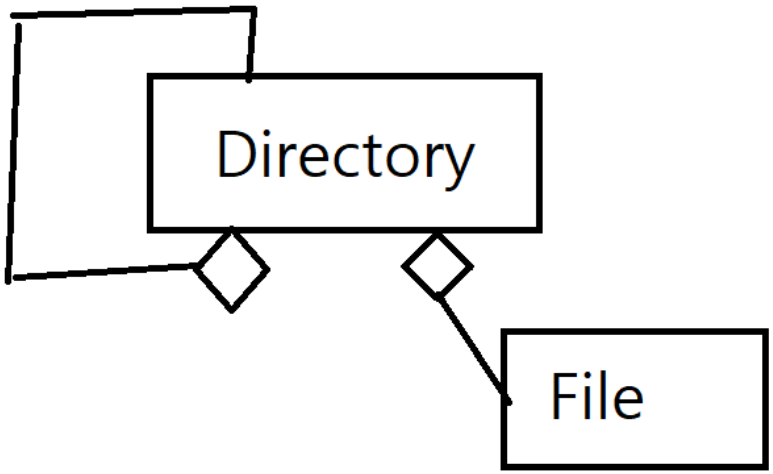
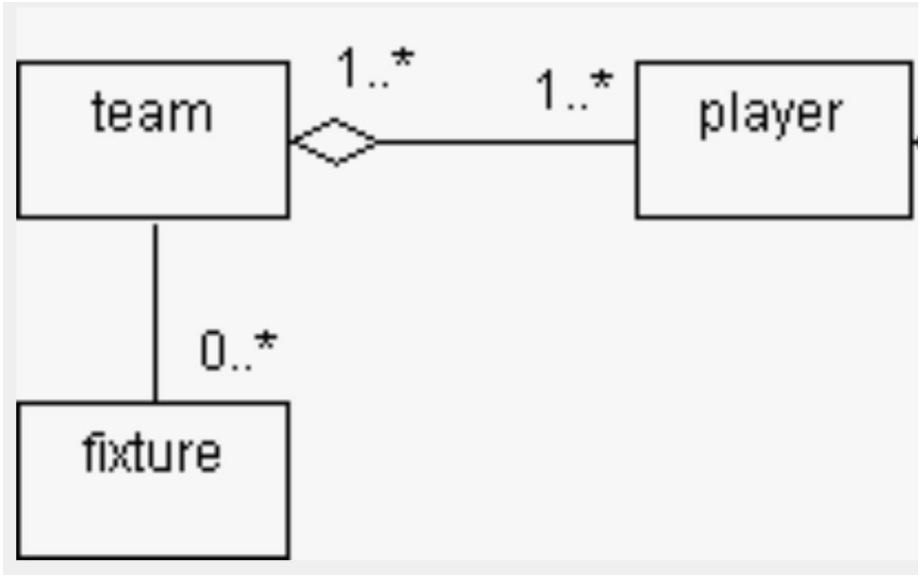
관계의 표현

- **집합(aggregation)** ➔ 공유 집합(shared aggregation: aggregation)
  과 구성 집합 (composite aggregation: composition) 으로 세분





aggregation

관계의 표현

- **합성**(composite aggregation: composition)
  - 어떤 클래스가 다른 클래스의 모임으로 구성
  - 예> 자동차와 부품과의 관계

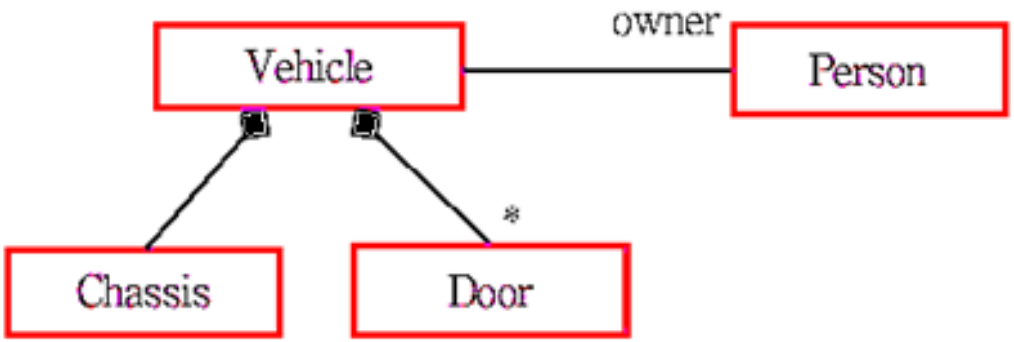
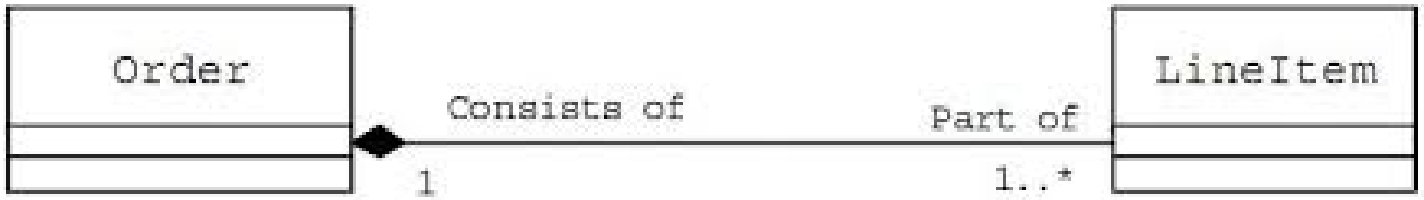
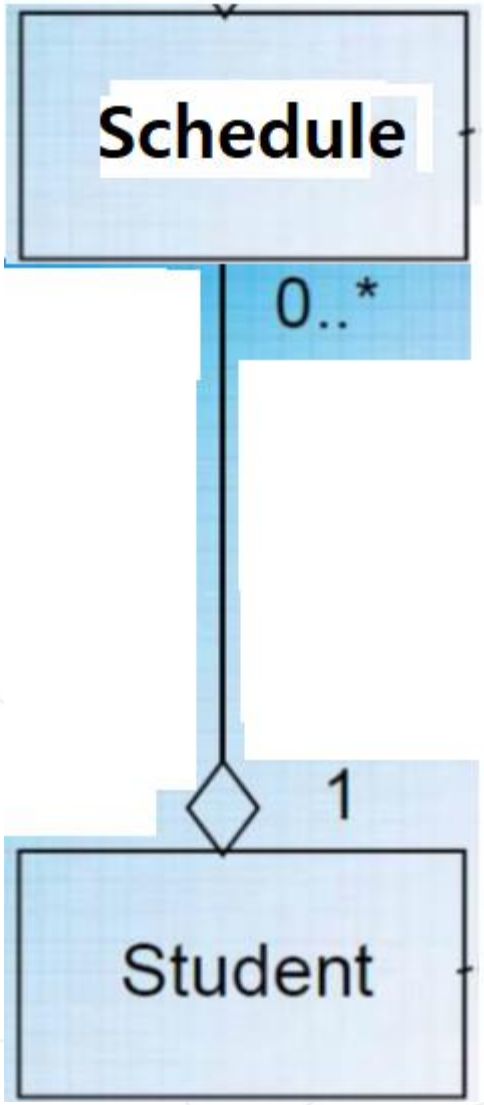


그림 5.18 ▶ 전체 부분 관계의 예



The Composite Relationship



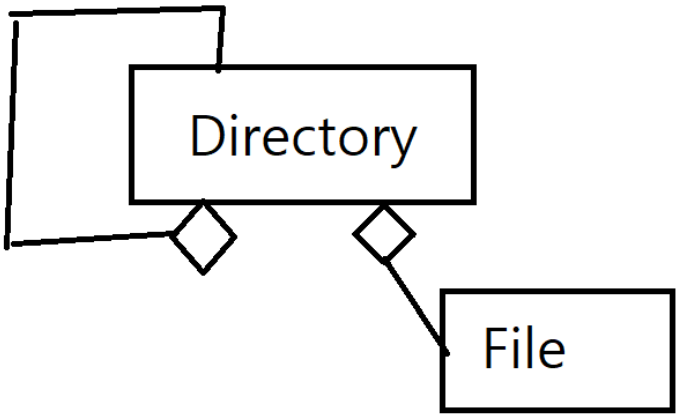

```
class Schedule
{
  public     Schedule() { }
  private    Student theStudent;
}

importjava.util.Vector;
classStudent
{
  public    Student() { }
  private   Vector   theSchedule;
}
```

관계의 표현

- **집합 관계에서 전파현상(propagation)**

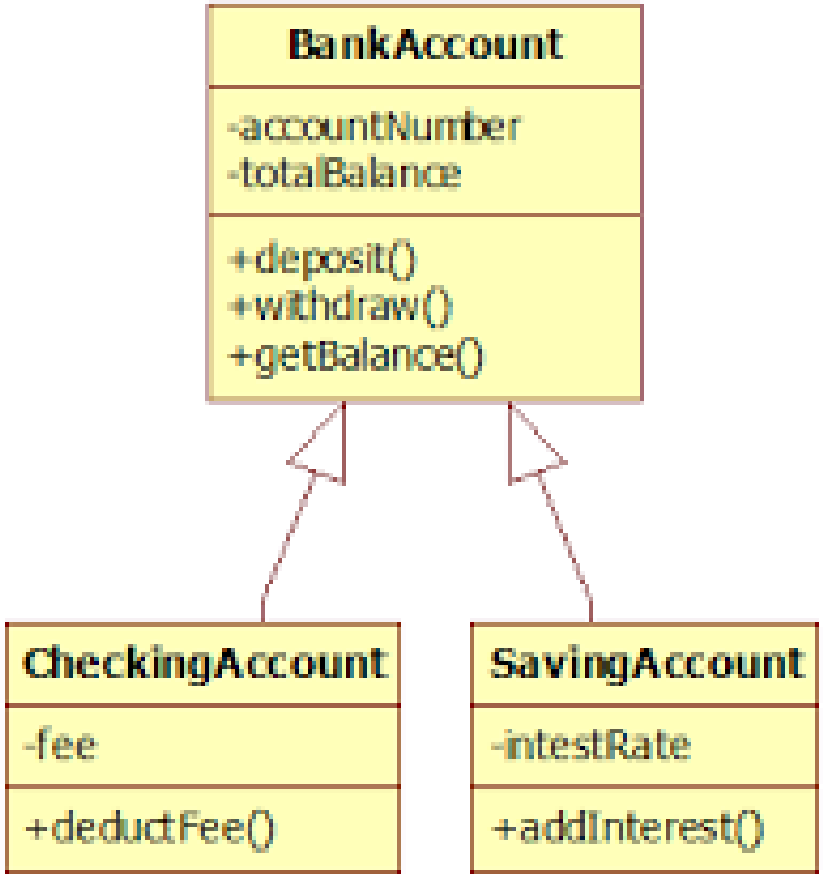
  - 전체 클래스의 오퍼레이션을 수행하는데, 부분 클래스의 오퍼레이션 을 수행하는 되는 현상.

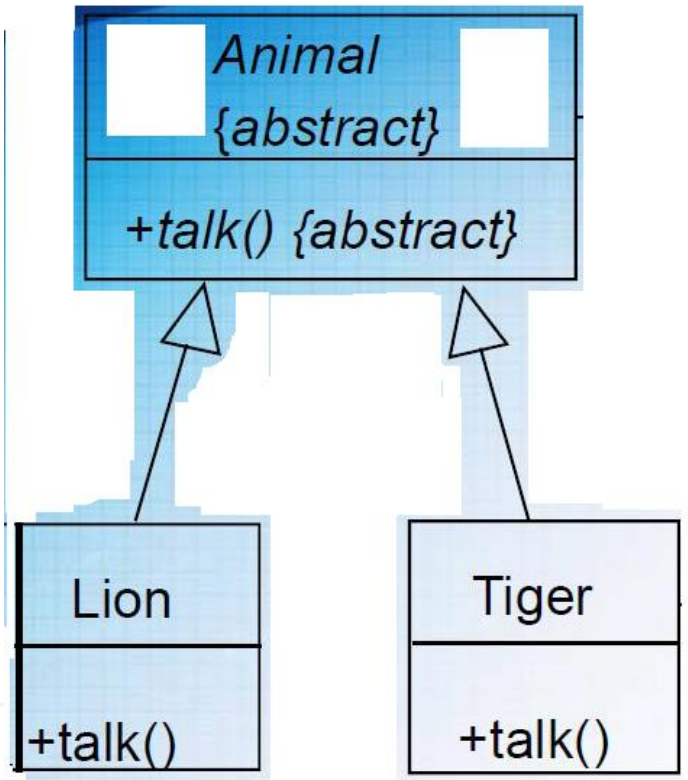
  - 예) Directory 의 size는 어떻게 구하겠는가?



- **상속(inheritance)**
  - **일반화된 개념의 클래스와 더 구체적인 개념의 클래스 사이의 관계**
  - **일반화(generalization)**

상속(inheritance)

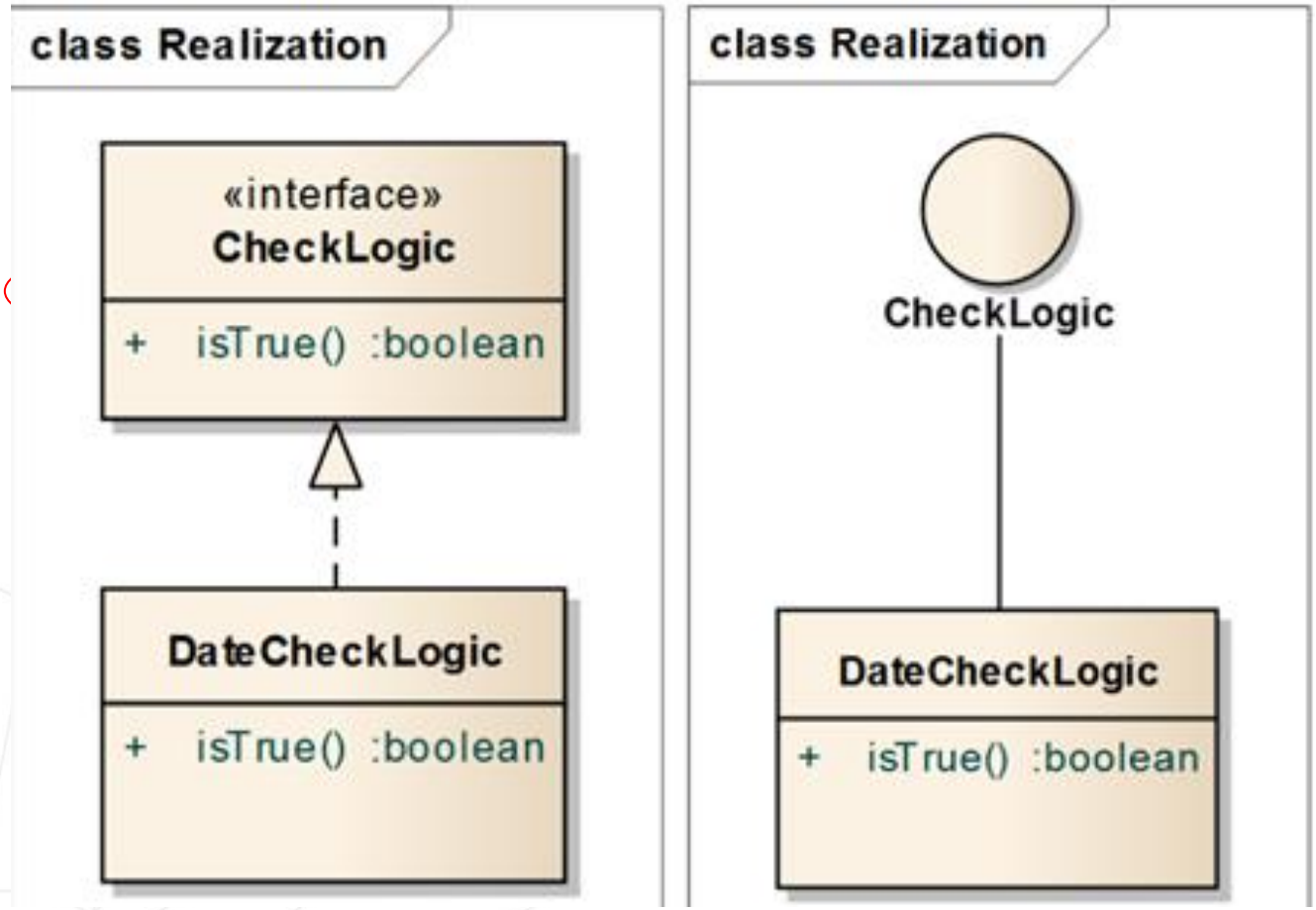





```
abstract class Animal
{
  public abstract void talk();
}
```

```
class Tiger extends Animal
{
  public Tiger() { }
  public void talk() { }
}
```

인터페이스 realization



```java
public interface CheckLogic {

    /**
     * 체크로직을 수행하여 true/false를 리턴
     * @return
     */
    public boolean isTrue();
}
```

```java
public class DateCheckLogic implements CheckLogic {

    @Override
    public boolean isTrue() {
        // 날짜와 관련된 체크로직 수행
        return true;
    }
}
```

클래스 다이어그램 작성 과정

- **Step 1** 클래스가 될 만한 후보를 파악
- **Step 2** 가장 중요한 클래스를 시작으로 연관, 상속, 속성을 추가
- **Step 3** 클래스의 주요 임무(responsibility)를 찾아내어 오퍼레이션으로 추가

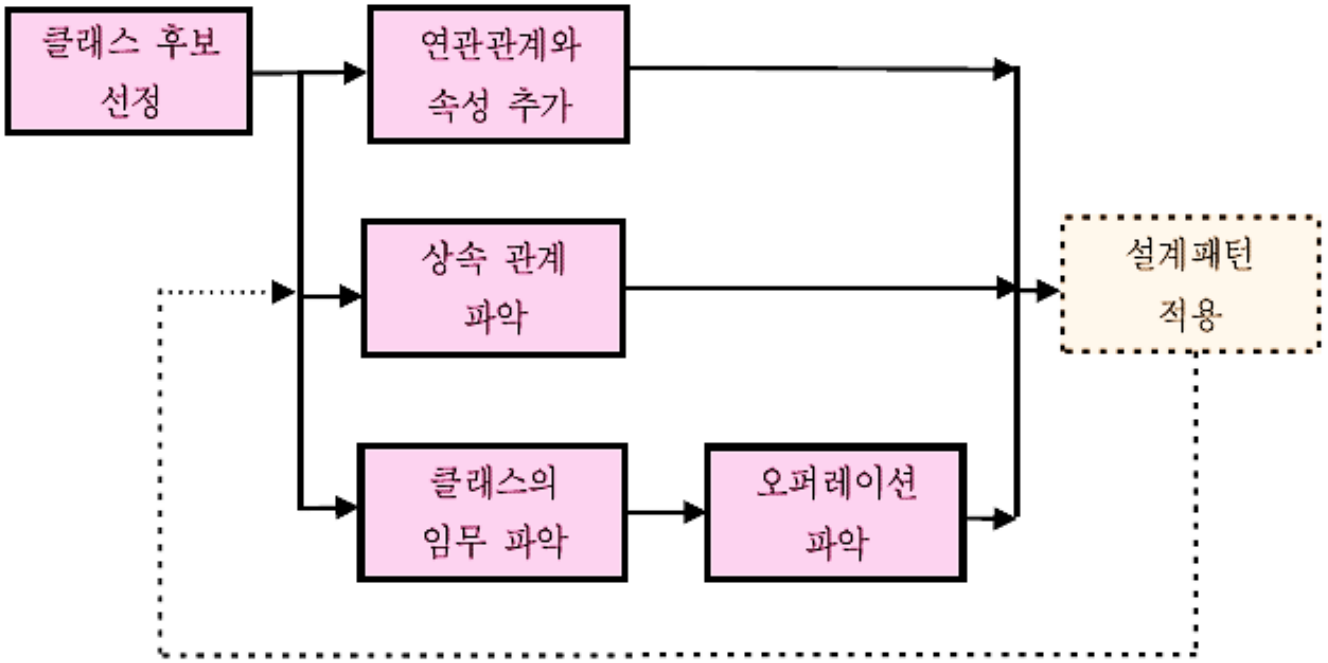


그림 5.20 ▶ 클래스 다이어그램 작성 과정

클래스 찾기

- 도메인 개념, 즉 사용 사례로부터 클래스가 될 만한 것을 찾는 일

- 클래스가 될 수 있는 요소
  - 구조
  - 외부 시스템
  - 디바이스
  - 역할
  - 운용 절차
  - 장소
  - 조직
  - 완성된 시스템에 의하여 조작되어야 할 정보

클래스 찾기

- **엔티티 클래스 찾기 <== 영구적으로 저장되어 사용될 자료 보관하는 역할**
  - 사용 사례를 이해하기 위하여 사용자와 개발자가 명확히 규정한 용어
  - 사용 사례에서 반복되어 나오는 용어 (예를 들면 Video Tape)
  - 시스템이 계속 추적하여야 하는 실세계의 엔티티 (예를 들면 Rental, Title)
  - 자료 저장소 또는 단말(예를 들어 Scanner)
  - 자주 사용하는 응용 도메인의 용어(예를 들어 Customer)

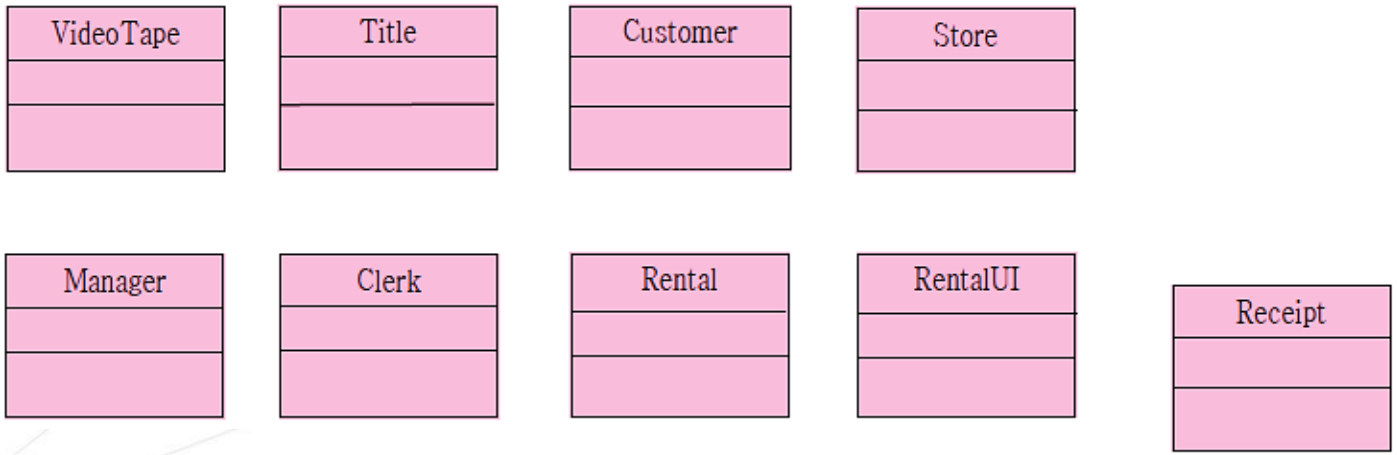


그림 5.20 ▶ 비디오 대여점을 위한 클래스 후보

클래스 찾기

- 경계 클래스 찾기 ==> 시스템 외부의 액터 와 상호작용하는 클래스
  - 사용자가 자료를 시스템에 입력하기 위하여 필요한 양식과 윈도우를 찾음 (예를 들면 RentalUI, ReportRental)
  - 시스템이 사용자에게 반응하는 메시지나 알림을 찾음(예를 들어 PendingRentalNotice)
  - 인터페이스가 어떻게 보이는지는 경계 객체에 모형화 하지 않음
  - 인터페이스를 나타내는 사용자 언어는 구현 기술과 관련 없는 용어 사용

클래스 찾기

- **제어 클래스 찾기**
  - 사용 사례가 복잡하여 소규모의 이벤트로 분할해야 한다면 하나 이상의 사용 사례 당 1개의 제어 클래스를 찾음
  - 사용 사례에서 액터 하나 당 하나의 제어 클래스를 찾음
  - 제어 클래스는 사용 사례 또는 사용자 세션 안에서만 유효. 제어 클래스가 활성되는 시점과 끝이 명확하지 않다면 사용 사례가 명확히 파악되지 못한 것

연관 찾기

- 연관(association) :
  - 어떤 클래스의 인스턴스가 작업을 수행하기 위하여 다른 클래스를 알아야 함

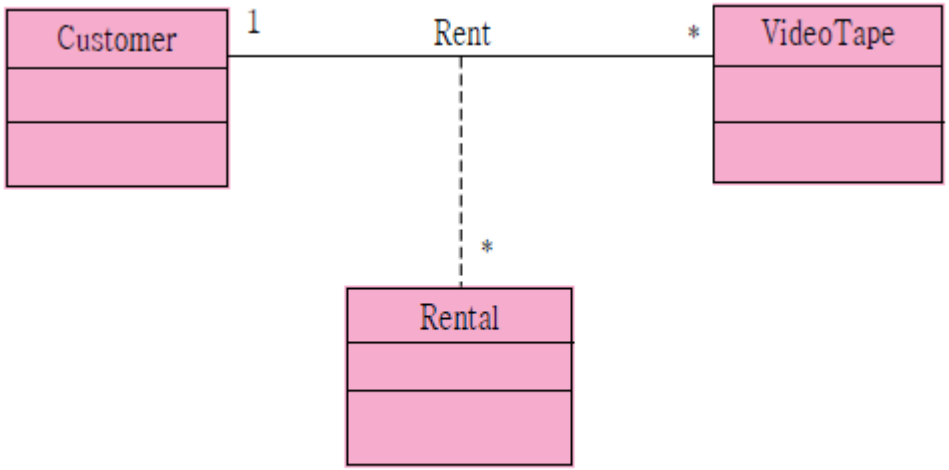


그림 5.21 ▶ Customer와 VideoTape 클래스 사이의 연관 관계

- 연관의 속성
  - **이름** - 두 클래스 사이의 연관 관계를 나타냄
  - **역할** - 연관 관계의 양쪽 끝에 있는 클래스의 기능을 나타냄
  - **다중도** - 연관 관계를 구성하는 인스턴스의 개수

연관 찾기

- 비디오 대여 시스템에 대한 클래스들의 관계

동사구 => 연관관계

다중도는 나중에

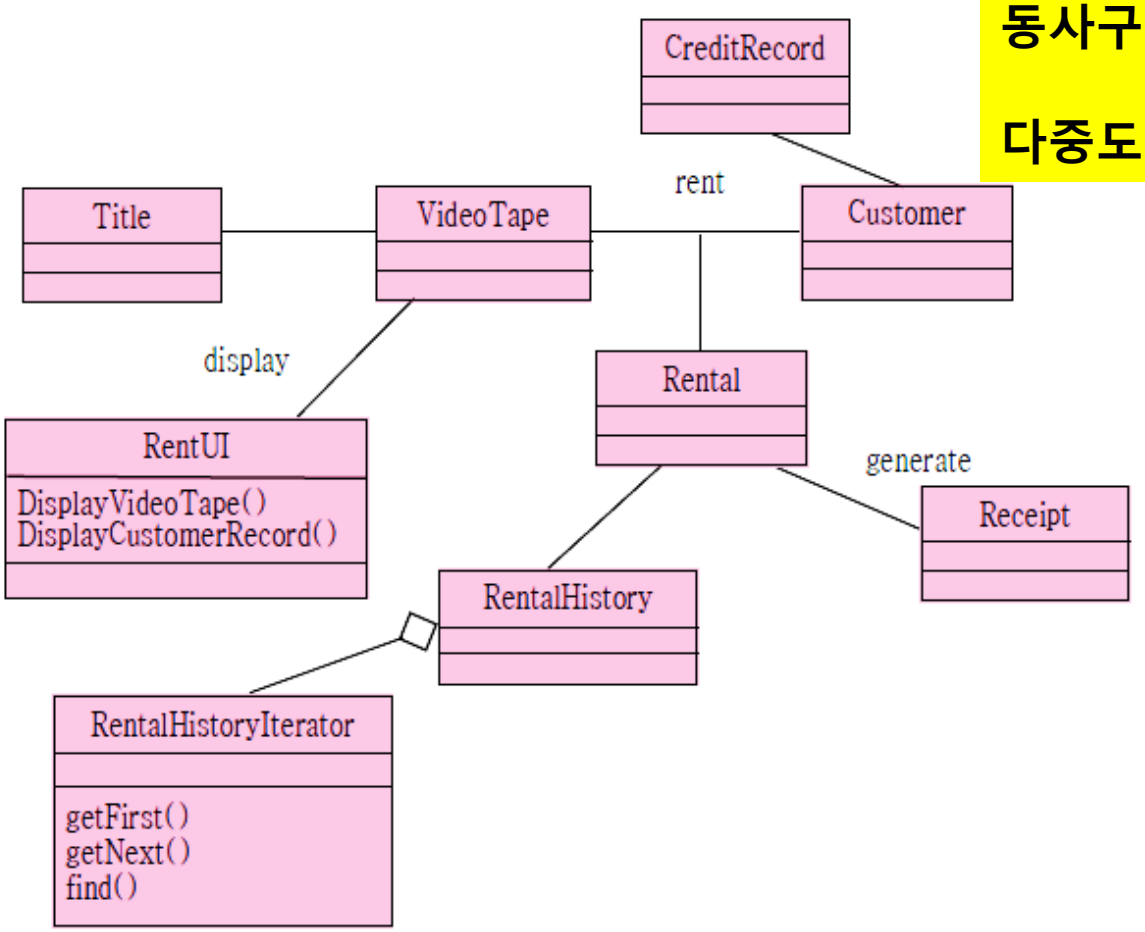


그림 5.23 ▶ 비디오 대여점의 클래스 다이어그램

속성 추가

- **속성 : 개별 객체들이 가지는 특성**

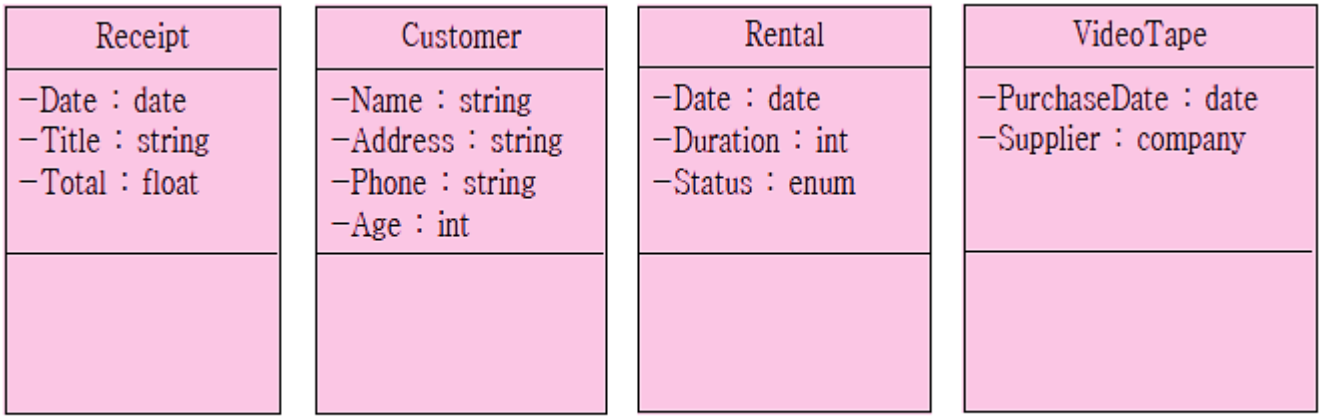


그림 5.24 ▶ 클래스의 속성

- **속성의 요소**
  - **이름** - 객체 안에서 구별할 수 있는 속성의 이름. 예를 들어 VideoTape은 PurchaseDate와 Supplier 속성을 가짐. PurchaseDate은 테이프를 구매한 날짜이며 Supplier는 비디오 공급업체를 나타냄
  - **간단한 설명** - 구현하는 프로그래머를 위하여 간단히 설명을 첨가
  - **속성값의 타입** - 예를 들어 Name 속성은 스트링. 또한 Status는 열거형으로 rentable, rented, returned라는 값을 가질 수 있음

클래스 다이어그램 작성

- 항공권 예매 시스템 클래스 다이어그램 작성

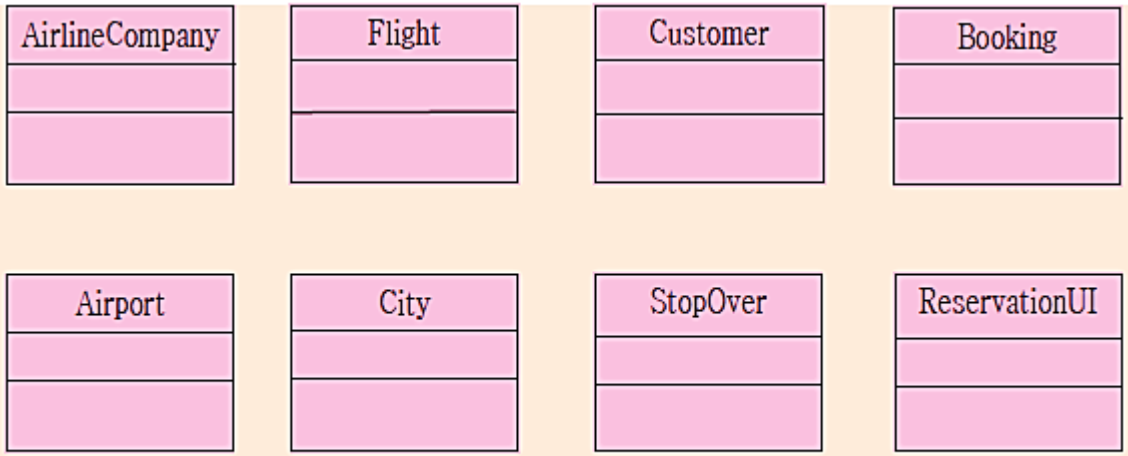


그림 5.25 ▸ 항공권 예약 시스템의 클래스 후보

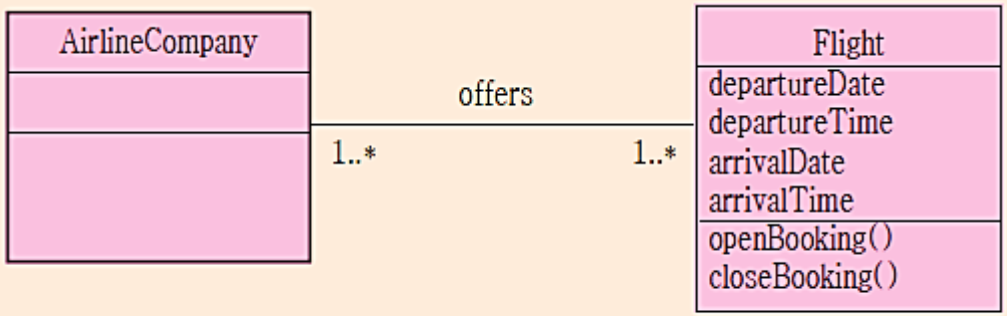


그림 5.26 ▸ 항공편과 항공사의 관계

항공사는 항공편을 제공한다

클래스 다이어그램 작성

- 항공권 예매 시스템 클래스 다이어그램 작성

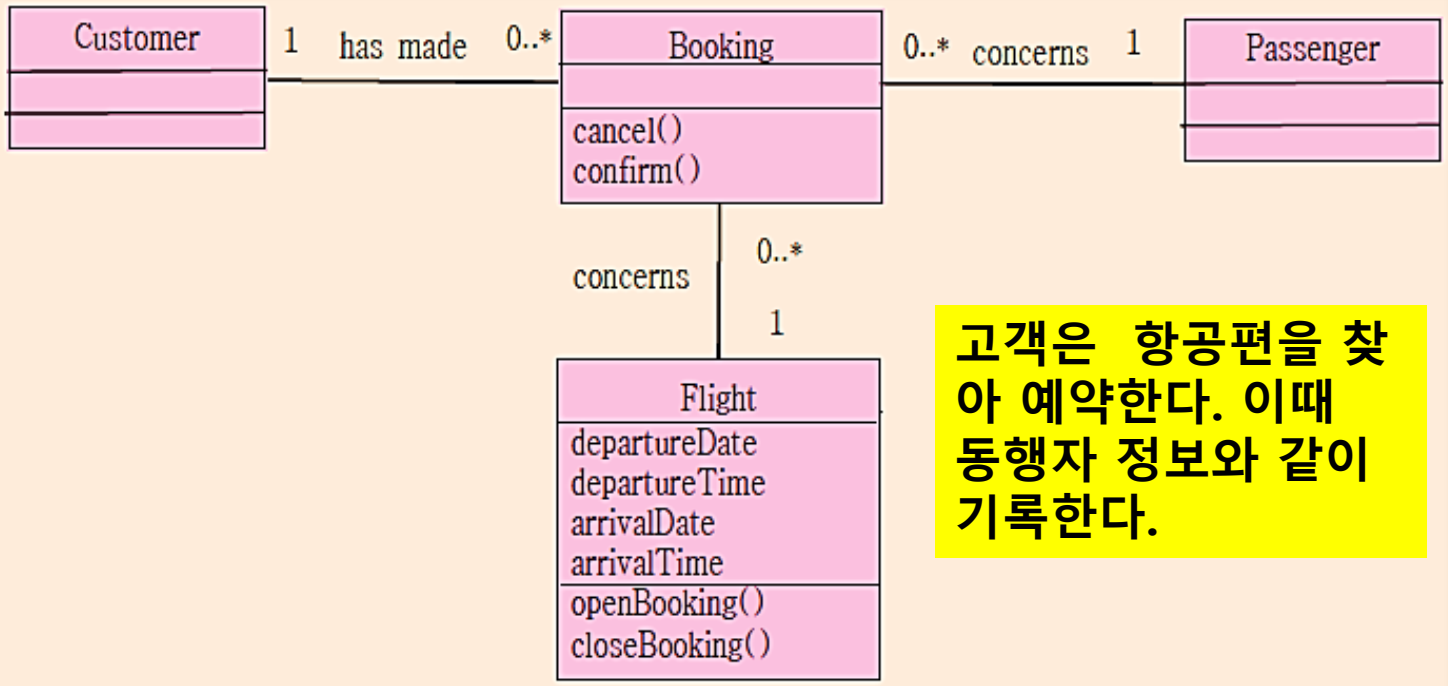


그림 5.27 ▶ 항공편과 예약, 고객, 탑승자 관계

클래스 다이어그램 작성

- 항공권 예매 시스템 클래스 다이어그램 작성

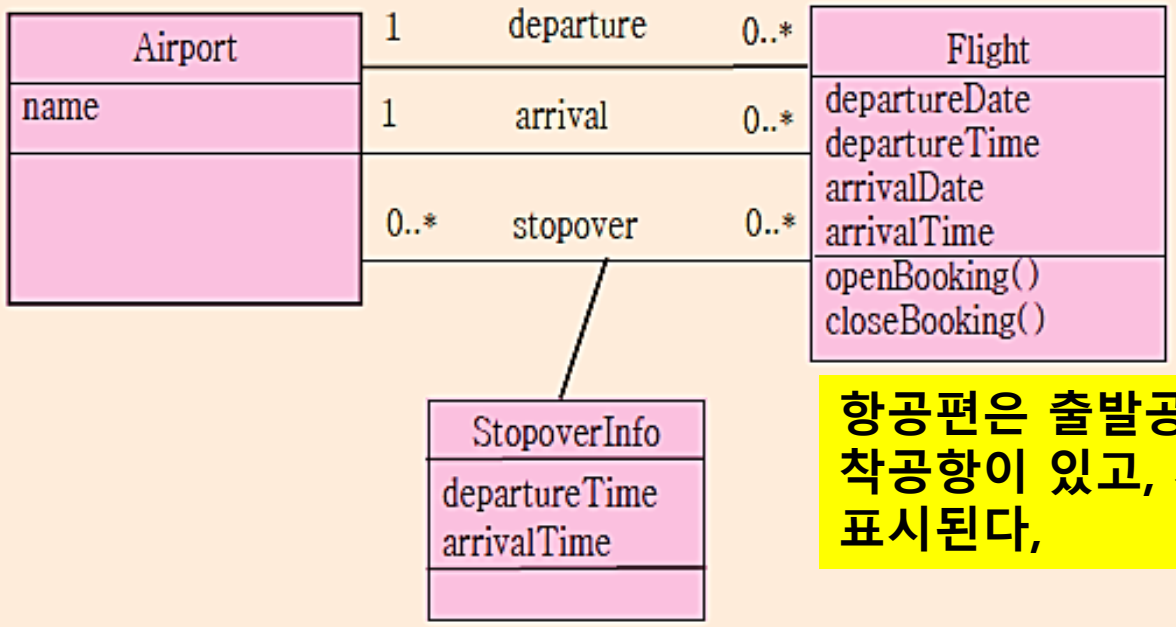


그림 5.28 ▶ 항공편과 공항의 관계

항공편은 출발공항, 도착공항이 있고, 시간 이 표시된다,

클래스 다이어그램 작성

- 항공권 예매 시스템 클래스 다이어그램 작성

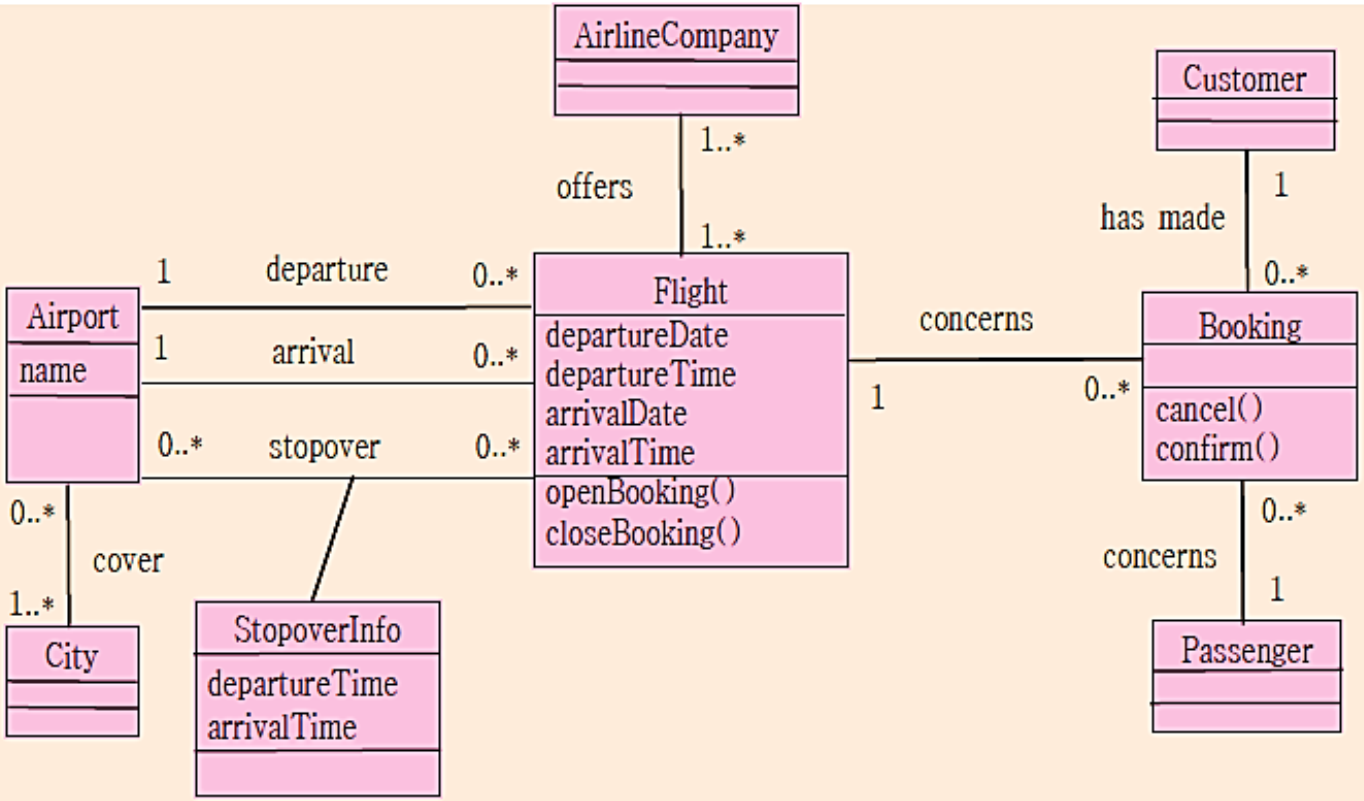
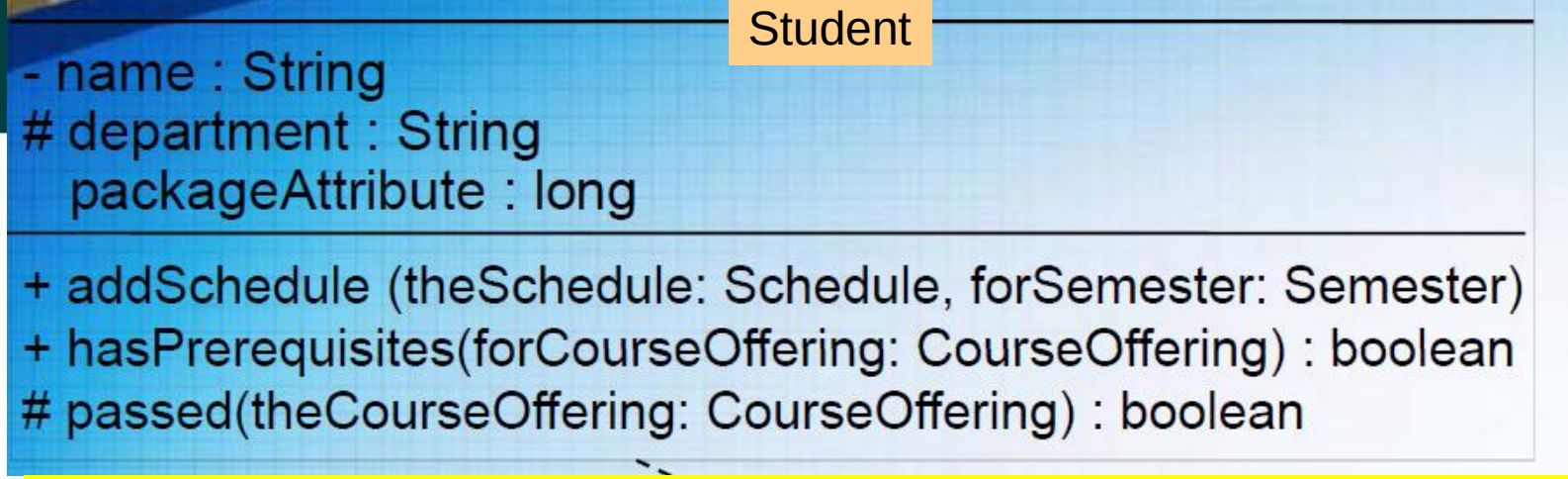
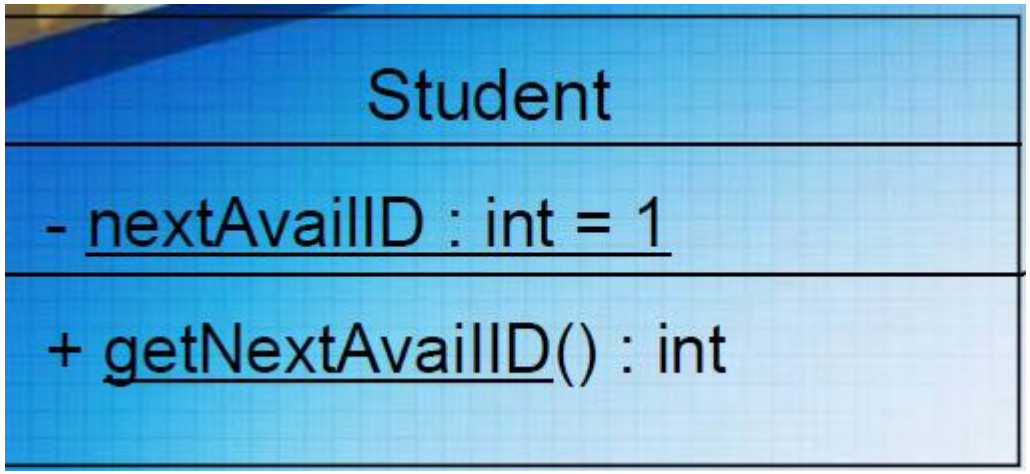


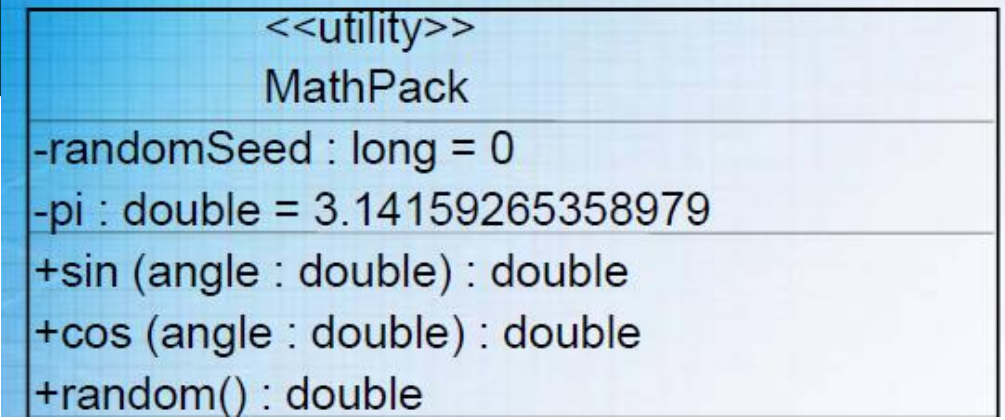
그림 5.29 ▶ 항공권 예약 시스템의 모델링




```
public class Student
{  private String name;
  protected String department;
  long packageAttribute;
  public void addSchedule (Schedule theSchedule; Semester forSemester) {
     …
   }
  public Boolean      hasPrerequisites(CourseOffering forCourseOffering) {
     …
  }
  protected Boolean passed(CourseOffering theCourseOffering) {
     …
 }
}
```



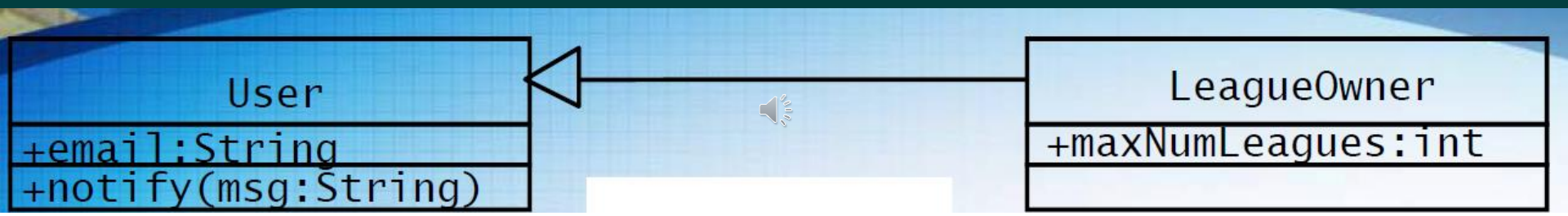

```
class Student {
  private static int nextAvailID = 1;
  static int getNextAvailID()   {
    …
  }
}
```



사용 예)

```
void somefunction() {
   . . .
 myCos = MathPack.cos(90.0);
  …
}
```

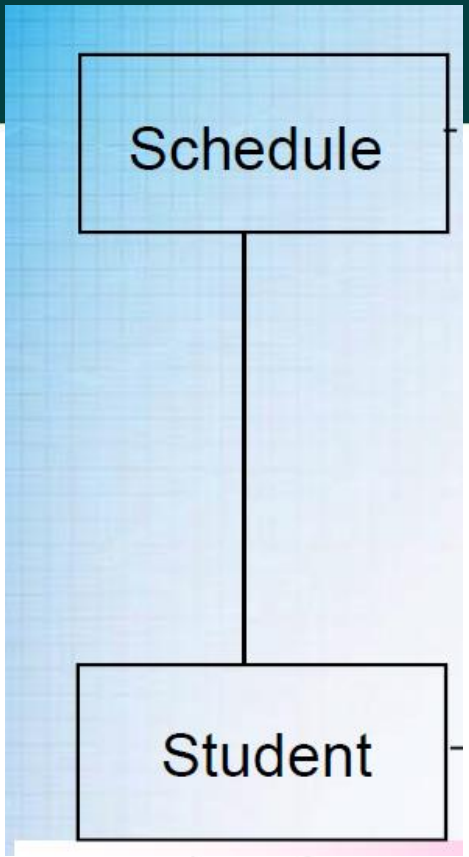
```
import java.lang.Math;
import java.util.Random;
class MathPack {
  private static randomSeed long = 0;
  private final static double pi = 3 14159265358979
  static double sin(double angle) {
     return Math.sin(angle);
  }
  static double cos(double angle) {
     return Math.cos(angle);
  }
  static double random() {
    return new Random(seed).nextDouble();
  }
}
```




```
public class User {
  private String email;
  public String getEmail() {
       return email;
  }

  public void setEmail(String value){
     email = value;
  }

  public void notify(String msg) {
    //
  }
}
```

```
public class LeagueOwner extends
  User {
  private int maxNumLeagues;
  public int getMaxNumLeagues() {
    return maxNumLeagues;
  }
  public void setMaxNumLeagues
       (int value) {
       maxNumLeagues = value;
   }
}
```

연관




```
class Schedule
{
  public Schedule() { } //constructor
  private Student theStudent;
}
```

```
class Student
{
  public Student() { }
  private Schedule theSchedule;
}
```