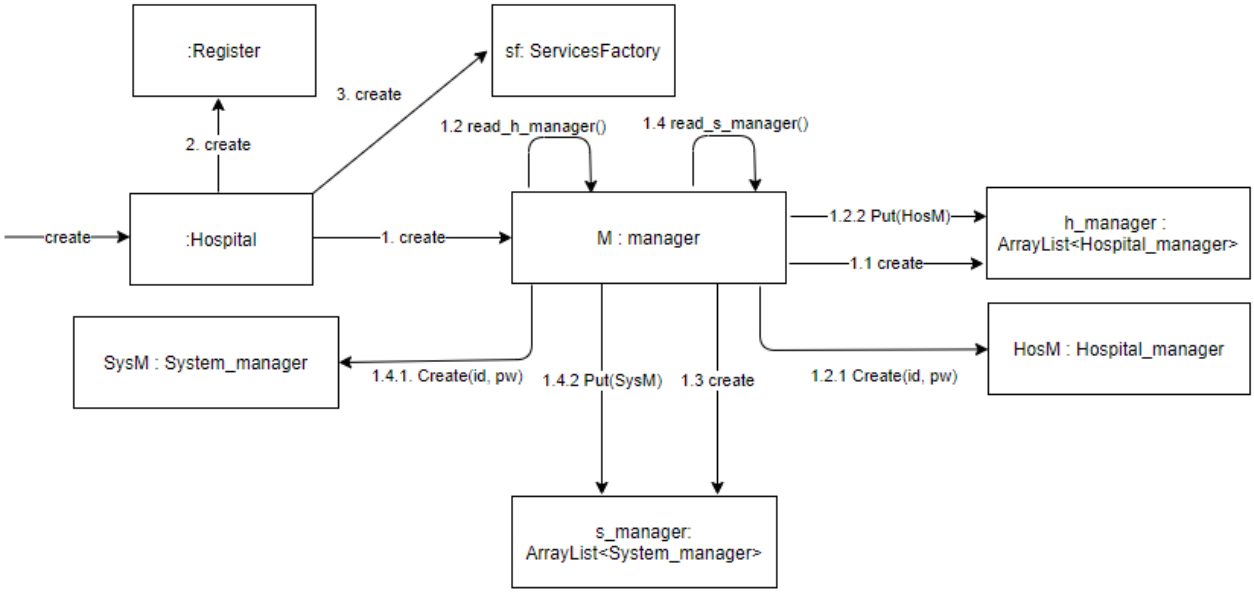


그림 12 Start Up Use Case_Communication Diagram

2.3 수정/보완된 Design Class Diagram

가) 예약 환자 접수

- class name, attribute, operation 설계 근거

표 24 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.
attributes	X
operation	makeNewReservationReceipt() : 예약 환자의 접수를 시작해야 하므로 이를 operation 으로 선정하였다. enterReservationNumber(ReservationNumber) : 예약 환자의 접수를 위해 예약 번호를 입력받아야 하므로 이를 operation 으로 선정하였다.

	confirmReservationNumber() : 예약 환자의 접수를 위해 입력된 예약 번호가 맞는지 확인해야 하므로 이를 operation 으로 선정하였다. confirmReservationInformation() : 예약 환자의 접수를 위해 예약 정보가 맞는지 확인해야 하므로 이를 operation 으로 선정하였다. requestReceipt() : 환자가 입력한 접수 내용을 외부시스템에 보내서 대기줄에 넣을 수 있어야 하므로 이를 operation 으로 선정하였다. 이는 sequence diagram 에서 가져왔다.

표 25 예약 환자 접수_Reception Class 설계 근거

Reception	
name	해당 class 는 접수에 필요한 정보를 저장하고 접수를 처리하는 데 사용된다. 목적이 접수이므로 이의 영문인 reception 을 선택했다.
attributes	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. 이의 근거로는 domain model 을 들 수 있다.
operation	enterNumber(reservationNumber) : 예약 번호를 입력받아야 하므로 이를 operation 으로 선정하였다. setReservation() : 예약 정보를 외부 시스템에서 찾아와야 하므로 이를 operation 으로 선정하였다.. confirmReservationInformation() : 해당 예약 정보가 맞는지 사용자에게 확인 받아야 하므로 이를 operation 으로 선정하였다. 이는 sequence diagram 에서 가져왔다.

표 26 예약 환자 접수_ReservationInformation Class 설계 근거

ReservationInformation	
name	해당 class 는 환자가 예약한 정보를 담고 있다. 따라서 환자의 예약 정보를 나타내는 이름인 ReservationInformation 을 선택했다.
attributes	reservationNumber : 예약 정보에 예약 번호가 있어야 예약 접수를 진행할 수 있다. 예약 번호는 예약 정보에 포함되므로, 이를 ReservationInformation 의 attribute 로 선정했다. date : 예약 정보에서 예약한 날짜를 저장하는 것은 필수이므로, 이를 ReservationInformation 의 attribute 로 선정했다. time : 예약 정보에서 예약한 시간을 저장하는 것은 필수이므로 이를 ReservationInformation 의 attribute 로 선정했다. 이의 근거로는 domain model 을 들 수 있다.
operation	X

표 27 예약 환자 접수_DoctorInformation Class 설계 근거

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

표 28 예약 환자 접수_PatientInformation Class 설계 근거

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

- relation 설계 근거

표 29 예약 환자 접수_relation 설계 근거

Register-Reception	Register 는 예약 접수를 처리하므로 Register 와 Reception 은 association 을 가져야 한다. 이의 근거로는 domain model 을 들 수 있다.
Reception-ReservationInformation	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.
ReservationInformation-DoctorInformation	예약 정보는 의사정보를 포함하며, 예약 정보가 반환될 때, 예약 진료의 의사 정보가 포함되기 때문에 ResevationInformation 과 DoctorInformation 은 association 을 가져야 한다. 이의 근거로는 domain model 을 들 수 있다.
ReservationInformation-PatientInformation	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 30 예약 환자 접수_role 설계 근거

Register-Reception	Register 는 예약 접수를 처리하며 Reception 객체는 환자의 예약 접수를 수행하는 역할을 하므로 Reception 을 r 의 role 을 주었다.
Reception-ReservationInformation	예약 접수가 처리될 때 예약 번호를 이용하기 때문에 Reception 에 해당 예약 정보가 반환되므로 ReservationInformation 을 줄인 ri 의 role 을 주었다.
ReservationInformation-DoctorInformation	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.
ReservationInformation-PatientInformation	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

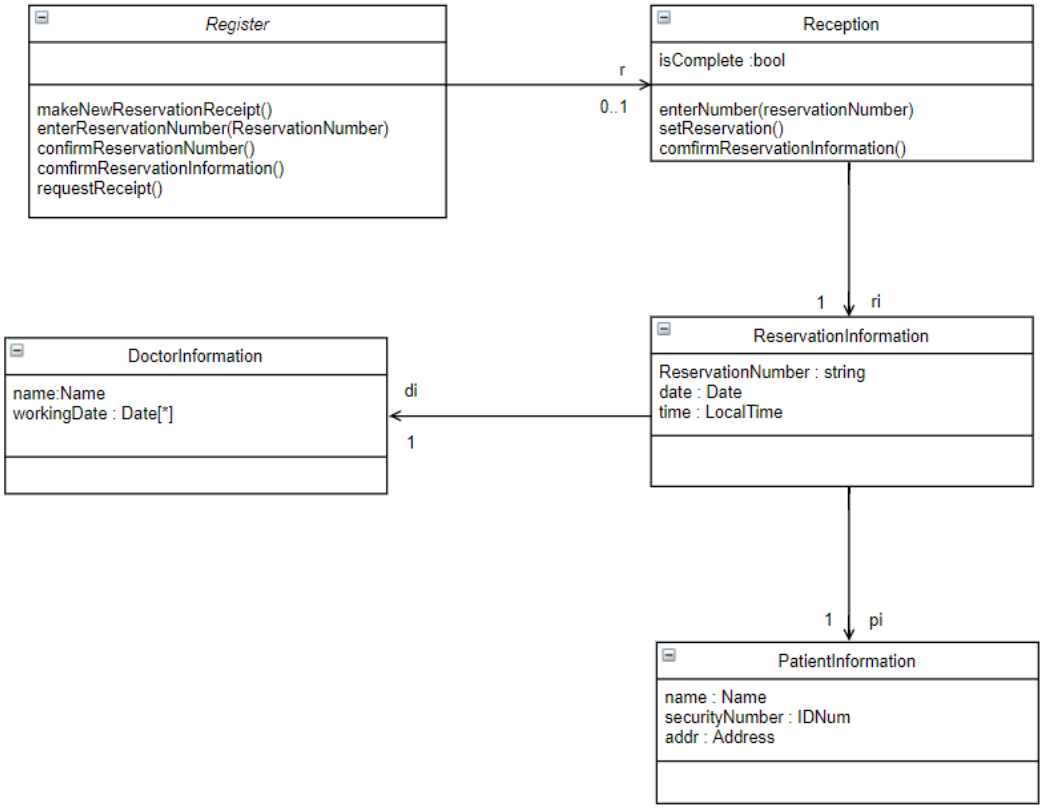


그림 13 예약 환자 접수_Class Diagram

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 31 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
operation	makeNewPayment() : 결제를 시작할 때 payment 객체를 만들기 위해서 필요하므로, 이를 operation 으로 선정하였다.

	enterPatientInformation(name,securityNumber, addr) : 결제를 시작할 때 사용자의 정보를 입력 받아야 하므로, 이를 operation 으로 선정하였다. confirmPatientInformation() : 환자로부터 결제할 금액을 입력 받고 결제를 처리해야 하므로, 이를 operation 으로 선정하였다. makePayment(amount) : 환자로부터 결제할 금액을 입력 받고 결제를 처리해야 하므로, 이를 operation 으로 선정하였다. 이의 근거로는 system sequence diagram 과 sequence diagram 를 들 수 있다.

표 32 진료비 결제_TreatmentRecord Class 설계 근거

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

표 33 진료비 결제_Payment Class 설계 근거

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

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

표 34 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 이름으로 선택했다.
attributes	X
operation	pay() : 외부 결제 시스템과 내부시스템이 연동해야 하므로, 이를 operation 으로 선정하였다. 이는 sequence diagram 에서 가져왔다.

표 35 진료비 결제_PayStrategy Class 설계 근거

PayStrategy	
name	해당 class 는 진료비 책정 정책에 관한 변동 가능성이 있기 때문에 내부에서 인터페이스를 구현해서 변동하는 결제 정책을 쉽게 적용하기 위한 class 이다. 결제 정책을 나타내는 의미를 가진 PayStrategy 를 이름으로 선택했다.
attributes	X
operation	getTotal() : 진료비 책정 정책을 거친 진료비를 반환해야 하므로, 이를 operation 으로 선정했다. 이는 sequence diagram 에서 가져왔다.

표 36 진료비 결제_ServicesFactory Class 설계 근거

ServicesFactory	
name	해당 class 는 외부 결제 시스템의 중간 접속자인 adapter 와 진료비 책정 정책을 담당하는 PayStrategy 를 생성하므로, 이는 Factory 를 나타낼 수 있는 이름으로 하는 것이 좋다. 따라서 이를 나타내는 ServicesFactory 를 이름으로 선택했다.
attributes	X
operation	getInstance() : singleton pattern 적용 시 이미 해당 객체가 생성되어 있으면 생성되어진 객체를 그대로 반환하고 생성 되어진 객체가 없으면 객체를 생성해서 반환해야 하기 때문에 ServicesFactory 객체에 대한 생성이 필요하다. getPayAdapter_1() : 외부 결제 시스템과 내부 시스템을, adapter 패턴을 기반으로 연동시키기 위해서 일반화 과정이 필요하다.

	getPayStrategy_1() : 여러가지 진료비 책정 정책을 Strategy 패턴을 이용해서 시스템에 적용하기 위해 필요하다. 이는 sequence diagram 에서 가져왔다.

- relation 설계 근거

표 37 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.
Payment-PayStrategy	Payment 는 진료비 결제를 목적으로 하고 이를 하기 위해서는 총 결제액을 계산하기 위해 진료비 책정 정책을 적용해야 한다. 이 때 PayAdapter 를 사용한다. 그러므로 Payment 와 PayAdapter 는 association 이 필요하다.
PayAdapter – PayAdapter_1 and PayAdapter_2	PayAdapter_1 과 Pay Adapter_2 는 시스템에 실제로 적용되는 결제 시스템의 중간 다리 역할을 하는 class 이다. 이는 interface 인 PayAdapter 를 통해 결제 시스템 변경 시 코드 변경을 최소화할 수 있다. 그러므로 PayAdapter 와 PayAdapter_1, PayAdapter_2 는 generalization 을 가져야 한다.
PayStrategy- PayStrategy_1 and PayStrategy_2	PayStrategy_1 과 PayStrategy_2 는 시스템에 실제로 적용되는 진료비 책정 정책 적용의 역할을 하는 class 이다. 이는 interface 인 PayStrategy 를 통해 진료비 책정 정책 변경 시 코드

	변경을 최소화할 수 있다. 그러므로 PayStrategy 와 PayStrategy_1, PayStrategy_2 는 generalization 을 가져야 한다.
ServicesFactory – PayAdapter_1 and PayAdapter_2	ServiceFactory 는 외부 결제 시스템의 중간 다리 역할을 하는 PayAdapter_1 과 PayAdapter_2 객체의 생성과 저장을 담당한다. 그러므로 ServiceFactory 와 PayAdapter_1, PayAdapter_2 는 association 을 가져야 한다.
ServicesFactory-PayStrategy_1 and PayStrategy_2	진료비 결제시 진료비 책정 정책을 담당하는 PayStrategy_1 과 Pay_Strategy_2 가 필요하다. 따라서 ServicesFactory class 를 만들어서 PayStrategy 의 생성과 저장을 담당하여 다양한 진료비 책정 정책을 제공할 수 있게 된다. 그러므로 ServicesFactory 와 PayStrategy_1, PayStrategy_2 는 association 이 필요하다.

- role 설계 근거

표 38 진료비 결제_role 설계 근거

TreatmentRecord - PatientInformation	환자 정보에는 환자가 받은 진료의 기록이 포함된다. 그러므로 TreatmentRecord 에게 TreaatmentRecord 를 줄인 tr 의 role 을 주었다.
PatientInformation - Payment	진료비 결제가 처리될 때 환자 정보를 이용하기 때문에 Payment 에 해당 환자 정보가 반환되므로 PatientInformation 을 줄인 pi 의 role 을 주었다.
Register-Payment	Register 는 결제를 처리하며 Payment 는 환자의 진료비 결제를 수행하는 역할을 하므로 Payment p 의 role 을 주었다.
Payment-PayAdapter	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.
TreatmentRecord-Payment	Payment 가 처리한 내용을 TreatmentRecord 에 전달해주면서 기록해야 한다. 따라서 Payment class 의 instance 인 p 의 role 을 주었다.
ServicesFactory-PayAdapter_1, PayAdapter_2, PayStrategy_1, PayStrategy_2	ServiceFactorys 는 제공할 서비스인 PayAdapter_1, PayAdapter_2, PayStrategy_1, PayStrategy_2 을 저장하여 나중에 또 이를 요청하는 객체에게 이를 반환해야 한다. 따라서 ServicesFactory 가 제공하는 서비스인 PayAdapter_1, PayAdapter_2, PayStrategy_1, PayStrategy_2 객체들에게 payAdapter_1, payAdapter_2, payStrategy_1, payStrategy_2 의 role 을 주었다.

- Class Diagram

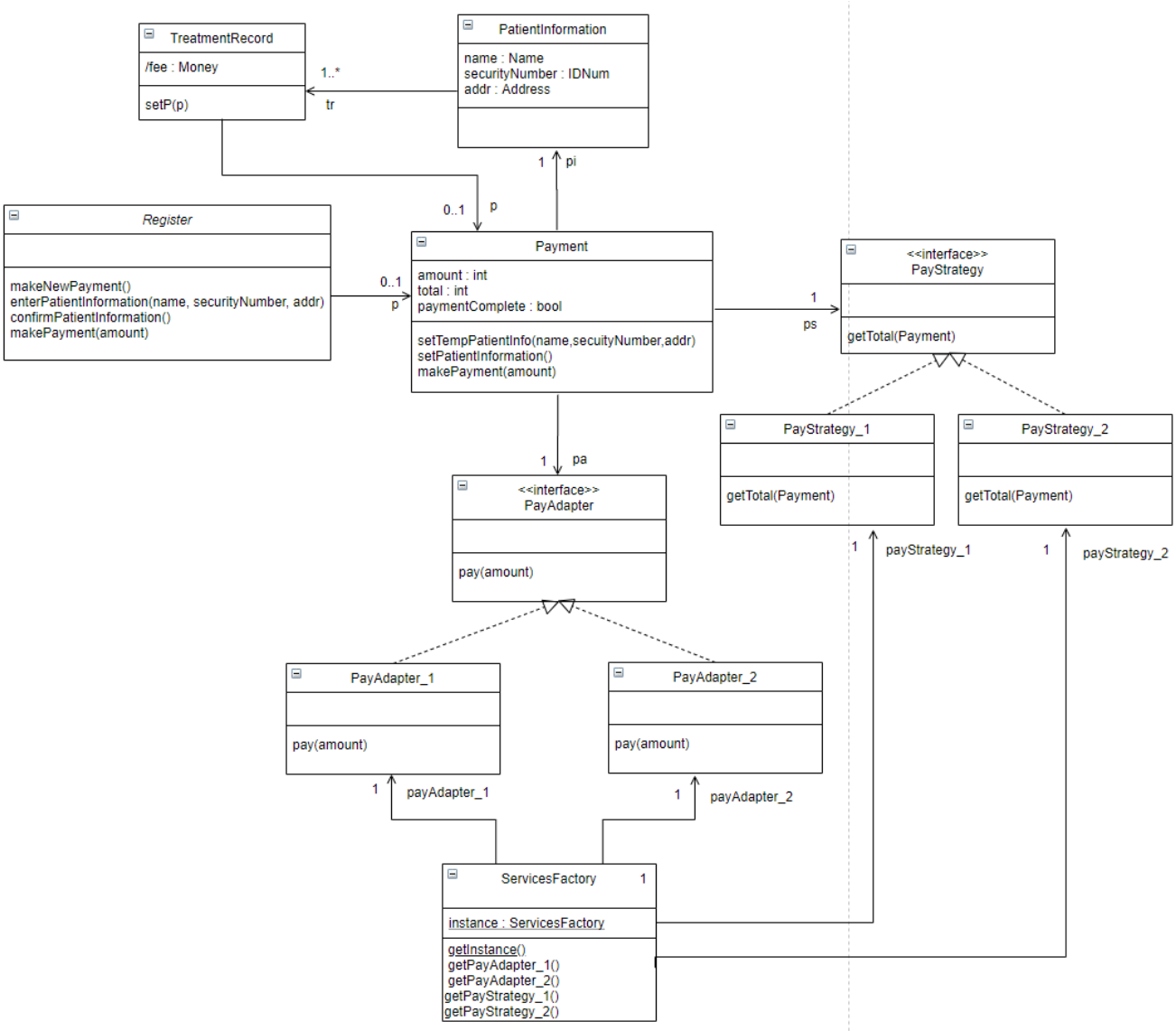


그림 14 진료비 결제_Class Diagram

다) Start Up Use Case

- class name, attribute, operation 설계 근거

표 39 Start Up Use Case_Hospital Class 설계 근거

Hospital	
name	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.
attributes	X
operation	X

표 40 Start Up Use Case _Register Class 설계 근거

Register	
name	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.
attributes	X
operation	X

표 41 Start Up Use Case_Manager Class 설계 근거

Manager	
name	해당 class 는 병원 관리자와 시스템 관리자의 정보를 contain 한다. 따라서 관리자를 나타내는 Manager 를 이름으로 선택했다.
attributes	X
operation	read_h_manager() : 병원 관리자의 id 와 password 에 대한 정보를 불러와야 하므로, 이를 operation 으로 선정하였다. 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. read_s_manager() : 시스템 관리자의 id 와 password 에 대한 정보를 불러와야 하므로, 이를 operation 으로 선정하였다. 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.

표 42 Start Up Use Case_Hospital manager Class 설계 근거

Hospital Manager	
name	해당 class 는 병원 관리자의 ID 와 Password 를 저장하는데 사용된다. 따라서 병원 관리자를 나타내는 Hospital_manager 를 이름으로 선택했다.
attributes	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.
operation	X

표 43 Start Up Use Case_System_manager Class 설계 근거

System_manager	
name	해당 class 는 시스템 관리자의 ID 와 Password 를 저장하는 데 사용된다. 따라서 시스템 관리자를 나타내는 System_manager 를 이름으로 정했다.
attributes	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.
operation	X

표 44 Start Up Use Case_ServicesFactory Class 설계 근거

ServicesFactory	
name	해당 class 는 외부 결제 시스템의 중간 접속자인 adapter 와 진료비 책정 정책을 담당하는 PayStrategy 를 생성하며, 이는 Factory 를 나타낼 수 있는 이름으로 하는 것이 좋다. 따라서 이를 나타내는 ServicesFactory 를 이름으로 선택했다.
attributes	X
operation	X

- relation 설계 근거

표 45 Start Up Use Case_relation 설계 근거

Hospital-Register	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.
Hospital-Manager	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.
Manager-Hospital_manager	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.
Manager-System_manager	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.

Hospital - ServicesFactory	외부 결제 시스템의 중간 접속자인 adapter 를 생성하는 Factory 객체를 만들어야 하기 때문에, Hospital 가 ServicesFactory Class 를 생성할 필요가 있으므로 Hospital 과 ServicesFactory 는 association 을 가져야 한다.

- role 설계 근거

표 46 Start Up Use Case_role 설계 근거

Hospital-Register	Hospital 에는 system 초기화 시 Register 가 생성될 필요가 있으므로 Register 를 줄인 rg 의 role 을 주었다.
Hospital-Manager	Hospital 에는 system 초기화 시 Manager 가 생성될 필요가 있으므로 Manager 를 줄인 M 의 role 을 주었다.
Manager-Hospital_manager	Manager 에는 system 초기화 시 Hospital_manager 가 생성될 필요가 있으므로, Hospital_manager 를 줄인 h_manager 의 role 을 주었다.
Manager-System_manager	Manager 에는 system 초기화 시 System_manager 가 생성될 필요가 있으므로 System_manager 를 줄인 s_manager 의 role 을 주었다.
Hospital - ServicesFactory	Hospital 에는 맡은 임무를 수행할 때 필요한 서비스를 제공하는 ServicesFactory 가 생성될 필요가 있으므로, ServicesFactory 를 줄인 sf 의 role 을 주었다.

- Class Diagram

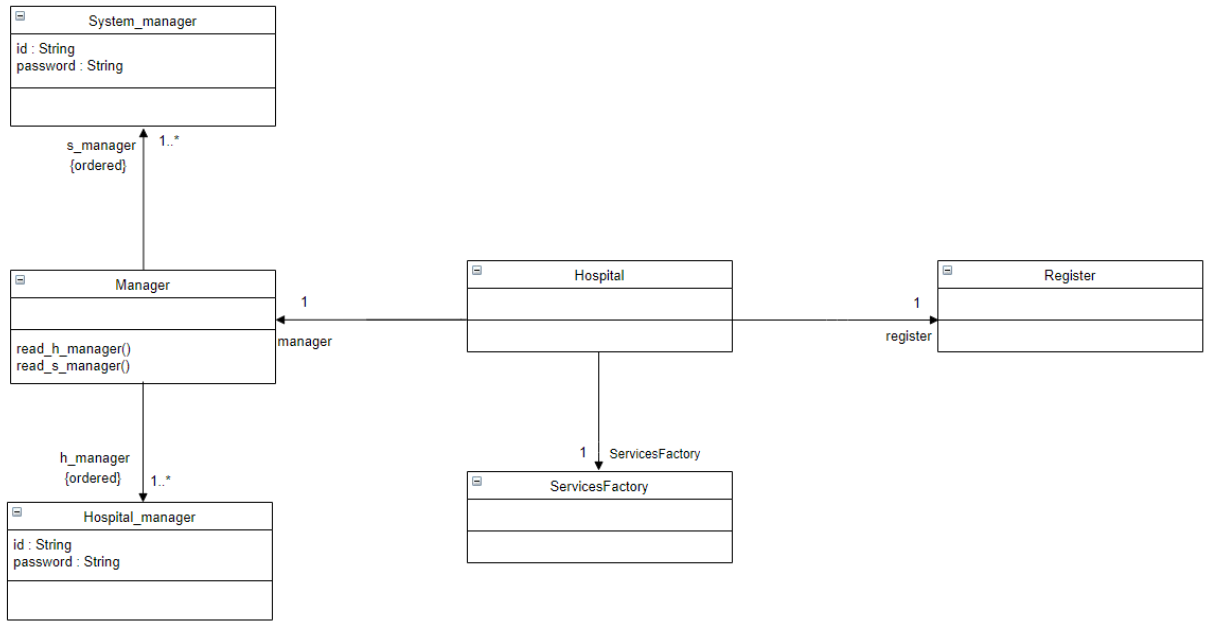


그림 15 Start Up Use Case_Class Diagram

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
Elaboration phase 4-3	20.07.16	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

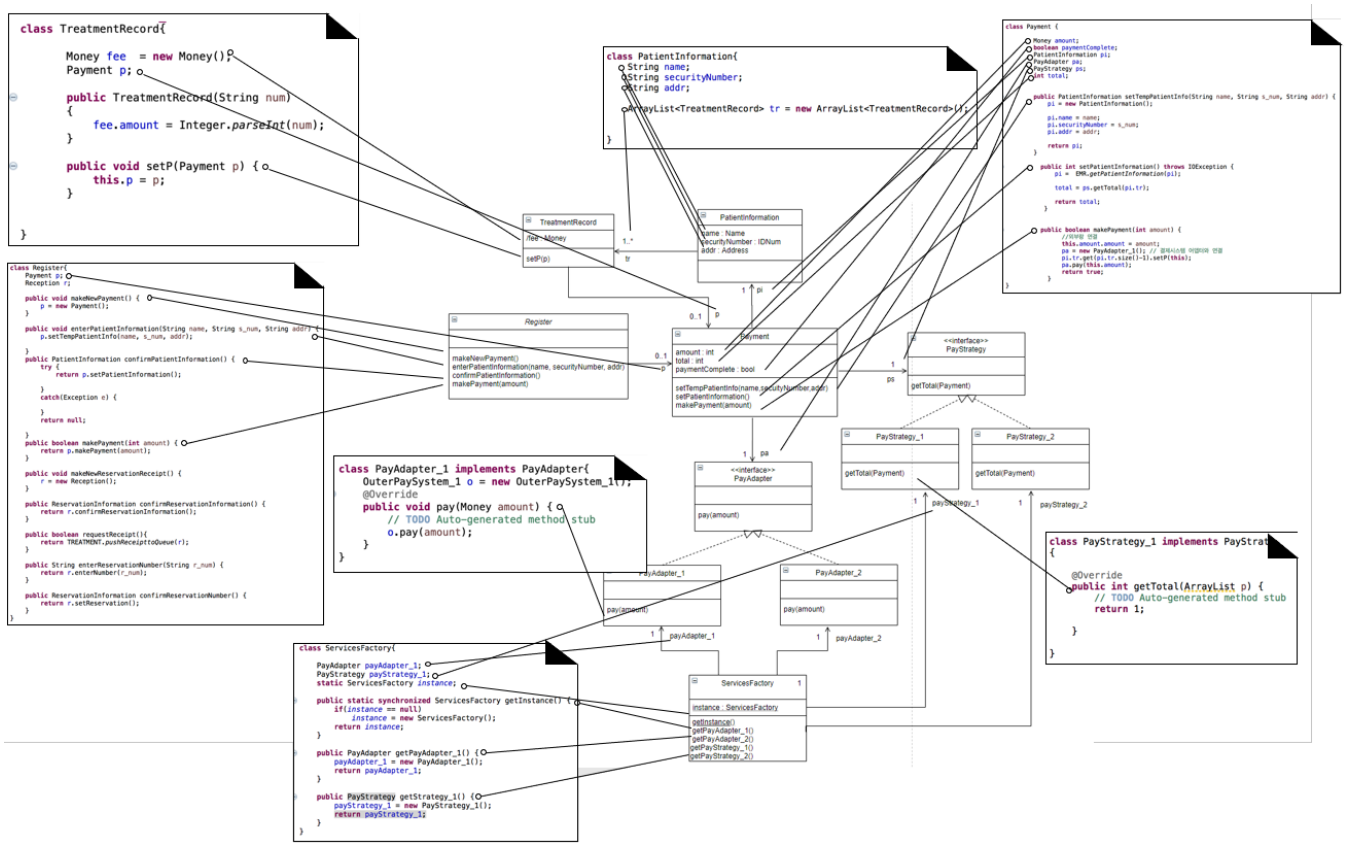


그림 16 Source Code-Class Diagram mapping_진료비 결제

나) Start up Use Case

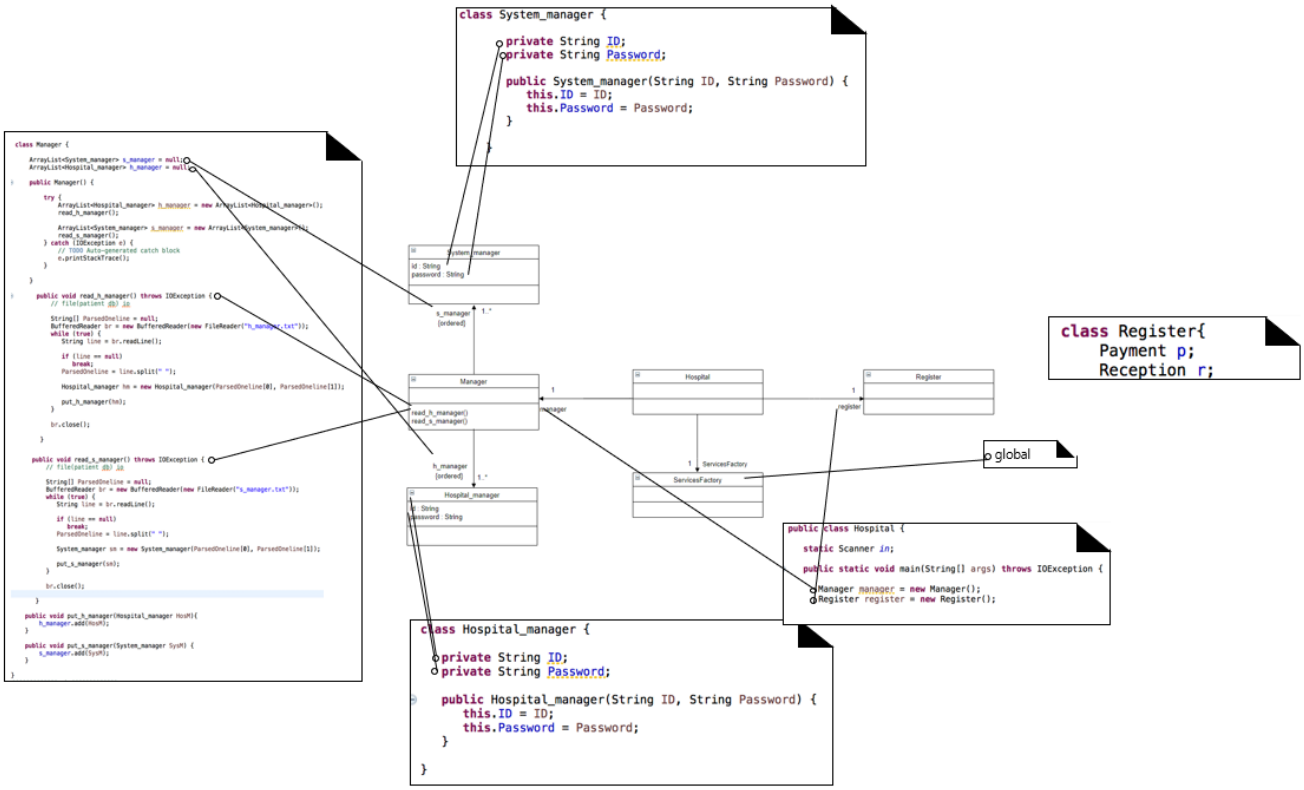


그림 17 Source Code-Class Diagram mapping_start up use case

3.3 수정/보완된 Source Code

가) Register

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

    public Register() {
        sf = new ServicesFactory();
    }

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

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

    }
    public int confirmPatientInformation() {
        try {
            return p.setPatientInformation();

        }
        catch(Exception e) {

        }
        return -1;

    }
    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 18 SourceCode_Register

나) ReservationInformation

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

그림 19 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 20 SourceCode_Reception

라) PatientInformation

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

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


}
```

그림 21 SourceCode_PatientInformation

마) DoctorInformation

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

}
```

그림 22 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 23 SourceCode TreatmentRecord

사) Payment

```
class Payment {

        Money amount;
        boolean paymentComplete;
        PatientInformation pi;
        PayAdapter pa;
        PayStrategy ps;
        int total;

        public Payment() {
            pa = ServicesFactory.getInstance().getPayAdapter_1();
            ps = ServicesFactory.getInstance().getStrategy_1();
        }

        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 int setPatientInformation() throws IOException {
              pi =  EMR.getPatientInformation(pi);

              total = ps.getTotal(pi.tr);

              return total;
           }


          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 24 SourceCode_Payment

아) Money

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

그림 25 SourceCode_Money

자) payAdapter

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

그림 26 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 27 SourceeCode_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 28 SourceCode_외부 결제

카) PayStrategy

```
interface PayStrategy{
    public int getTotal(ArrayList p);
}
```

그림 29 SourceCode_PayStrategy

타) PayStrategy_1

```
class PayStrategy_1 implements PayStrategy
{

    @Override
    public int getTotal(ArrayList p) {
        // TODO Auto-generated method stub
        return 1;

    }

}
```

그림 30 SourceCode_PayStrategy_1

파) ServicesFactory

```
class ServicesFactory{

    PayAdapter payAdapter_1;
    PayStrategy payStrategy_1;
    static ServicesFactory instance;

    public static synchronized ServicesFactory getInstance() {
        if(instance == null)
            instance = new ServicesFactory();
        return instance;
    }

    public PayAdapter getPayAdapter_1() {
        payAdapter_1 = new PayAdapter_1();
        return payAdapter_1;
    }

    public PayStrategy getStrategy_1() {
        payStrategy_1 = new PayStrategy_1();
        return payStrategy_1;
    }
}
```

그림 31 SourceCode_ServicesFactory

하) 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 32 SourceCode_System_manager

가) 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 33 SourceCode_Hospital_manager

나) 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[] ParsedOneline = null;
          BufferedReader br = new BufferedReader(new FileReader("h_manager.txt"));

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

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

          while (true) {
             String line = br.readLine();
             if (line == null)
                break;
             ParsedOneline = line.split(" ");
             System_manager sm = new System_manager(ParsedOneline[0], ParsedOneline[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 34 SourceCode_manager

다) 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 35 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.enterPatiemtInformation(name, security_number, "");
       System.out.println("\n" + name + " " + security_number + "\n" + addr+ " "+"정보가 맞습니까?(y,n) : ");
       opt = in.nextLine();

       if (opt.equals("y")) {
           int total = register.confirmPatientInformation();
         if (total != -1)
         {
                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 ");
               }


         }
         else {
            System.out.println("등록된 정보가 없습니다. \n");
            opt = "n";
         }
      }
   } while (opt.equals("n"));
}
```

그림 36 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 47 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	- 시스템이 정상 작동하고 있어야 한다. - 시스템이 환자 정보 시스템과 정상적으로 연결되어 있어야 한다. - 환자는 예약 번호를 가지고 있다.				

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(reservatio 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
Postcondition	- 중간에 접수가 중단되지 않았다면, 접수 정보가 정상적으로 원하는 의사의 대기줄에 등록된다.					

나) 진료비 결제

표 48 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	- 시스템이 정상 작동하고 있어야 한다. - 시스템이 환자 정보 시스템과 정상적으로 연결되어 있어야 한다. - 시스템이 결제 시스템과 정상적으로 연결되어 있어야 한다. - 환자는 이름, 주민등록번호, 주소를 가지고 있어야 한다.				

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 49 Test 보고서_Start up Use case

Test Case ID	3	**Test Name**	StartUp	**Use Case**	StartUp
Description	시스템을 시작할 때 초기화하는 과정을 테스트한다.				
Precondition	- 시스템이 정상 작동하고 있어야 한다. - 시스템은 초기화 단계여야 한다.				

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 hongkildong2, pAsSwOrd22 hongkildong3, pAsSwOrd33	병원 관리자의 id, password 정보 하나를 읽어온다.	honkildong1, pAsSwOrd11	P
			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
Postcondition		- 병원 관리자와 시스템 관리자의 목록이 정상적으로 생성되고 초기화된다.				