객체지향프로그래밍및실습
(OOP-HW2-201721070)
2차 프로그래밍과제 보고서

이름 : 장효택

학번 : 201721070

학과 : 미디어학과

학년 : 2학년

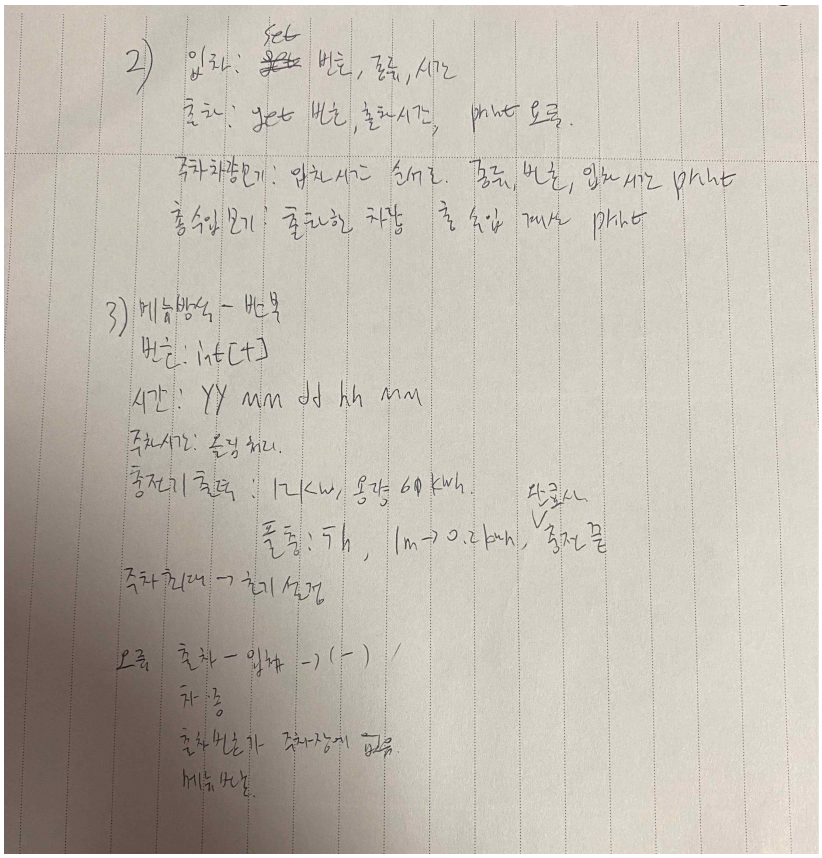
나) 소개

구현한 부분	구현 여부 (O/X)
상속(inheritance)과 다형성(polymorphism)을 활용	O
주요 데이터는 반드시 클래스화	O
입력 값에 대한 유효성 검사 등 오류 검사	O
주차장의 주차 최대 차량 수는 초기에 설정	O
주차시간의 계산 시 시간은 올림 하여 처리	O
메뉴방식으로 처리하며 반복적으로 처리	O
차량 입차 기능 및 출차 기능	O
입차순서로 정렬하여 주차 차량 보기 기능	O
차량별 주차시간 계산 및 금액 계산 기능	O
전기 승용차의 전기 충전 요금 계산 기능	O

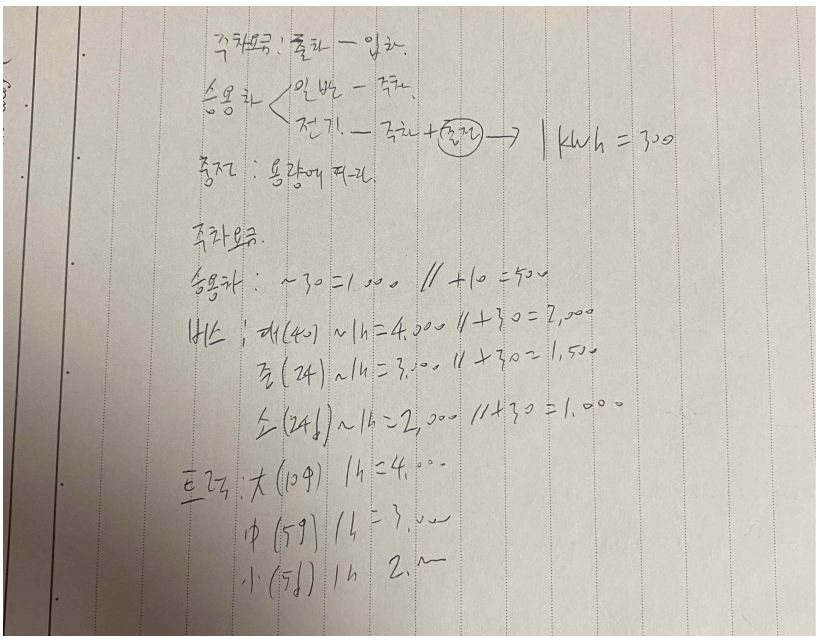
프로그램 소개

입차 시 키보드를 통해 받은 사용자의 입력에 따라 차량을 객체화하여 주차장 ArrayList에 담아 입차를 합니다.
출차 시 출차 시간을 받고 입차 시간과 출차 시간의 차를 가지고 각 차량 종류마다 주차시간 및 주차요금을 기준에 맞게 계산하고 요금을 받아 주차장을 관리하는 프로그램입니다.

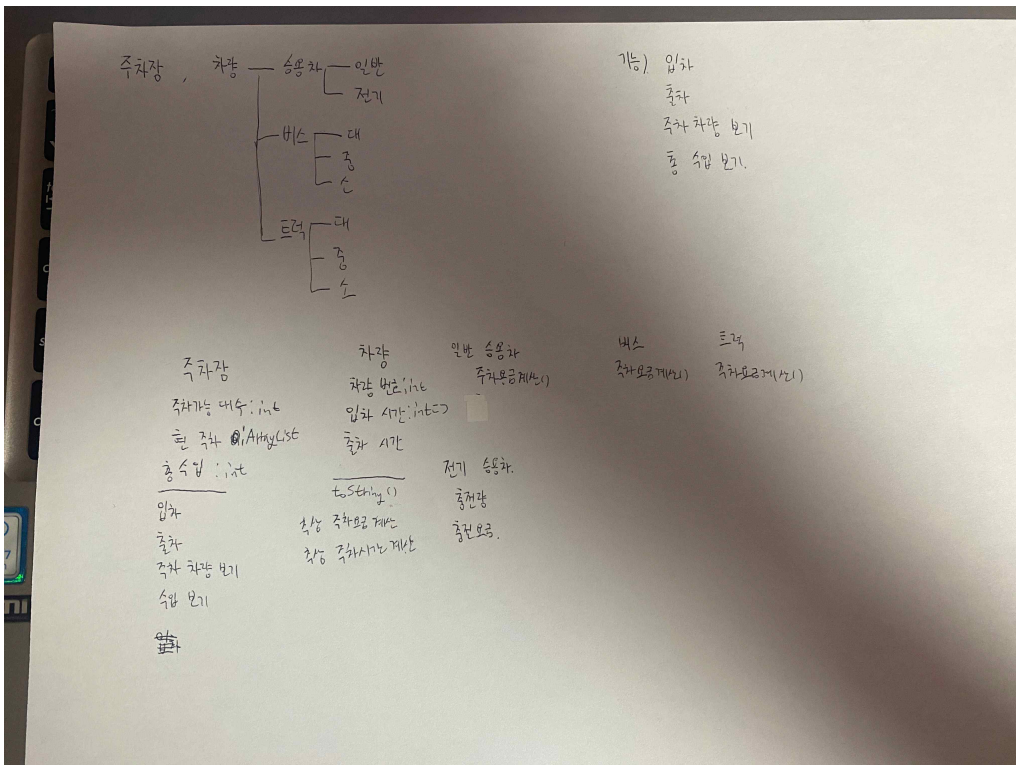
다) 분석 및 설계



과제 시작 전, 주어진 문제 안내 pdf 파일을 통해 원하는 요구사항과 필요한 속성을 분석했다. 요구사항을 분석하며 설계의 큰 틀을 구축해나갔다.

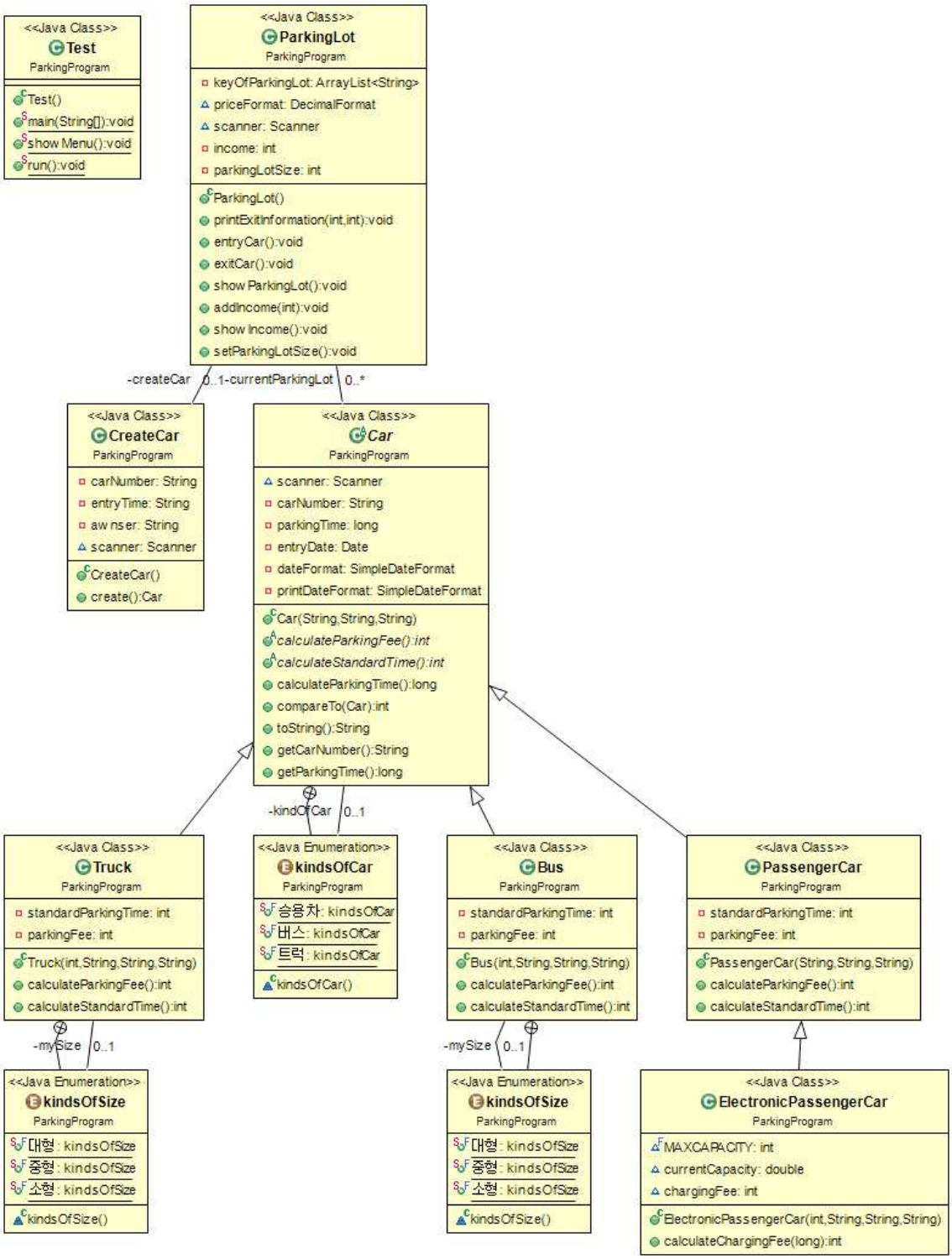


위 내용과 비슷하게 pdf를 요구사항을 분석해나갔다.



큰 틀을 구축해야 더 안정적인 프로그래밍이 가능하다 생각했다. 요구사항을 구현하기 위해, 필요한 클래스를 생각하고 클래스 간의 관계와 각 클래스에 필요하겠다 느끼는 속성과 메소드를 적어 나갔다.
처음엔 아주 세부적인 메소드와 속성을 생각하지 못했지만, 최대한 가이드라인을 맞추기 위해 큰 틀이라도 잡고 시작했다.

- class diagram -



각 클래스에 대한 설명

Test Class	
Operation	
showMenu():void	관리 프로그램의 항목을 띄운다.
run():void	스캐너로 번호를 받고 그에 맞는 항목을 실행한다. 만약 항목에 없는 번호면 오류를 띄우고 처음 시작으로 돌아간다. case 5: 종료를 선택할 때까지 while문을 통해 반복적으로 프로그램을 작동시킨다.

ParkingLot Class	
Attribute	
keyOfParkingLot: ArrayList<String>	차량번호를 넣을 ArrayList로 entry()할 때, 차량 번호를 저장하여 나중에 차량번호로 인덱스를 찾을 때 활용한다.
currentParkingLot: ArrayList<String>	주차장 ArrayList로서, entry()를 하면 이곳에 객체를 저장한다.
priceFormat: DecimalFormat	금액을 출력하기 위한 포맷
income: int	수입을 나타내는 int
parkingLotOfSize: int	주차장의 크기를 나타내는 int
Operation	
entryCar(): void	주차장의 현재 남은 크기를 확인하고 CreateCar.create()를 통해 차량 객체를 만들고 ArrayList에 차량과 차량번호를 삽입한다.
exitCar(): void	출차번호를 받고 keyOfParkingLot에서 검색을 하여 인덱스를 받고 동일한 인덱스로 currentParkingLot을 검색하여 차량을 찾고, 각 객체의 주차시간과 요금을 계산하여 수입을 더하고 ArrayList에서 제거하여 출차시킨다.
showParkingLot(): void	Car()에 compareTo를 오버라이딩하여 입차시간으로 정렬하게 만들고, for-each문을 통해 정렬된 차량을 프린트한다. 프린트 시, toString()을 오버라이딩하여 주어진 형식에 맞게 출력한다.
addIncome(int): void	요금을 매개변수로 받아 income에 더한다.
printExitInformation (int, int): void	단위가 분으로 된 시간을 H,M으로 변환하여 시간과 요금을 출력한다.

CreateCar Class	
Attribute	
carNumber: String	차량번호를 뜻하는 속성
entryTime: String	입차 시간을 뜻하는 속성, 추후 포맷을 통해 Date 형식으로 변환
awnser: String	차량 종류와 용량을 받는 String. split(" ")를 통해 종류와 용량을 저장할 것이다.
Operation	
create():Car	awnser을 통해 종류와 용량을 받고, 나머지 차량 번호, 입차시간을 저장하고, 종류와 용량을 통해 각 객체의 생성자에 매개변수를 전달하여 객체를 생성한다. 그리고 만들어진 객체를 반환하여 entry()에 활용한다.

Car Class	
Attribute	
carNumber: String	차량번호를 뜻하는 속성
entryTime: String	입차 시간을 나타내는 속성
entryDate: Date	위의 String을 양식에 맞게 포맷팅하여 변환한 Date 날짜 속성
dataFormat: SimpleDataForamt	입차, 출차시간을 변환하기 위한 포맷
printDataFormat: SimpleDataForamt	차량 정보 출력을 위한 포맷
Operation	
Car(String,String,String)	차량 종류에 따라 enum 데이터를 가져와 종류를 나누고, 차량번호와 입차시간을 set한다.
calculateParkingFee(): int	요금을 계산하는 Abstract method로서 추후 각 객체에서 구현할 것이다.
calculateStandardTime(): int	요금을 계산하는 Abstract method로서 추후 각 객체에서 구현할 것이다.
calculateParkingTime(): long	출차시간을 받고, 입차시간과 차이를 분으로 받아서 리턴한다.
compareTo(Car): int	showParkingLot() 함수에서 sort를 위해 오버라이딩했다.

	입차시간을 기준으로 정렬한다.
toString(): String	프린트에 Car를 사용하기 위해, 주어진 포맷대로 출력하기 위해 toString() 오버라이딩하였다.
kindsOfSize :Enum	
승용차, 버스, 트럭	차량 종류를 구분한다.

Truck Class	
Attribute	
standardPakingTime: int	주어진 주차시간을 각 기준에 맞게 반올림하여 계산한 표준 주차시간 속성이다.
parkingFee: int	각 기준에 맞게 주차요금을 계산한 요금 속성이다.
Operation	
Truck(String, String, String)	kindOfCar, carNumber, entryTime은 super를 통해 Car의 생성자를 활용하고, capacity로 차량의 세부 종류를 결정한다.
calculateParkingFee() : int	차량의 표준 주차 시간을 가지고 주어진 계산식을 계산한다.
calculateStandardTime(): int	실제 주차시간을 받고 주어진 계산식을 통해 표준 주차 시간을 구한다.
kindsOfSize :Enum	
대형, 중형, 소형	차량의 capacity에 따라 종류를 구분한다.

Bus Class	
Attribute	
standardPakingTime: int	주어진 주차시간을 각 기준에 맞게 반올림하여 계산한 표준 주차시간 속성이다.
parkingFee: int	각 기준에 맞게 주차요금을 계산한 요금 속성이다.
Operation	
Bus(String, String, String)	kindOfCar, carNumber, entryTime은 super를 통해 Car의 생성자를 활용하고, capacity로 차량의 세부 종류를 결정한다.
calculateParkingFee() : int	차량의 표준 주차 시간을 가지고 주어진 계산식을 계산한다.
calculateStandardTi	실제 주차시간을 받고 주어진 계산식을 통해 표준 주차 시간을

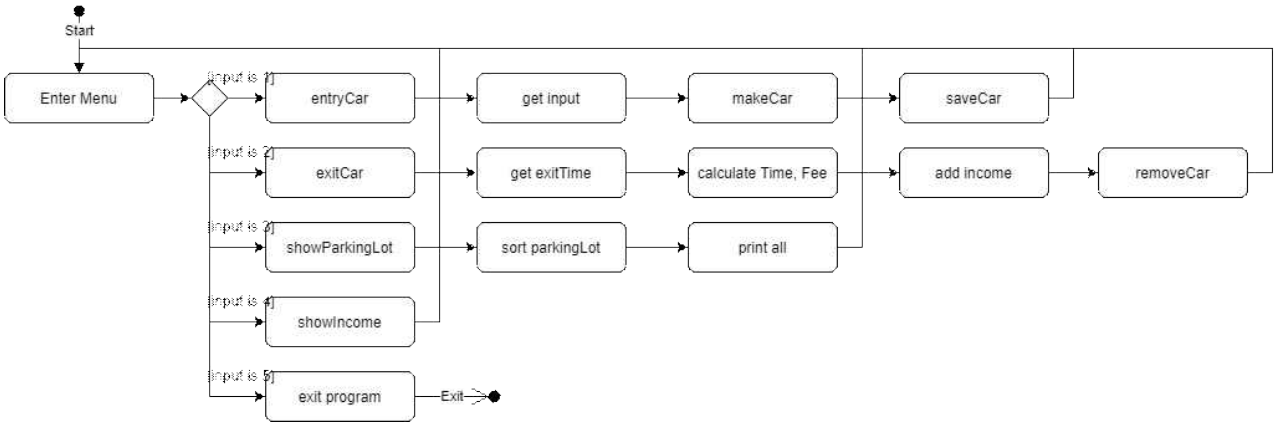
me(): int	구한다.
kindsOfSize :Enum	
대형, 중형, 소형	차량의 capacity에 따라 종류를 구분한다.

PassengerCar Class	
Attribute	
standardPakingTime: int	주어진 주차시간을 각 기준에 맞게 반올림하여 계산한 표준 주차시간 속성이다.
parkingFee: int	각 기준에 맞게 주차요금을 계산한 요금 속성이다.
Operation	
PassengerCar(String, String, String)	kindOfCar, carNumber, entryTime은 super를 통해 Car의 생성자를 활용한다.
calculateParkingFee() : int	차량의 표준 주차시간을 가지고 주어진 계산식을 계산한다.
calculateStandardTime(): int	실제 주차시간을 받고 주어진 계산식을 통해 표준 주차시간을 구한다.

EletronicPassengerCar Class	
Attribute	
MAXCAPACITY: int	차량의 최대 배터리 충전 용량을 정의한다.
currentCapacity: double	현재 충전된 용량을 나타낸다.
chargingFee: int	충전 용량에 따라 충전요금을 계산한다.
Operation	
ElectronicPassengerCar(int, String, String, String)	kindOfCar, carNumber, entryTime은 super를 통해 Car의 생성자를 활용하고, capacity로 차량의 현재 충전용량을 set한다.
calculateChargingFee(long): int	While문을 통해 주차시간을 줄이고 요금을 올리며 주차시간이 끝나거나 완충되면 while문을 탈출한다.

예외처리	
차량번호 입력 Exception	차량번호가 4자리 정수가 아니면 InputMismatchException을 발생시킨다.
날짜정보 포맷팅 Exception	주어진 포맷에 맞지 않은 입력이 들어오면 ParseException을 발생시킨다.
주차장 만차 Exception	초기 설정한 주차장 크기가 꽉 찼을 때, entryCar()를 하면 IndexOutOfBoundsException를 발생시킨다.
입차, 출차 시간차 Exception	출차시간이 입차시간보다 빠르면 ArithmeticException을 발생시킨다.
차량 종류 Exception	존재하지 않는 차량종류를 입력하면 InputMismatchException을 발생시킨다.
출차번호 Exception	주어진 인자가 출차번호리스트와 일치하지 않으면 IllegalArgumentException을 발생시킨다.
메뉴 번호 Exception	메뉴번호 1-5 외의 숫자를 입력하면 default에서 오류를 발생시키고 다시 while을 반복한다.

-Activity Diagram-



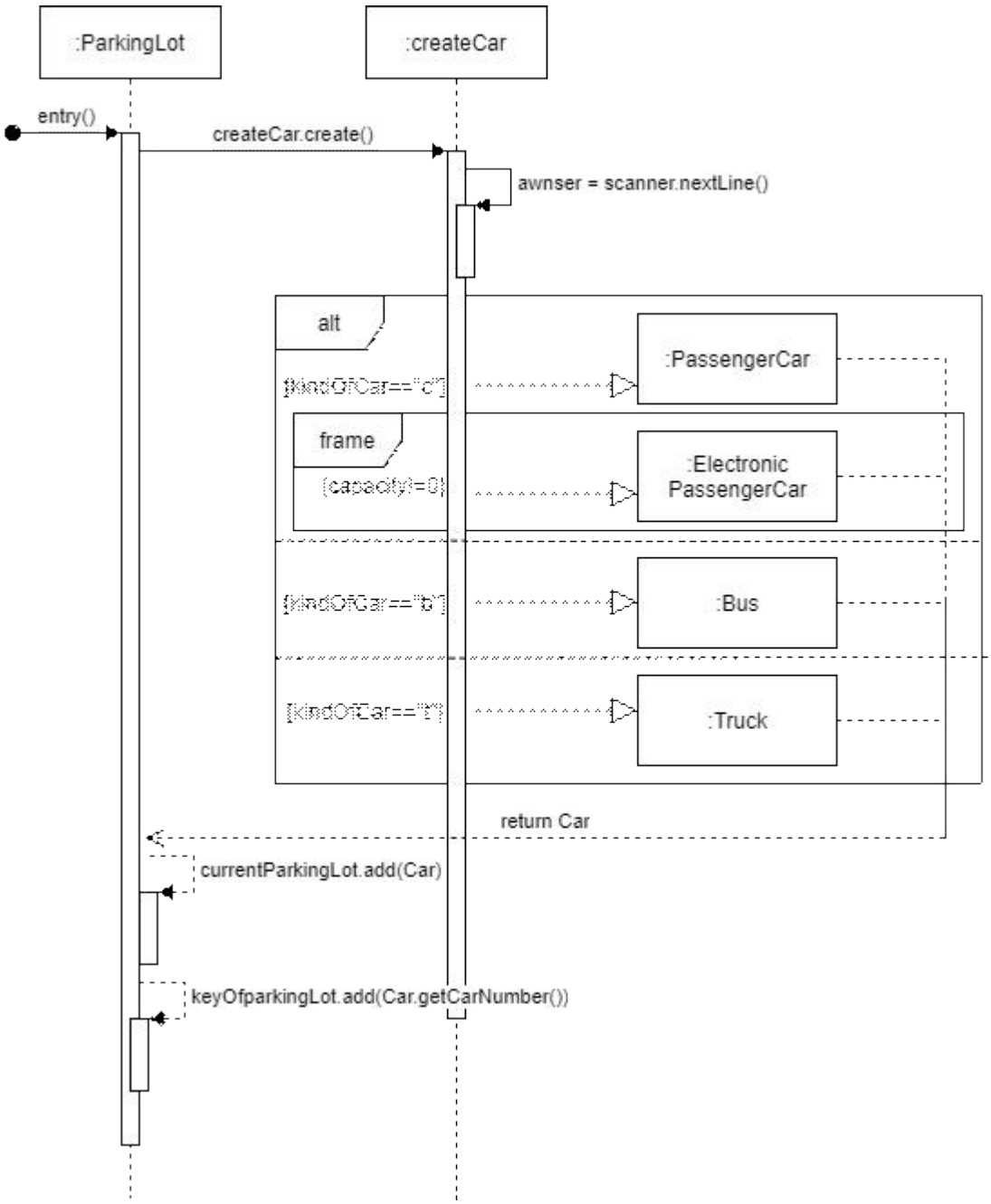
설명

- 메뉴에서 1-5번 선택에 따라 다른 활동을 보인다
- 1번을 선택 시 정보를 받아 차량을 만들고 주차장에 차량을 저장한다
- 2번을 선택 시 출차시간을 받아 주차시간과 금액을 계산 후 요금을 정산하여 출차한다
- 3번을 선택 시 주차장을 정렬하여 출력한다
- 4번을 선택 시 현재 수입을 보여준다
- 5번을 선택 시 프로그램을 종료한다.

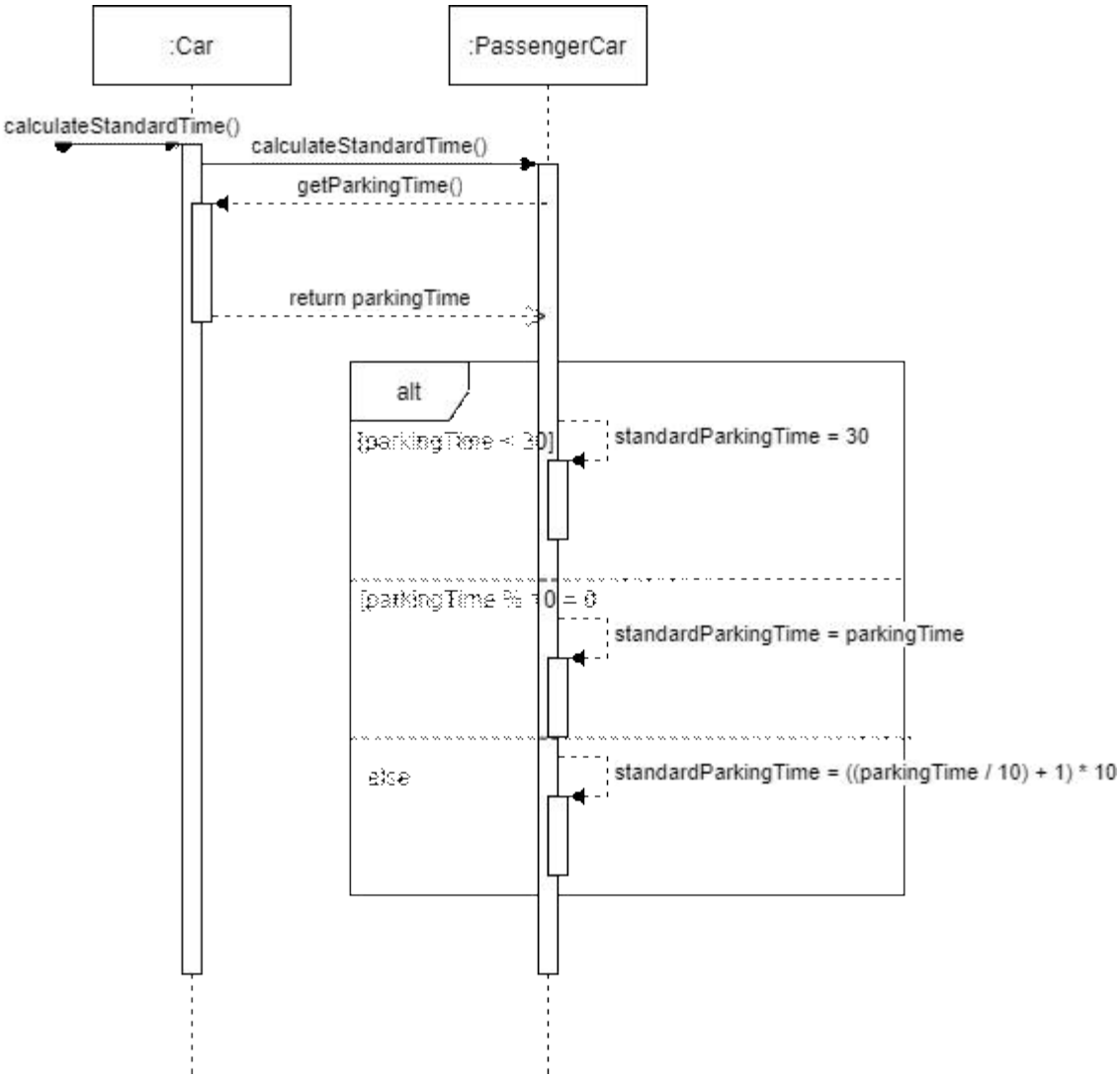
- 라) 주요 알고리즘

- sequence diagram-

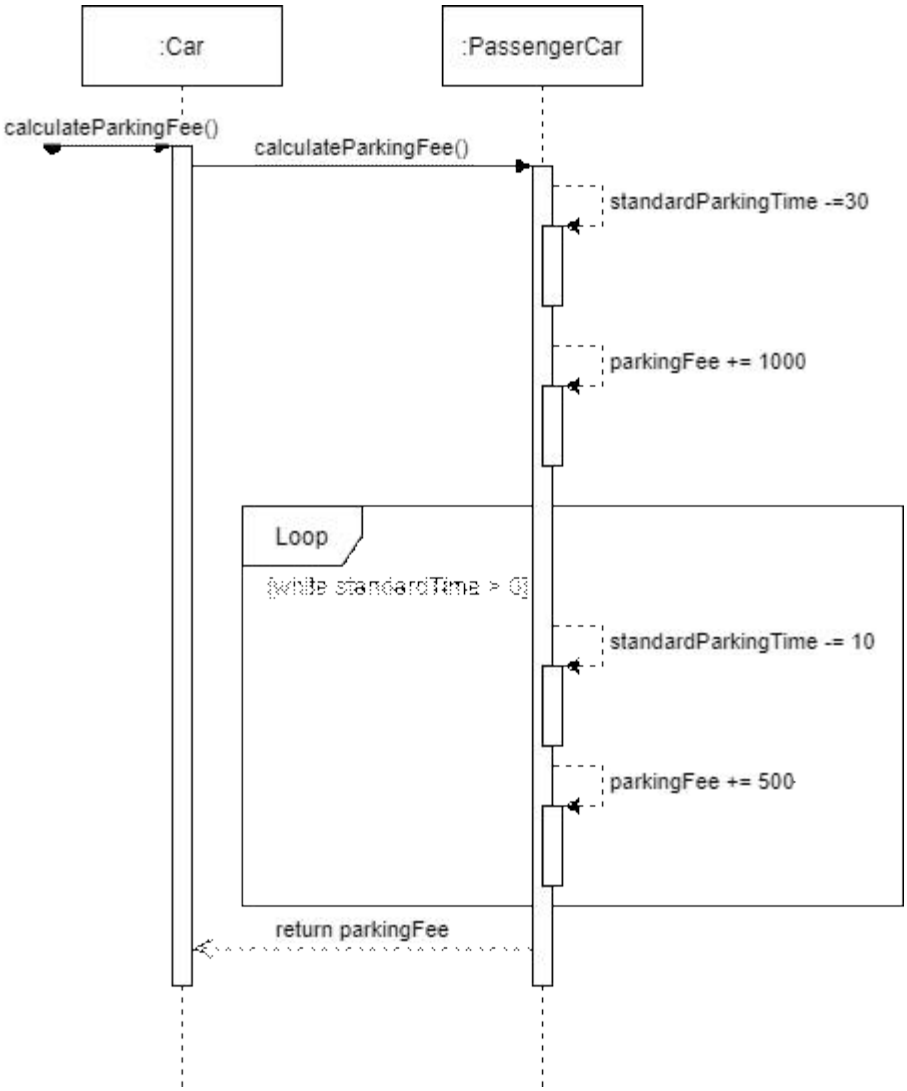
-entry()



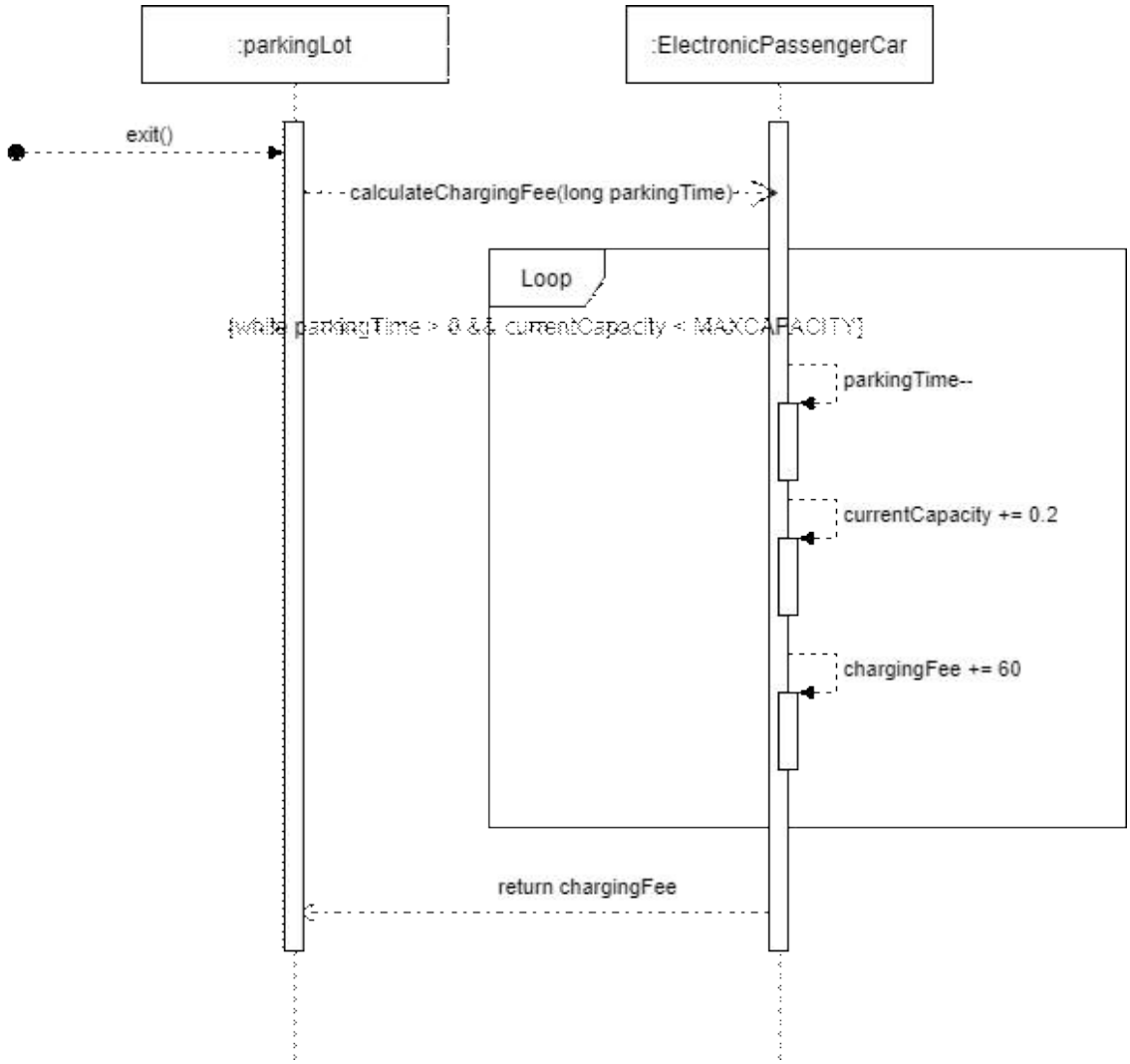
-PassengerCar.calculateStandardTime()



-PassengerCar.calculateParkingFee()



-ElectronicPassengerCar.calculateChargingFee(long parkingTime)



마) 실행결과

정상 실행 부분

```
주차장 관리 프로그램이 실행됐습니다.
주차장의 주차 최대 차량 수를 입력해주세요. : 10
주차장은 최대 10대 주차 가능합니다.
```

그림 10 주차장 사이징 기능 구현

```
4. 총 수입 보기
5. 종료
1
차량 종류 및 용량을 입력하세요! 승용차(c), 트럭(t), 버스(b)
c 10
차량 번호를 입력하세요!(4자리 숫자)
1234
입차시간을 입력하세요! (년 월 일 시 분)
2020 12 18 11 00
입차가 완료됐습니다.
```

그림 11 입차기능 구현

```
2
출차할 차량번호를 입력하세요!
1234
출차시간을 입력하세요!
2020 12 18 14 00
주차시간은 3시간 0분입니다.
주차요금은 19,300원입니다.
```

그림 12 전기차 출차 시, 충전요금과
주차요금 합산

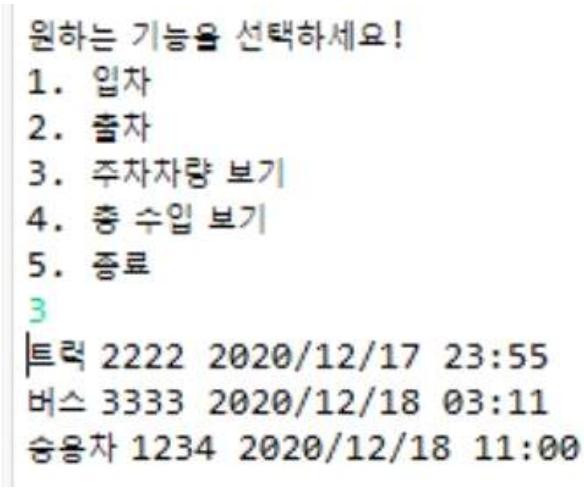


그림 13 주차장 보기 기능 구현

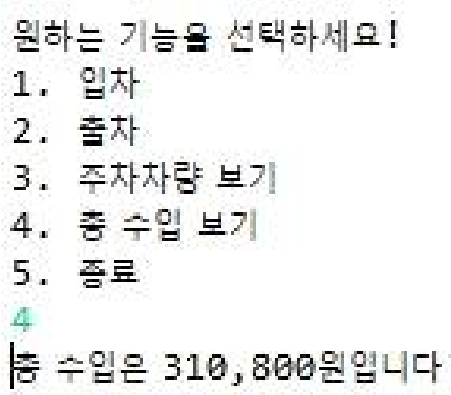


그림 14 총 수입 보기 기능 구현

Exception 핸들링 부분

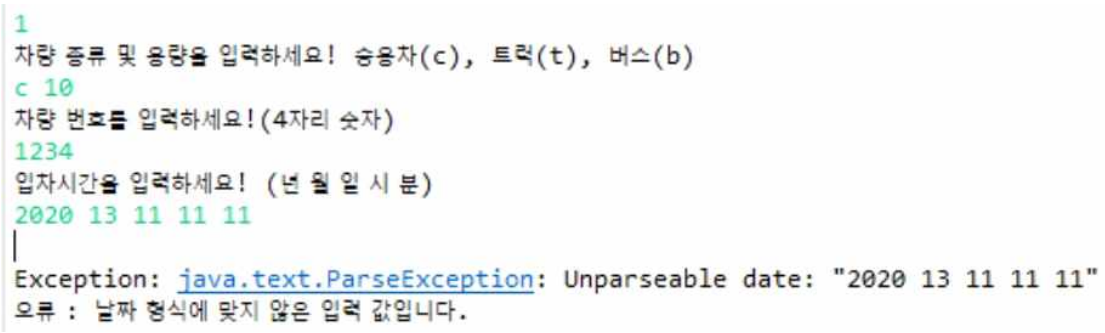


그림 15 날짜 형식 익셉션

```
원하는 기능을 선택하세요!
1. 입차
2. 출차
3. 주차차량 보기
4. 층 수입 보기
5. 종료
7
오류 : 메뉴에 존재하지 않는 번호입니다.

원하는 기능을 선택하세요!
1. 입차
2. 출차
3. 주차차량 보기
```

그림 16 메뉴선택 예외처리

```
입차가 완료됐습니다.

원하는 기능을 선택하세요!
1. 입차
2. 출차
3. 주차차량 보기
4. 층 수입 보기
5. 종료
1

Exception: java.lang.IndexOutOfBoundsException: 오류 : 주차장에 자리가 없습니다.
```

그림 17 주차장 만차 예외처리

```
1
차량 종류 및 용량을 입력하세요! 승용차(c), 트럭(t), 버스(b)
t 5
차량 번호를 입력하세요!(4자리 숫자)
12345

Exception: java.util.InputMismatchException: 오류 : 차량 번호가 올바르지 않습니다
```

그림 18 입차 시, 형식에 어긋난 차량 번호 예외

```
원하는 기능을 선택하세요!
1. 입차
2. 출차
3. 주차차량 보기
4. 중 수입 보기
5. 종료
1
차량 종류 및 용량을 입력하세요! 승용차(c), 트럭(t), 버스(b)
h 10
Exception: java.util.InputMismatchException: 오류 : 차량 종류 및 용량이 옳지 않습니다.
```

그림 19 존재하지 않는 차량 종류 선택 시 예외처리

```
차량 번호를 입력하세요!(4자리 숫자)
1234
입차시간을 입력하세요! (년 월 일 시 분)
2020 12 12 12 12
입차가 완료됐습니다.

원하는 기능을 선택하세요!
1. 입차
2. 출차
3. 주차차량 보기
4. 중 수입 보기
5. 종료
2
출차할 차량번호를 입력하세요!
3333
Exception: java.lang.IllegalArgumentException: 오류 : 출차번호가 존재하지 않습니다.
```

그림 20 존재하지 않는 출차번호 예외처리

```
2. 출차
3. 주차차량 보기
4. 중 수입 보기
5. 종료
2
출차할 차량번호를 입력하세요!
1234
출차시간을 입력하세요!
2020 1 1 1 1 1
Exception: java.lang.ArithmeticException: 오류 : 출차시간이 입차시간보다 빠릅니다.
```

그림 21 입,출차 시간 차이 예외처리

바) 결론

프로그래밍을 하며 배운 점

1. 상속과 다형성

abstract method, 상속 등을 활용하여 수업시간에 배운 상속과 다형성 부분을 활용하였다.

2. 데이터의 클래스화

Car, ParkingLot 등 중요한 데이터를 클래스화하고 캡슐화하여 외부에서 접근을 못하고 프로그램을 통해서만 다룰 수 있게 만들었다.

3. OOP를 통해 클래스 분산

저번 과제엔 한 클래스가 너무 많을 일을 하여 OOP의 특성을 활용하지 못했다. 이번엔 한 클래스가 너무 많은 짐을 지지 않도록 기능을 분산시켰고, 하지만 단순히 일을 나누는 것이 아니라 책임 분배의 원칙에 입각하여 결정하였다.

4 .Exception Handling

만약 사용자의 실수로 Exception 혹은 프로그램 버그가 나올 시 프로그램 데드가 되는 것이 아닌, 핸들링을 통해 잘못을 복구하고 계속하여 프로그램을 사용할 수 있게 하는, Exception Handling을 사용했다. 더불어 자바 라이브러리를 통해 올바를 익셉션을 찾는 법도 배웠다.

프로그래밍 시 어려웠던 점

1. 초기 설계와 달라진 최종 설계

이번 과제에서 가장 중요하고 집중한 부분은 초기 설계를 꼼꼼히 하여 최종 설계와 큰 차이가 안나게 하는 것이었다. 하지만 미처 다루지 못한 부분이 많아 수정 단계를 많이 거쳤다. 그러나, 수정 단계에서 코딩이 먼저 움직이는 것이 아닌 설계 단계로 다시 돌아가 구조를 다시 확인하는 과정을 통해 1차 과제처럼 중복된 내용이 많이 발생하거나 치중된 클래스를 만드는 것은 피했다. 하지만 단단한 초기 설계를 구축하는 것은 상당히 어렵게 다가왔다.