‘주차관리 시스템’ 프로젝트 보고서

김주희

1. 소개

1-1) 개요

본 프로젝트에서는 주차 관리 시스템을 개발하고자 했다. 이 시스템은 주차장 내의 자동차들을 효과적으로 관리하고, 거주자와 비거주자 차량에 대한 주차 공간 할당, 요금 부과, 입차, 출차 기록 등을 자동으로 처리하는데 중점을 두고 있다. 시스템은 거주자와 비거주자 간의 우선순위를 고려하여 주차 공간을 할당하고, 주차 기간에 따른 요금을 정확히 계산하여 수입을 관리한다. 또한, 차량의 입차와 출차 시에는 정확한 기록을 유지하여 주차장의 운영을 효율적으로 추적하고 관리할 수 있도록 설계하려 하였다.

해당 시스템은 관련된 기능을 캡슐화하기 위해 클래스 (RCar, Car, FunctionHandler, 및 ParkingManagementSystem)을 사용하는 객체 지향적 접근을 따르고 있다. 또한, 데이터 저장 Collection 구조로 리스트를 사용하였다. 리스트는 거주자 및 비거주자 차량의 인스턴스를 저장하는 데에 사용하였다.

1-2) 주요 클래스 및 기능 (구체적인 분석 및 설계 포함)

1-2-1) ParkingManagementSystem 클래스 및 FunctionalHandler 클래스

-ParkingManagementSystem 클래스는 프로그램의 진입점인 메인 클래스이다.

-Resident 생성자를 이용하여 객체를 초기화하고 배열에 저장하여 해당 클래스에서 ParkingManagementSystem 에서 바로 거주자에 대한 정보를 불러와서 읽고 사용할 수 있도록 하는 작업을 수행하였다.

-(a, e, x, w, s, i)과 같은 사용자 입력을 받아 각 클래스 및 기능을 호출하며, 프로그램의 흐름을 제어하도록 하였다.

-명령 ‘a’에 관련해서는 입력 중 주차 공간이 있는지 확인한 후 배정 하고자 하는 차 정보를 Car 생성자에 정보를 저장하도록 하였다. 또한 FunctionalHandler 클래스의 공간 배정 매소드 allocateParkingSpace를 호출하여 RCar 리스트에 정보를 저장하도록 설계하였다. ‘a’를 사용하여 입력 받을 때, 차종 값에 따라 RCar를 상속받는 subclass gasolineRCar, vanRCar, electricRCar의 생성자에도 정보를 저장하도록 하였다.

-명령 'w'에 관련해서는 입력 중 배정을 철회하고자 하는 차 정보를 입력하면 RCar 리스트에 차 번호 일치 여부를 확인한 후 RCar 리스트에서 해당 차 정보를 지우도록 시스템화하였다. 배정된 차량이 아닐 경우 해당 차량은 배정된 차량이 아님 문구를 출력하도록 하였다.

-명령 'e'에 관련해서는 입력 중 주차 공간이 있는지 확인한 후 입차 하고자 하는 차 정보를 Car 생성자에 정보를 저장하도록 하였다. FunctionalHandler 클래스의 입차 메소드 ParkingSpace를 호출하여 Car 리스트에 정보를 저장하도록 설계하였다. 'e'를 사용하여 입력 받을 때, 차종 값에 따라 Car를 상속받는 subclass gasolineCar, vanCar, electricCar의 생성자에도 정보를 저장하도록 하였다.

-명령 'x'에 관련해서는 입력 중 출차하고자 하는 차 정보를 입력하면 Car 리스트에 차 번호가 있는지 확인한 후 FunctionalHandler 클래스의 출차 메소드 NotParkingSpace를 호출하여 Car 리스트에서 Car 리스트에서 해당 차 정보를 지우도록 하였다. 차량 정보를 빼면서 'x' 명령을 사용하여 입력한 날짜를 출자 시간으로 정의하여 해당 차량의 주차 시간을 분 단위로 계산하였고, 분 단위 주차 시간에 따른 수입을 계산하여 출력하도록 설계하였다.

-명령 's'에 관련해서는 지금까지 입차된 차량의 List<RCar>와 List<Car>에 저장되어 있는 차들 정보를 출력하도록 설계하였다. 차량이 거주자 차량인지 확인한 후, 거주자 차량이 맞으면 거주자 차량으로, 아니면 비거주자 차량으로 각각 정보를 분류하여 출력하도록 하였다.

-명령 'i'에 관련해서는 일반 차량의 입출차 시간을 고려하여 ceiling(주차시간/10)*(입력받은 분당 요금)을 계산하여 출력하도록 설계하였다.

1-2-2) RCar 클래스

-RCar 클래스는 명령 'a'와 'w' 즉 주차 공간 배정과 철회에 관련한 기능을 수행하도록 설계하였다.

-명령 'a'에 관련해서는 입력 중 전화번호가 거주자 명단에 있는 전화번호 인지 확인한 후, 거주자에 한해 RCar 생성자에 정보를 저장하고 해당 정보의 새로운 RCar 객체를 RCar 리스트에 저장되도록 시스템화하였다.

-명령 'w'에 관련해서는 입력 중 배정을 철회하고자 하는 차 정보를 입력하면 RCar 리스트에 차 번호가 있는지 확인한 후 RCar 리스트에서 해당 차 정보를 지우도록 시스템화하였다.

-주차 공간 배정에 필요한 기본 속성(주자 차량 번호, 거주자 전화번호, 배정 날짜, 차 종류)을 정의한다.

-주차 공간 배정에 필요한 기본 속성에 대해 getter 메소드와 setter 메소드를 통해 차량의 등록 번호, 입차 시간 등을 관리하도록 설계하였다.

1-2-3) Car 클래스

-Car 클래스는 명령 'e'와 'x' 즉 입차와 출차 관련한 기능을 수행하도록 설계하였다.

-명령 'e'에 관련해서는 입력 중 주차 공간이 있는지 확인한 후 입차 하고자 하는 차 정보를 Car 생성자에 정보를 저장하고 해당 정보의 새로운 Car 객체를 Car 리스트에 저장하도록 시스템화하였다.

-명령 'x'에 관련해서는 입력 중 출차하고자 하는 차 정보를 입력하면 Car 리스트에 차 번호가 있는지 확인한 후 Car 리스트에서 해당 차 정보를 지우도록 시스템화하였다.

-차량 입출차에 필요한 기본 속성(주차 차량 번호, 배정 날짜, 차 종류)을 정의한다.

-차량 입출차에 필요한 기본 속성에 대해 getter 메소드와 setter 메소드를 통해 차량의 등록번호, 입차 시간 등을 관리하도록 설계하였다.

1-2-4) ResidentManager 클래스

-주거자 정보를 관리하는 클래스이다.

-텍스트 파일(res.txt)에서 주거자 정보를 읽어와서 2차원 배열인 'res_arr'로 반환하도록 하여 주거자 정보를 필요한 형태로 가공하도록 설계하였다.

1-3) 제대로 구현하지 못한 부분

-거주자 차량에 대한 요금 계산 부분을 제대로 구현하지 못하였다.

-일반 차량에 대한 요금 계산 부분 ceiling(주차시간/10)*(입력받은 분당 요금)을 구현하였지만, gasoline, van, electric 의 차종에 대한 요금 계산을 따로 구현하지 못하였다.

-현재 주차되어 있는 차량을 보여주는 기능 's' 에 대해 각각 거주자 우선 주차 차량과 일반 차량에 대해 차량 번호와 배정 날짜, 배정 시간이 출력되긴 하지만 출력 양식을 제대로 구현하지 못하였다.

-입력받은 날짜에 대해 유효성 검사를 진행하지 못하였다.

2. inheritance polymorphism 및 exception

2-1) inheritance polymorphism 및 이 적용되는 부분

2-1-1) RCar를 상속받는 subclass gasolineRCar, vanRCar, electricRCar

-carNumber, phoneNumber, currentYear, currentMonth, currentDay, vehicleType와 같은 공통 속성을 포함한다.

-메서드 오버로딩: gasolineRCar, vanRCar, electricRCar 클래스에서 getVehicleType()이 getdisplacement() 등과 함께 오버로딩되어 차량 유형에 대한 자세한 정보를 제공한다.

-컬렉션 및 리스트: List<RCar> 사용은 이러한 리스트에 다양한 유형의 차량을 저장함으로써 다형성을 나타낸다.

-인스턴스 변수 및 메서드: gasolineRCar, vanRCar, electricRCar 및 RCar 클래스는 인스턴스 변수(속성) 및 해당 클래스의 인스턴스에 작용하는 메서드를 포함한다.

2-1-2) Car를 상속받는 subclass gasolineCar, vanCar, electricCar

-carNumber, phoneNumber, currentYear, currentMonth, currentDay, vehicleType와 같은 공통 속성을 포함한다.

-메서드 오버로딩: gasolineCar, vanCar, electricCar 클래스에서 getVehicleType()이 getdisplacement() 등과 함께 오버로딩되어 차량 유형에 대한 자세한 정보를 제공한다.

-컬렉션 및 리스트: List<Car> 사용은 이러한 리스트에 다양한 유형의 차량을 저장함으로써 다형성을 나타낸다.

-인스턴스 변수 및 메서드: gasolineCar, vanCar, electricCar 및 Car 클래스는 인스턴스 변수(속성) 및 해당 클래스의 인스턴스에 작용하는 메서드를 포함한다.

2-2) exception 처리가 적용되는 대표적인 부분

2-2-1) ResidentManager 클래스의 FileNotFoundException

BufferedReader가 생성될 때 사용된 FileReader에서 지정한 파일("res.txt")을 찾을 수 없는 경우(FileNotFoundException이 발생하는 경우) 이 부분에서 해당 예외를 처리한다. 예외가 발생하면 콘솔에 "파일명을 찾을 수 없습니다."라는 메시지를 출력한다.

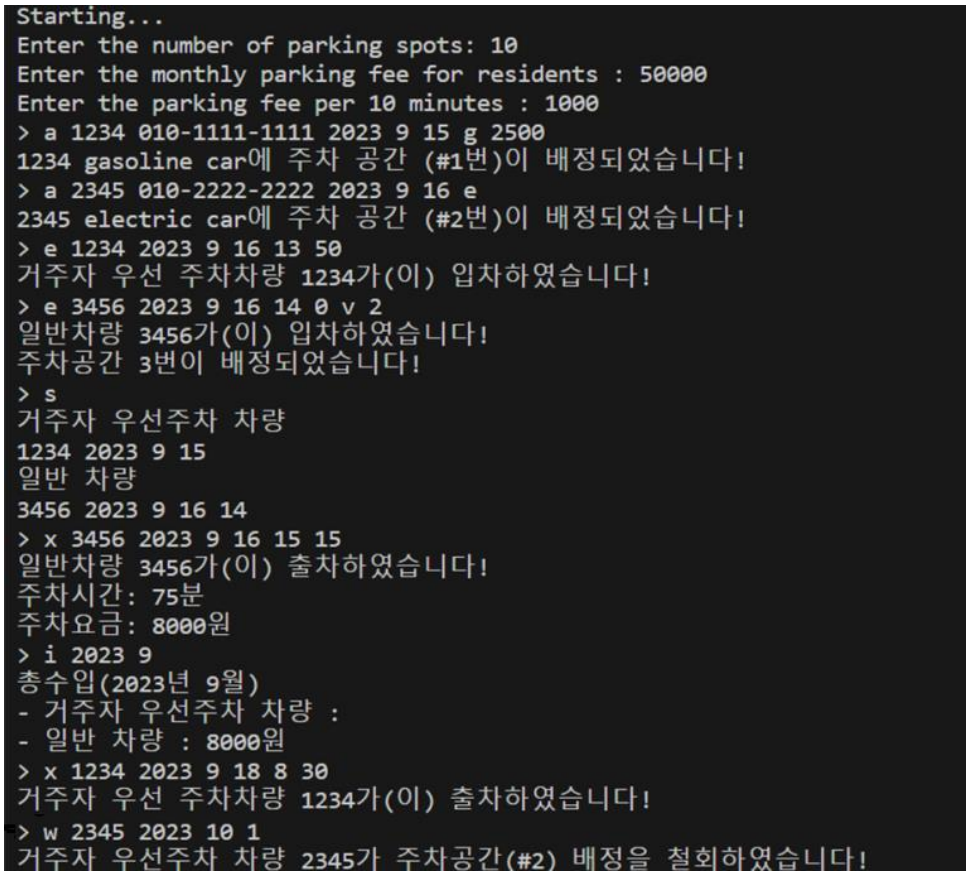
2-2-2) ResidentManager 클래스의 NullPointerException

BufferedReader에서 readLine을 호출하고, 이 때 파일의 끝에 도달하여 더 이상 읽을 내용이 없는 경우(null이 반환되는 경우) 이 부분에서 해당 예외를 처리한다. 예외가 발생하면 콘솔에 "Err : "와 예외 정보를 함께 출력한다.

2-2-3) ParkingManagementSystem 클래스의 IOException

try 블록 안에서 getTxt() 메서드와 관련된 코드들이 실행된다. 이 메서드는 파일에서 데이터를 읽어와 처리하는데, 파일 읽기 도중 발생할 수 있는 IOException이 발생할 가능성이 있다. 따라서 이를 처리하기 위해 try-catch 블록을 사용하고, 예외가 발생하면 catch 블록이 실행된다.

3. 실행결과 및 설명



-명령 'a'에 관련해서 입력 중 주차 공간이 있는지 확인한 후 배정 하고자 하는 차 정보를 Car 생성자에 정보를 저장하도록 하였다. "1234 gasoline car에 주차 공간 (#1번)이 배정되었습니다!"와 "2345 electric car에 주차 공간 (#2번)이 배정되었습니다!" 출력 문구를 통해 거주자 차량인 1234

차량과 2345 차량이 주차 공간에 잘 배정되었음을 확인할 수 있다.

-명령 'e'에 관련해서는 입력 중 주차 공간이 있는지 확인한 후 입차 하고자 하는 차 정보를 Car 생성자에 정보를 저장하도록 하였다. 거주자 차량이 1234 차량은 "거주자 우선 주차차량 1234가(이) 입차하였습니다!" 출력 문구를 통해, 일반 차량 3456 차량은 "일반차량 3456가(이) 입차하였습니다! 주차공간 3번이 배정되었습니다!" 출력 문구를 통해 입차가 잘 되었음을 확인할 수 있다.

-명령 's'에 관련해서는 지금까지 입차된 차량의 List<RCar>와 List<Car>에 저장되어 있는 차들 정보를 출력하도록 하였다. "거주자 우선주차 차량: 1234 2023 9 15, 일반 차량: 3456 2023 9 16 14" 출력 문구를 통해 거주자 우선주차 차량과 일반 차량을 구분하여 리스트에 잘 저장하고 있음을 확인할 수 있다.

-명령 'x'에 관련해서는 입력 중 출차하고자 하는 차 정보를 입력하면 Car 리스트에 차 번호가 있는지 확인한 후 정보를 지우도록 하였다. "일반차량 3456가(이) 출차하였습니다! 주차시간: 75분 주차요금: 8000원" 출력 문구를 통해 리스트에서 해당 차량이 잘 지워졌고, 주차 시간과 주차 요금이 잘 출력된 것을 확인할 수 있다.

-명령 'w'에 관련해서는 입력 중 배정을 철회하고자 하는 차 정보를 입력하면 RCar 리스트에 차 번호 일치 여부를 확인한 후 RCar 리스트에서 해당 차 정보를 지우도록 하였다. "거주자 우선주차 차량 2345가 주차공간(#2) 배정을 철회하였습니다!" 출력 문구를 통해 리스트에서 정보가 잘 지워졌음을 확인할 수 있다.

-명령 'i'에 관련해서는 일반 차량의 입출차 시간을 고려하여 주차요금을 출력하고 있음을 확인할 수 있다.

4. 배운 점과 느낀 점

상속 및 다형성 부분이 어려우면서도 배운 점이 많은 것 같다. 클래스 간에 계층 구조를 구축하여 코드 재사용을 촉진하는 상속의 개념이 사용되었다. 메서드 오버라이딩을 통해 다형성이 나타난다. 같은 메서드를 여러 클래스에서 다르게 구현하여 다양한 유형의 객체를 공통적으로 처리할 수 있음을 배웠다. 다음 번에 코드를 작성할 땐 더 효율적으로 할 수 있으리란 확신이 들었고 내가 어떤 방향으로 더 공부해야 할지 알 수 있었다.