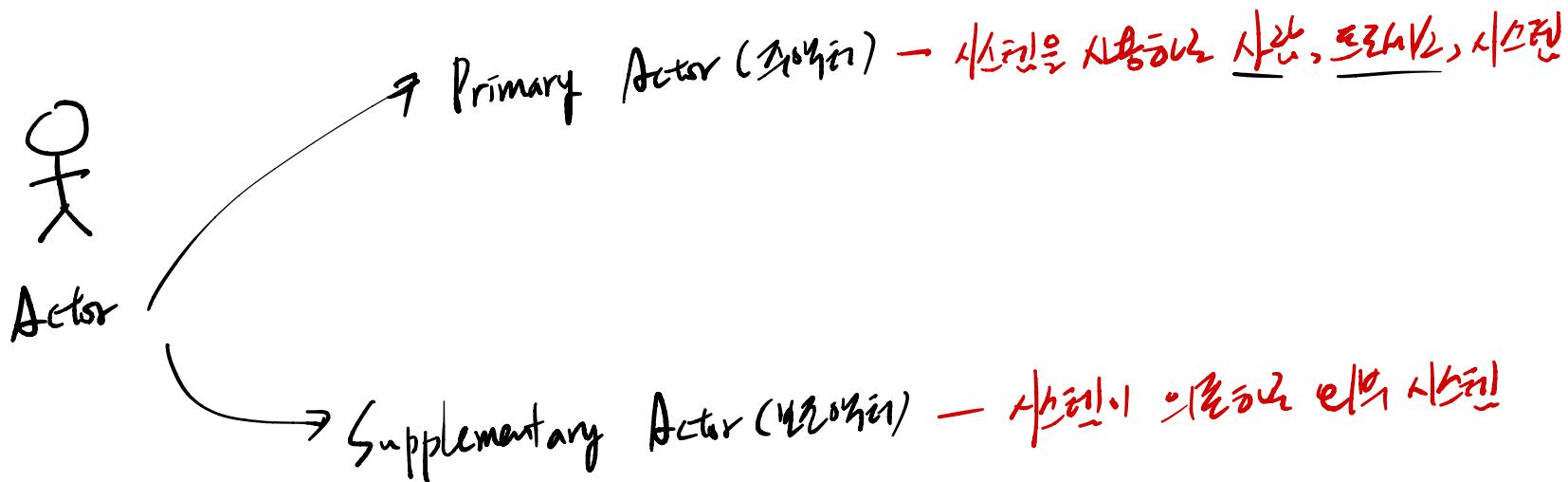


* Use-case 분석

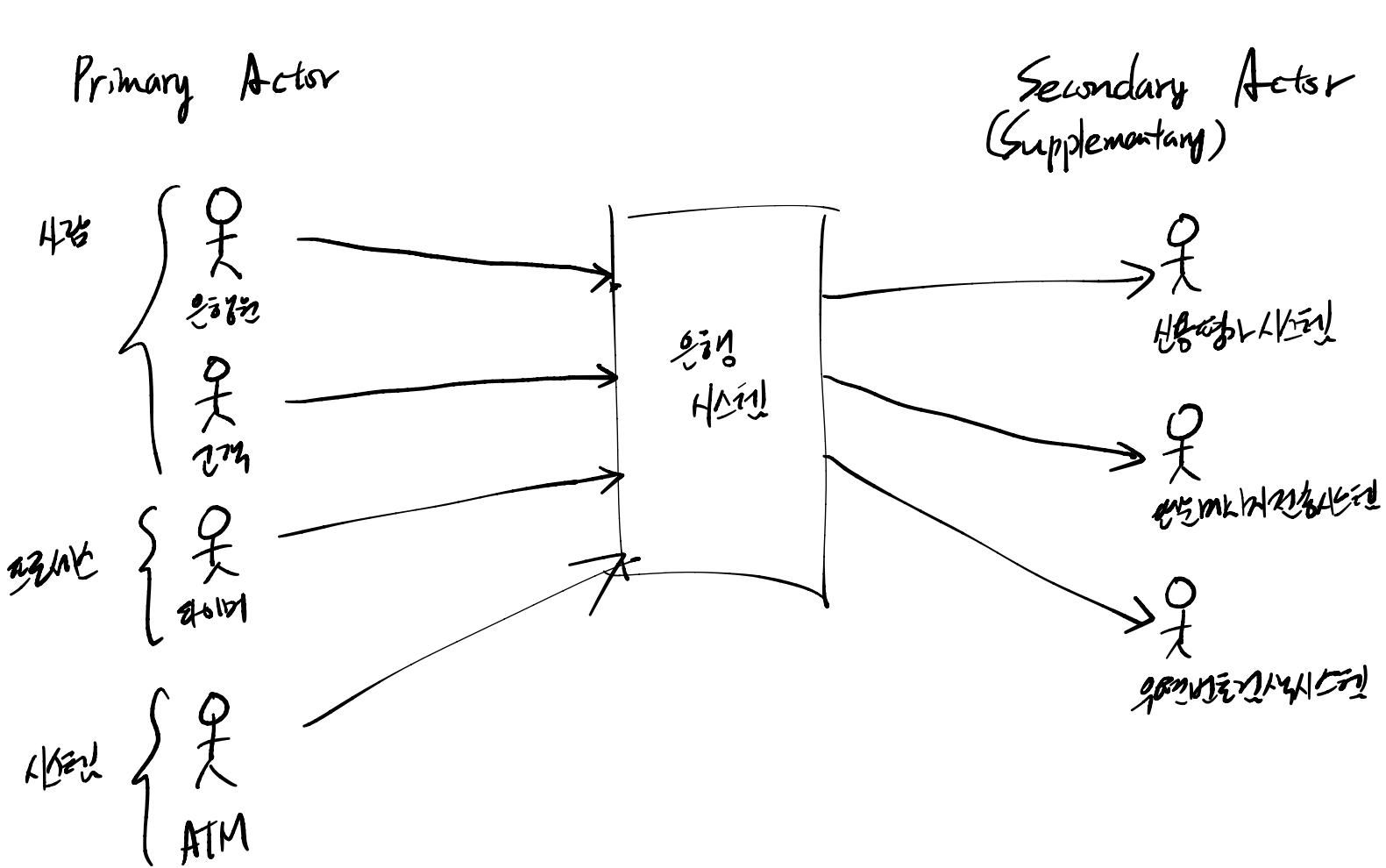
↳ 시스템의 사용 가능성을 명시화

- ① Actor 분석
↳ 시스템을 사용하는 사람, 프로그램
- ② Use-case 분석
↳ Actor가 시스템을 통해 얻을 수 있는 일부 목적
- ③ Use-case diagram
↳ Actor가 Use-case를 수행하는 경로
 - 설계도면
 - 기능구현도
 - 데이터구조도
 - 처리구조도
 - 에러구조도
 - 설정도면

① Actor 티입



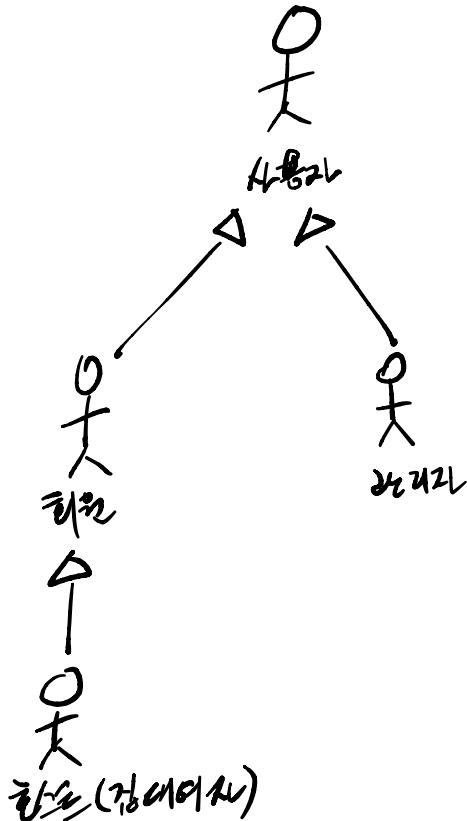
Primary Actor



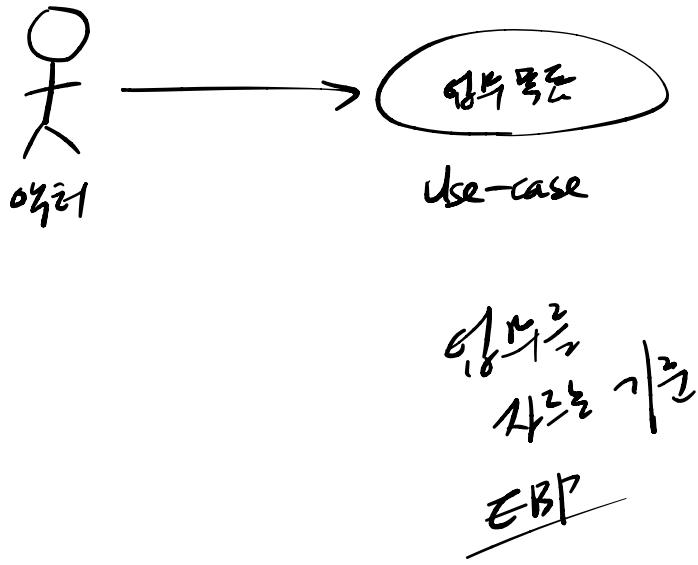
Secondary Actor (Supplementary)

* 여러의 성을 갖기

여러의 부인 성을 갖기



* Use-case 히트

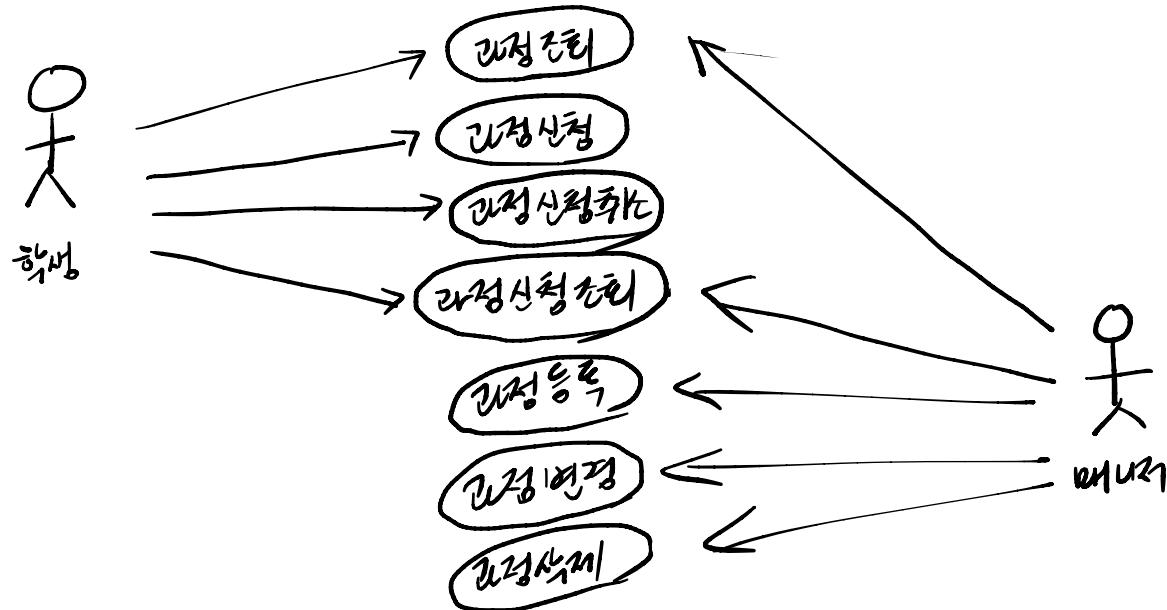


시작할 사이드케이스!

↳ 개별 관리하기에 적합한 크기로
작별하는 방법
(2쪽 ~ 4쪽 기간 동안 개별화를 초기)

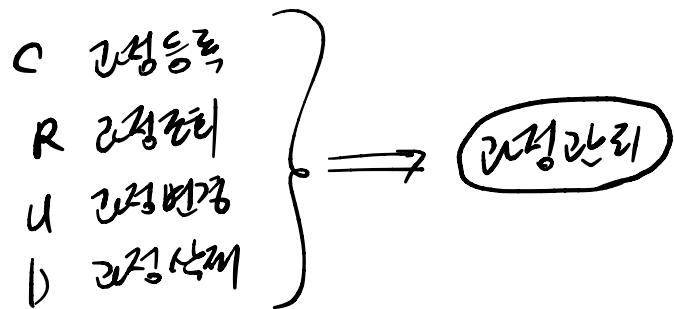
- } ① 한 사람이 한번에 수행하는 임무
한 번에 하나의 한 번 수행
- ② 시스템을 사용해서 처리하는 경우
- ③ 카운트 가능한 임무
↳ 시작과 끝이 명확

a)

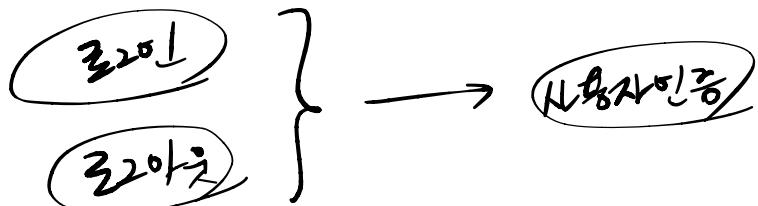


* Use-case 키워드 사용하기 II

① CRUD의 행동들을 각각의 키워드 Use-case를 합친것이 처리하기 좋다
↳ 2~4개로 단위로 쪼개기

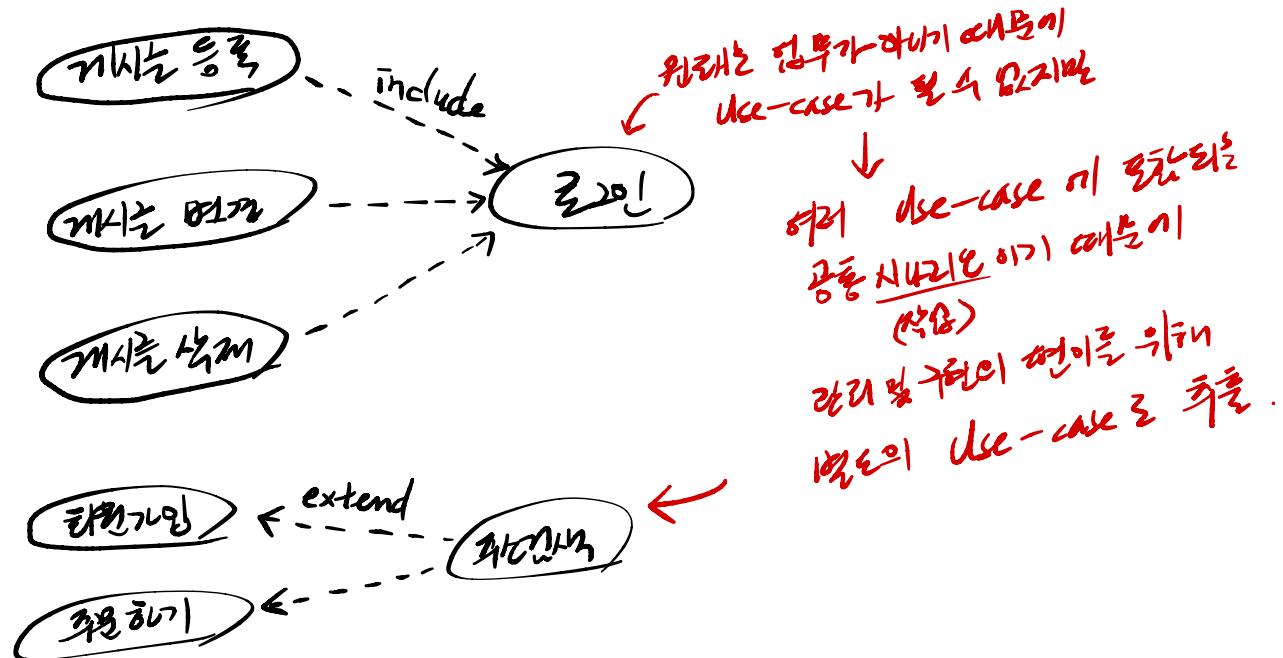


② 서로 연관된 Use-case의 경우 같은 키워드 Use-case를 구성화.

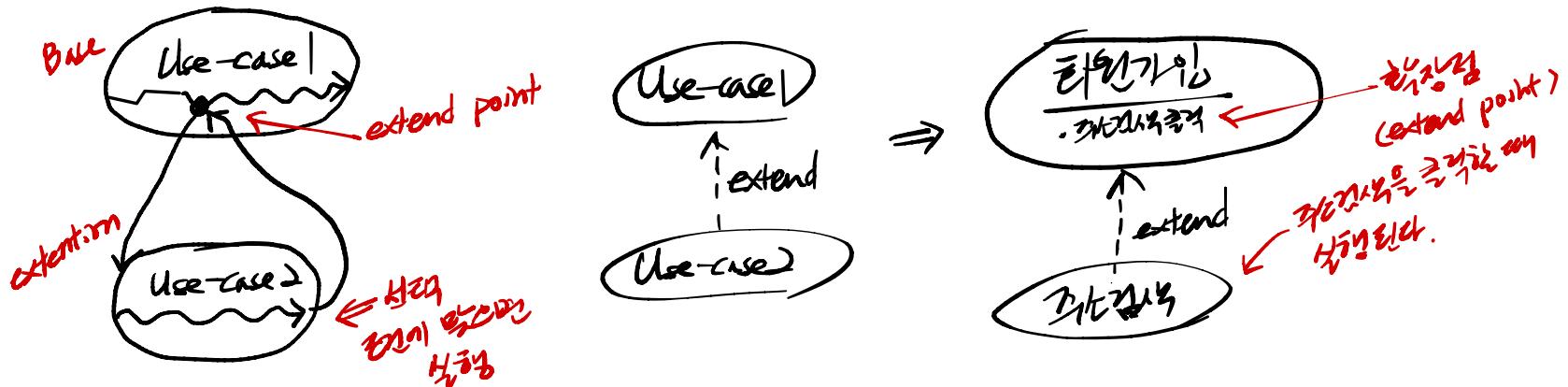
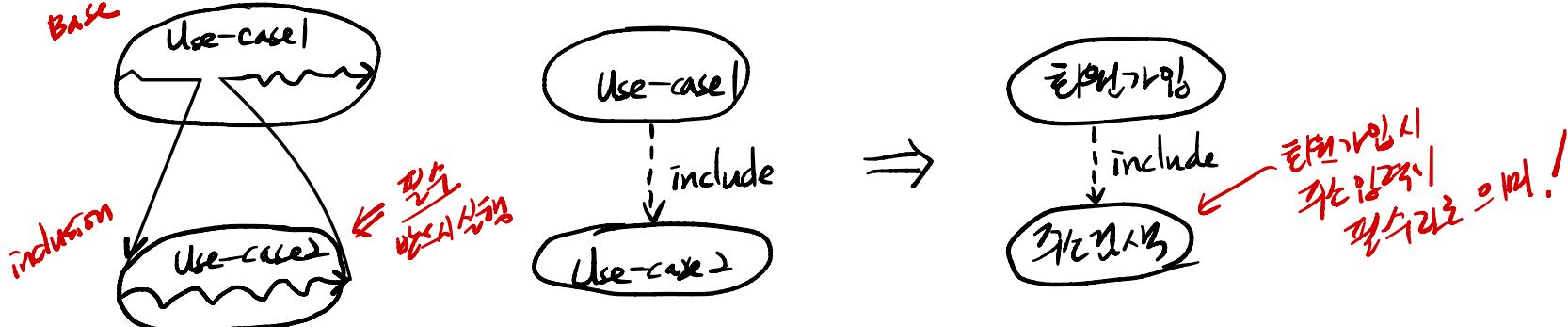


* Use-case 티어 구조화하기 II

③ 여러 Use-case의 종속되는 시나리오가 있는 경우의 Use-case 구조

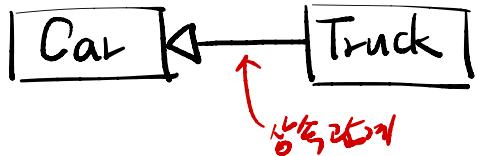


* Use-case의 include와 extend



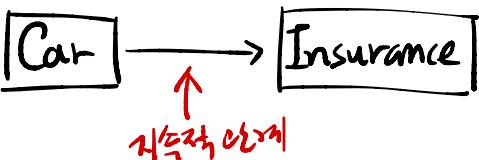
* 클래스 다이어그램 - 클래스 관계

① 상속



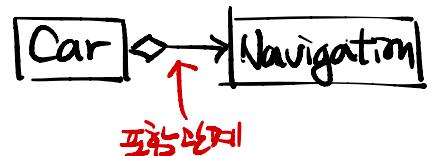
```
class Truck < extends Car {  
}
```

② 연관(Association)



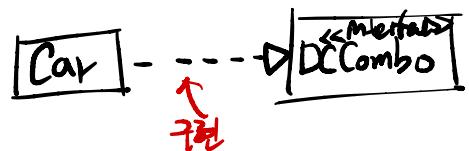
```
class Car {  
    Insurance 네임;  
}
```

③ 집합(Aggregation)



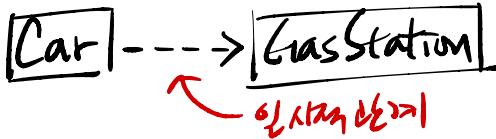
```
class Car {  
    Navigation 네임;  
}  
Car != Navigation
```

⑥ 구현(Realization)



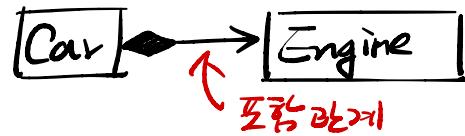
```
class Car implements DCCombo {  
} ==
```

⑤ 의존(Dependency)



```
class Car {  
    void fuelUp(GasStation gs) {  
    } ==  
}
```

④ 핍성(Composition)



```
class Car {  
    Engine 예지;  
}  
Car = Navigation
```