학번: 2015110227

이름: 주희수

**프로그램 설명서:**

**[car.py]Car 클래스:** Fleet를 구성하는 자동차 클래스이므로 자동차에 속하는 속성과 메소드로 구성

변수:

* car\_id: 자동차 객체를 식별하기 위한 이름
* brand: 자동자 브랜드: Sonata, Tucson 등
* max\_fuel: 최대 연료량
* fe\_org: 기본 연비이며, 실제 연비는 계산 과정을 통해 산출
* max\_cap: 최대 승객
* fuel: 현재 연료량
* self.fe\_100: 최근 100km 간 연비이며 값이 달라질 경우 최근 연비부터 100분율을 계산하여 적용
* cap: 현재 승객
* mlg: 운행 거리이며 중고차의 경우 50000에서 시작
* used: 중고 여부이며 렌탈되지 않은 상태에서도 차량을 운행하면 중고로 전환
* rented: 해당 차가 이미 대여한 상태인지 식별

메소드:

* fill\_up(): 최대 연료량까지 기름을 채워줌
* fill(amt): amt 만큼 연료를 채워줌
* ride(num): num 만큼 인원이 차에 탑승
* drop(num): num 만큼 차에서 인원이 빠져나감
* drop\_all(): 차량을 반환하는 경우에 모든 승객이 차에서 빠져나감
* get\_range(): 현재 연료량과 연비로 주행 가능한 거리를 반환
* get\_cur\_fe(): *연비 - ((현재 탑승/최대 탑승) \* 연비 \* 0.1)* 식으로 실제 연비 계산
* move(dist): dist만큼 주행하면서 *거리 / 실제 연비* 식으로 연료 소비
* set\_fe\_100(): 최근 100km 간 연비를 계산  
  가장 최근 주행거리부터 100km될때까지 연비와 (거리/100)를 구해 연비의 변화량 반영

**[car.py]Sonata 클래스, Bongo 클래스, Tesla\_S 클래스:** Car 클래스의 자손 클래스이므로 동일한 변수로 초기화

변수: Car 클래스와 동일

메소드:

* show\_dash(): 운행거리, 연료량, 주행 가능거리, 100km 주행 간 연비, 탑승인원을 출력
* show\_state(): 렌탈 시 차량의 정보를 볼 수 있도록 차종, 연비 등의 기본 정보 출력

**[car.py]Tucson 클래스:** Car 클래스의 자손 클래스이므로 동일한 변수로 초기화되지만 적재량이 새로 추가

변수:

* max\_load: 최대 적재 무게
* load: 현재 적재 무게

메소드:

* ride, drop, drop\_all, get\_cur\_fe 모두 부모 클래스와 동일한 역할을 하지만 load라는 새로운 파라미터가 추가되어 계산 방식에 반영

**[fleet.py]Fleet 클래스:** Car 클래스를 모아둔 composition 형태의 클래스

변수:

* cars: Car 클래스로 이루어진 배열이므로 모든 렌탈 차를 여기서 관리

메소드:

* save\_fleet(): fleet 클래스의 cars 배열에 있는 모든 객체를 car\_data.json 파일로 저장
* load\_fleet(): json 파일에 있는 데이터를 클래스의 \_\_init\_\_() 함수를 통해 변환시켜 불러옴
* add\_car(car\_id, brand, used): 새로운 차량을 만들어 배열에 추가
* del\_car(car\_id): 식별자로 차량을 배열 내에서 찾아 삭제
* get\_car(car\_id): 식별자로 차량을 배열 내에서 반환(차량 운행, 주유 등의 작업에 사용)
* get\_rented\_week(): 7일 전부터 어제까지 렌탈된 차량들을 반환
* show\_all(): fleet에 있는 모든 차량 목록을 출력
* show\_available(): 대여하지 않은 할당 가능한 차량 목록 출력
* show\_rented(): 대여한 차량 목록 출력
* get\_best\_match(): 차종, 최대 연료량, 연비, 마일리지, 중고 여부, 최대 탑승 인원, 적재 무게 등의 기준을 선택적으로 제공하여 해당될 경우 각 차량마다 점수를 매겨 가장 높은 점수를 가진 차량을 사용자에게 추천하는 방식으로 작동

**[customer.py]Customer 클래스:** 고객에 대한 클래스로 이름과 현재 렌탈된 차량의 식별자를 가지고 있음

변수:

* user\_name: 사용자 이름
* rented\_cars: 해당 사용자가 렌탈한 차량 배열

메소드:

* get\_rental\_period(car\_id): 해당 사용자가 주어진 식별자의 차량을 대여하고 있는지 확인하고 만약 참이면 가장 최근의 대여 이벤트를 찾아 가장 최근의 반환 이벤트와 날짜를 비교해 대여 기간 출력

**[customer.py]Customers 클래스:** 고객 클래스로 구성되며 각 객체를 배열로 관리하는 역할 수행

변수:

* users: 사용자 객체로 이루어진 배열
* parent: 상위 클래스의 레퍼런스를 가지고 있으므로 상위 클래스를 통해 fleet 클래스에 접근하여 자동차에 대한 작업을 수행할 수 있음

메소드:

* save\_clients(): 현재 Customers 클래스의 고객 배열에 있는 각각 객체를 dict로 json에 저장
* load\_clients(): json에 저장된 dict 데이터를 클래스 init으로 변환시켜 가져옴
* create\_user(user\_name): 새로운 사용자 혹은 고객을 추가
* delete\_user(user\_name): 파라미터로 전달된 이름을 가진 Customer 객체를 삭제
* show\_users(): 모든 사용자 목록과 각각 렌털하고 있는 차량 목록 출력
* get\_user(user\_name): 입력된 이름을 가진 사용자 객체를 반환
* rent\_car(user\_name, car\_id): fleet에서 자동차를 해당 고객에게 할당하여 렌탈 작업 수행  
  이미 렌탈된 경우 에러 메시지 발생
* return\_car(user\_name, car\_id): Customer 객체의 렌탈 목록에서 해당 자동차를 삭제하고 다시 반환하여 다른 사용자가 렌탈할 수 있도록 rented 값을 false로 변경

**[event.py]Event 클래스:** 기록되는 이벤트를 클래스 형태로 구현

변수:

* date: 이벤트 발생 일시
* car\_id: 이벤트가 어떤 Car 객체에 대해 발생했는지 식별자 저장
* event\_type: 이벤트가 어떤 이벤트인지 저장

**[event.py]Fuel\_Event, Move\_Event, Rental\_Event, Return\_Event, Add\_Event, Delete\_Event 클래스:**

위 이벤트 클래스는 모두 Event 클래스를 부모로 생성된 자손 클래스이며 fuel 이벤트의 경우 주유량, move 이벤트의 경우 주행거리, 연비 등 각각 다른 변수를 가짐

**[logger.py]Event\_List 클래스:** 위에서 언급된 모든 Event 클래스의 자손 클래스를 log\_events라는 배열에 저장하여 필요한 범위의 데이터를 반환하는 역할 수행

변수:

* file\_name: 이벤트를 json 파일로 저장할 때 사용하는 파일이름
* log\_events: 주유, 주행, 추가, 제거, 대여, 반환 등의 모든 이벤트로 구성되는 배열

메소드:   
해당 클래스의 모든 메소드는 클래스 메소드이므로 객체를 생성하지 않고도 다른 패키지에서 호출 가능

* save\_events(): 이벤트를 json 파일로 저장
* load\_events(): json 파일로 저장된 이벤트를 각각 클래스를 통해 불러옴
* get\_car\_events(car\_id, event\_type): 이벤트 유형과 자동차 식별자를 파라미터로 제공하면 해당되는 이벤트만 배열로 반환
* get\_car\_events\_week(car\_id, event\_type): 지난 7일부터 어제까지의 이벤트 중에서 자동차와 이벤트 유형에 해당되는 것만 배열로 전달
* get\_last\_car\_event(car\_id, event\_type): 자동차 식별자와 이벤트 유형으로 찾은 이벤트 목록에서 가장 최근 발생한 이벤트 반환
* get\_events(event\_type): 유형에 해당하는 이벤트 목록을 전체 반환
* get\_events\_week(event\_type): 지난 일주일 간 발생한 특정한 유형의 이벤트만 반환
* get\_events\_range(event\_type, from\_date, to\_date): 시작과 끝에 해당되는 날짜를 이벤트 유형과 같이 전달해 해당 구간에 있는 이벤트만 배열로 반환
* get\_user\_events(user, event\_type): 사용자에 해당하는 특정 유형의 이벤트만 반환
* get\_user\_and\_car\_events(user, car\_id, event\_type): 사용자와 자동차 및 유형이 모두 충족하는 이벤트만 반환
* get\_last\_user\_and\_car\_event(user, car\_id, event\_type): 사용자와 자동차 및 이벤트 유형이 맞는 이벤트 중에서 가장 최근에 발생한 것만 반환
* convert\_to\_date(str): string으로 받은 날짜를 datetime 형식으로 반환해주는 메소드
* get\_json\_now(): 이벤트 발생 일시를 기록할 때 json 형태로 반환해주는 메소드
* get\_today(): 오늘 날짜를 년도-월-일 형식으로 반환하는 메소드
* get\_date\_prior(days): days만큼 전의 날짜를 출력해주는 메소드
* json\_deserialize(d\_str): json으로 저장되면서 datetime 객체가 인식되지 않아 deserialize하는 메소드
* add\_event(event\_class): log\_events 배열에 새로운 이벤트 객체를 추가하는 역할

**[monitor.py]Monitor 클래스:** 각종 현황 정보를 확인하기 위한 메소드를 제공하는 클래스

변수:

* parent: 상위 클래스의 레퍼런스로 자동차 클래스에 접근

메소드:

* show\_rented\_event\_count\_week(event\_type): 일주일 간 대여한 차량에서 발생한 주유 및 주행 거리 출력
* show\_rented\_event\_count\_all(event\_type): 전체 기간에서 대여한 차량에서 발생한 주유 및 주행 거리 출력
* show\_rented\_event\_count\_range(event\_type): 특정 구간에서 대여한 차량에서 발생한 주유 및 주행 거리 출력
* show\_ event\_count\_week(event\_type): 일주일 간 발생한 이벤트의 개수 출력
* show\_ event\_count\_all(event\_type): 전체 기간에서 발생한 이벤트의 개수 출력
* show\_event\_count\_range(event\_type): 특정 구간에서 발생한 이벤트의 개수 출력
* show\_ event\_car\_week(event\_type): 일주일 간 특정 이벤트가 발생한 차량 목록 출력
* show\_ event\_car\_all(event\_type): 전체 기간에서 특정 이벤트가 발생한 차량 목록 출력
* show\_event\_car\_range(event\_type): 특정 구간 동안 이벤트가 발생한 차량 목록 출력

**[rental.py]Rental 클래스:** 모든 패키지를 임포트시켜 jupyter notebook에서 임포트할 경우 해당 파일만 임포트하여 실행 가능

메소드:

* \_\_init\_\_(): fleet, customers, monitor, event\_list에 해당되는 객체를 생성하고 동시에 모든 데이터를 불러옴
* save(): fleet, customers, event\_list에 해당하는 모든 객체를 json 형태로 변환시켜 저장