

# 4장. 다른 데이터 모델들

---

- ◆ 객체 지향 개념
- ◆ ODL의 소개
- ◆ ODL의 메소드, 타입, 서브 클래스
- ◆ ODL에서 관계 설계로의 변환
- ◆ 객체 관계형 모델



# 객체 지향 개념

---

## ◆ 개관

- 개선된 프로그램 구조, 신뢰성 있는 **SW** 구현 도구로 출발
- **C++**, **Java** 등의 객체 지향 프로그래밍 언어가 각광 받음
- 객체 지향 개념이 데이터베이스 분야에 적용되어 확장 됨

## ◆ 특징

- 강력한 타입 시스템
- 클래스(**Classes**) : 공동된 특성을 가지는 객체들의 집합 또는 정의
- 객체 식별자(**Object Identity:OID**) : 각각의 객체가 가지는 유일한 식별자
- 상속(**Inheritance**) : 클래스 계층에서 상위 클래스의 특징들을 물려 받음

# 타입 시스템

---

- ◆ 기본 타입 : 정수, 실수, 부울, 문자열 등의 원자 타입
- ◆ 사용자 정의 타입 : 타입 생성자(**type constructor**)로 확장된 타입
  - 레코드 구조체(**Record Structure**) : **Struct**  $N \{T_1 \ F_1, T_2 \ F_2, \dots, T_n \ F_n\}$ 는 타입 이름이  $N$ 이며  $n$ 개의 필드로 구성된 구조를 나타내는 타입이다.  $i$ 번째 필드의 이름은  $F_i$ 이며 타입은  $T_i$  이다.
  - 컬렉션 타입(**Collection Type**) : 타입  $T$ 에 대해서 컬렉션 연산자를 적용하여 새로운 타입 생성. 배열, 리스트, 집합, 백 등의 연산자.
    - » “정수의 배열” , “실수의 리스트” , “정수의 집합”
  - 참조 타입(**Reference Type**) : 타입  $T$ 에 대한 참조는 타입  $T$ 의 값이 저장된 위치 값을 가진다. C와 C++의 포인터 타입.

# 메소드

---

- ◆ 클래스에서 메소드가 정의된다.
  - 클래스 **C**의 메소드는 이 클래스의 객체를 인자로 갖는다.
  - 메소드는 함수
  - 메소드는 클래스가 추상 데이터 타입이 되도록 도와준다.
- ◆ 추상 데이터 타입(**Abstract Data Type :ADT**)
  - 캡슐화(encapsulation) : 클래스의 객체에 대한 접근은 해당 클래스의 메소드를 통해서만 가능
    - » 의도하지 않은 객체의 접근을 메소드를 통해서 제어
  - 클래스의 구현자가 의도하지 않는 방향으로 클래스의 객체가 변경되지 못하게 한다.
    - » 신뢰성 있는 **SW** 개발의 중요한 수단

# 클래스 계층(Class Hierarchy)

---

## ◆ 클래스 C를 클래스 D의 서브클래스(subclass)로 선언

- 클래스 C는 클래스 D에 정의된 모든 메소드와 특성들을 상속 받음
- 클래스 C에서 상속 받은 메소드를 재정의 또는 대체하거나 새로운 메소드의 추가가 가능

## [예제] 은행 계좌 클래스

```
Class Account = {  
    accountNo: integer;  
    balance: real;  
    owner: REF Customer;  
}
```

```
Class TimeDeposit : Account = {  
    dueDate: date;  
}
```

- 메소드 **deposit(a: Account, m: real)** : 계좌 a의 잔액을 m만큼 증가
- 클래스 **TimeDeposit**에 메소드 **penalty(a: TimeDeposit)** 추가 : 계좌 a에서 만기 이전에 예금을 인출 할 때 부과시키는 금액을 계산

# ODL의 소개

---

## ◆ 객체 기술 언어 (Object Definition Language: ODL)

- 객체 지향 방법으로 (예를 들어 C++와 같이) 데이터베이스의 구조, 스키마(schema)를 명시하는 언어
- 데이터 정의어(data definition language)
  - » 데이터 조작어(data manipulation language) 또는 질의어(query language)가 아님
- ODL로 기술된 설계는 OODBMS의 선언문으로 직접 변환될 수 있다

# ODL의 소개 (계속)

---

## ◆ 객체(object)

- 모델링하고자 하는 실 상황이 객체들로 구성된다고 간주  
[예] 사람, 은행 계좌, 항공편

## ◆ 클래스(class)

- 비슷한 특성을 가진 객체들의 타입 선언

## ◆ 클래스의 특성(properties)

- ① 애트리뷰트(attribute) : 객체의 어떤 특징에 값을 연관 시킴
  - » ODL에서는, 애트리뷰트 타입에 클래스가 포함되어서는 안됨
- ② 관계성(relationship) : 다른 객체와의 연결
  - » 타입은 객체에 대한 참조이거나, 이러한 참조들의 collection이다.
- ③ 메소드(method) : 해당 클래스의 객체들에 적용될 수 있는 함수

# ODL에서 클래스 선언

---

## ◆ ODL에서 클래스 선언

- 키워드 `Class`
- 클래스의 이름
- 클래스의 특성들을 나열한 중괄호 리스트
  - » 특성들로는 애트리뷰트, 관계성, 메소드가 있다.

```
Class <name> {  
    <list of properties>  
}
```



# ODL의 애트리뷰트

---

```
Class Movie {  
    attribute string title;  
    attribute integer year;  
    attribute integer length;  
    attribute enum Film {color, blackAndWhite} filmType;  
};
```

→ 열거 타입(**enumerated type**). 이 타입의 이름은 **Film**이며, 애트리뷰트의 이름은 **filmType**이다.

## ■ Movie 클래스의 객체

```
("Gone with the Wind", 1939, 231, color),  
("Sting", 1973, 129, color)
```

# ODL의 애트리뷰트 (계속)

---

```
Class Star {  
    attribute string name;  
    attribute Struct Addr  
        {string street, string city} address;  
};
```



레코드 구조(record structure) 타입


# ODL에서 관계성

---

## ◆ ODL에서 관계성(relationship)

- 객체가 동일한 클래스나 다른 클래스의 객체들과 연관된 방식

```
Class Movie {  
    attribute string title;  
    attribute integer year;  
    attribute integer length;  
    attribute enum Film {color, blackAndWhite} filmType;  
    relationship Set<Star> stars;  
};
```



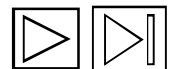
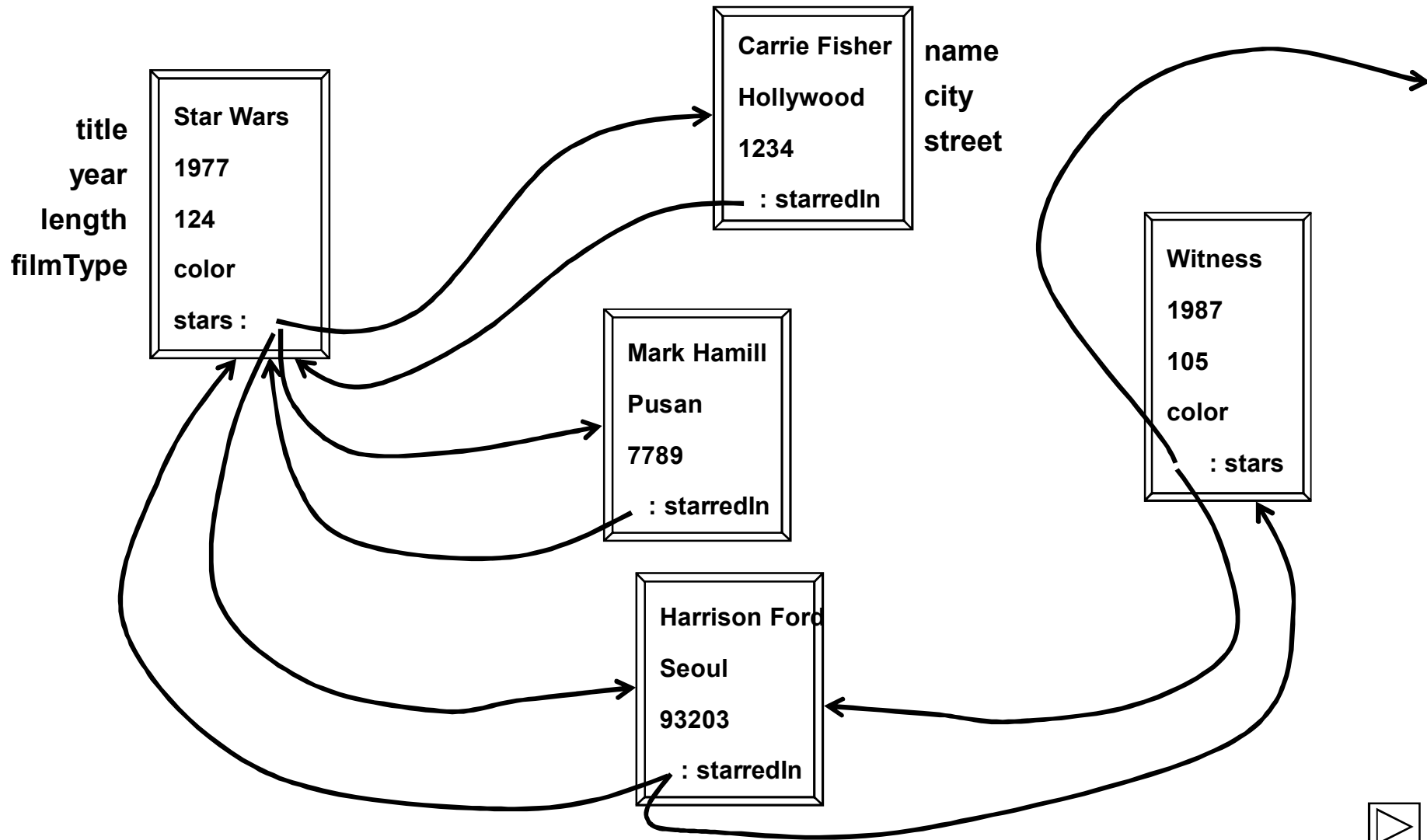
Movie객체에 있는 스타들의 집합

**stars**는 다른 객체에 대한 참조(reference)들을 가지고 있다.

☞ Movie에 있는 stars는 Movie와 Star클래스의 관계성을 나타낸다.

☞ relationship Star starOf : 각 영화 객체를 하나의 Star객체와 연관시킴

# ODL에서 관계성 (계속)



# ODL에서 역(inverse) 관계성

- ◆ S라는 스타가 영화 M에 대한 스타들의 집합인 stars에 있으면, 영화 M은 스타 S에 대한 starredIn 집합에도 있어야 한다.

```
Class Star {  
    attribute string name;  
    attribute Struct Addr  
        {string street, string city} address;  
    relationship Set<Movie> starredIn  
        inverse Movie::stars;  
};
```

- ☞ ODL에서 관계성은 역관계성을 반드시 가져야 한다. 즉, 양방향 접근이어야 한다. C++ 는 단방향 관계성을 허용한다.

```
Class Studio {  
    attribute string name;  
    attribute string address;  
    relationship Set<Movie> owns  
        inverse Movie::ownedBy;  
};
```

attribute Star::Addr address

# 관계성의 다중 연관성(multiplicity)

---

클래스 C : 클래스 D	클래스 C의 관계성 타입	클래스 D의 관계성 타입
M:N	Set<D>	Set<C>
M:1	D	Set<C>
1:1	D	C

# 메소드(Method)

---

## ◆ 클래스와 연관된 함수

- 그 클래스의 객체들에게 적용할 수 있는 실행 가능한 코드
- 메소드 시그니처(signature) : 메소드 이름과 매개변수의 입출력 타입 선언

## ◆ 시그니처의 구문

- 아래 두가지 사항을 제외하고는 C의 함수 선언과 유사
  - » 매개변수의 명시 : `in`, `out`, `inout`
  - » 예외 사건(exception)의 발생
    - ◆ 비정상적이거나 예상치 못한 상태에 대한 특별한 응답 방식
    - ◆ 키워드 : `raises`

## ◆ 시그니처를 사용하는 이유

- 구현이 설계 사양과 일치하는 지를 검사
  - » 연산의 의미까지 정확하게 구현되었는지는 검사할 수 없다.
  - » 입출력 매개변수들의 수와 타입이 올바르게 사용되었는지는 검사할 수 있다.

# 메소드 시그니처를 추가한 Movie 클래스

---

```
Class Movie
    (extent Movies
     key (title, year))
{
    attribute string title;
    attribute integer year;
    attribute integer length;
    attribute enumeration(color,blackAndWhite) filmType;
    relationship Set<Star> stars
        inverse Star::starredIn;
    relationship Studio ownedBy
        inverse Studio::owns;
    float lengthInHours() raises(noLengthFound);
    starNames(out Set<string>);
    otherMovies(in Star, out Set<Movie>) raises(noSuchStar);
};
```



# ODL에서의 타입

---

## ◆ 기본 타입

- 원자적(**atomic**) 타입: 열거(enumeation), 정수, 실수, 문자, 문자열, 부울(**boolean**),
- 클래스 이름 : 애트리뷰트와 관계성을 그 구성 요소로 가지는 구조 타입 (**object class**를 의미)

(예) `Movie`, `Star`

- ☞ 기본 타입들은 결합되어 복합 타입이 된다.
  - » 타입 생성자(`constructor`)를 이용

# ODL에서의 타입 (계속)

---

## ◆ 타입 생성자(constructor)

- 집합 : `Set<T>`는 타입 `T`의 원소들로 만들 수 있는 모든 가능한 집합을 그 값으로 가지는 타입이다.
- 백(**Bag**) : `Bag<T>`는 타입 `T`의 원소들로 만들 수 있는 백(또는, 다중 집합(**multi-set**))을 그 값으로 가지는 타입이다.
  - » `{1, 2, 1}`은 집합이 아니라 백이다.
- 리스트 : `List<T>`는 제로 또는 하나 이상의 타입 `T`의 원소들로 구성된 유한 리스트를 그 값으로 가지는 타입이다. 원소들 사이의 순서가 중요하다.

# ODL에서의 타입 (계속)

---

- 배열 : `Array<T, i>`는  $i$ 개의 타입  $T$ 의 원소들로 구성된 배열을 나타내는 타입이다.
  - » `Array<char, 10>`은 10개의 문자로 구성된 문자열을 나타낸다.
- 사전 : `Dictionary<T, S>`는 유한 쌍의 집합. 각 쌍은 키타입(key type)  $T$ 의 값과 해당 키 값에 해당하는 범위 타입(range type)  $S$ 의 값을 가진다.
- 구조체 : `Struct N {T1 F1, T2 F2, ..., Tn Fn}`는 타입 이름이  $N$ 이며  $n$ 개의 필드로 구성된 구조를 나타내는 타입이다.  $i$ 번째 필드의 이름은  $F_i$ 이며 타입은  $T_i$  이다.

» `attribute Struct Addr`

`{string street, string city} address;`

# ODL에서의 타입 (계속)

---

## ◆ 컬렉션 타입

- 집합, 백, 리스트, 배열, 사전

## ◆ 애트리뷰트의 타입

- 원자적 타입이거나 각 필드가 원자적 타입인 구조(structure)이다. 컬렉션 타입이 단 한번 적용될 수 있다.

## ◆ 관계성의 타입

- 클래스 타입이거나 클래스 타입에 단 한번 적용된 컬렉션 타입이다.

☞ 클래스 타입은 애트리뷰트의 타입으로 사용되지 않으며, 원자적 타입은 관계성의 타입으로 사용되지 않는다.

☞ 관계성의 타입에는 구조형이 허용되지 않는다.

# ODL에서의 타입 (계속)

---

## ◆ 애트리뷰트 예제

- **integer, List<real>**
- **Struct N {string field1, integer field2}**
- **Array<Struct N {string field1, integer field2}, i>**

## ◆ 관계성 타입 예제

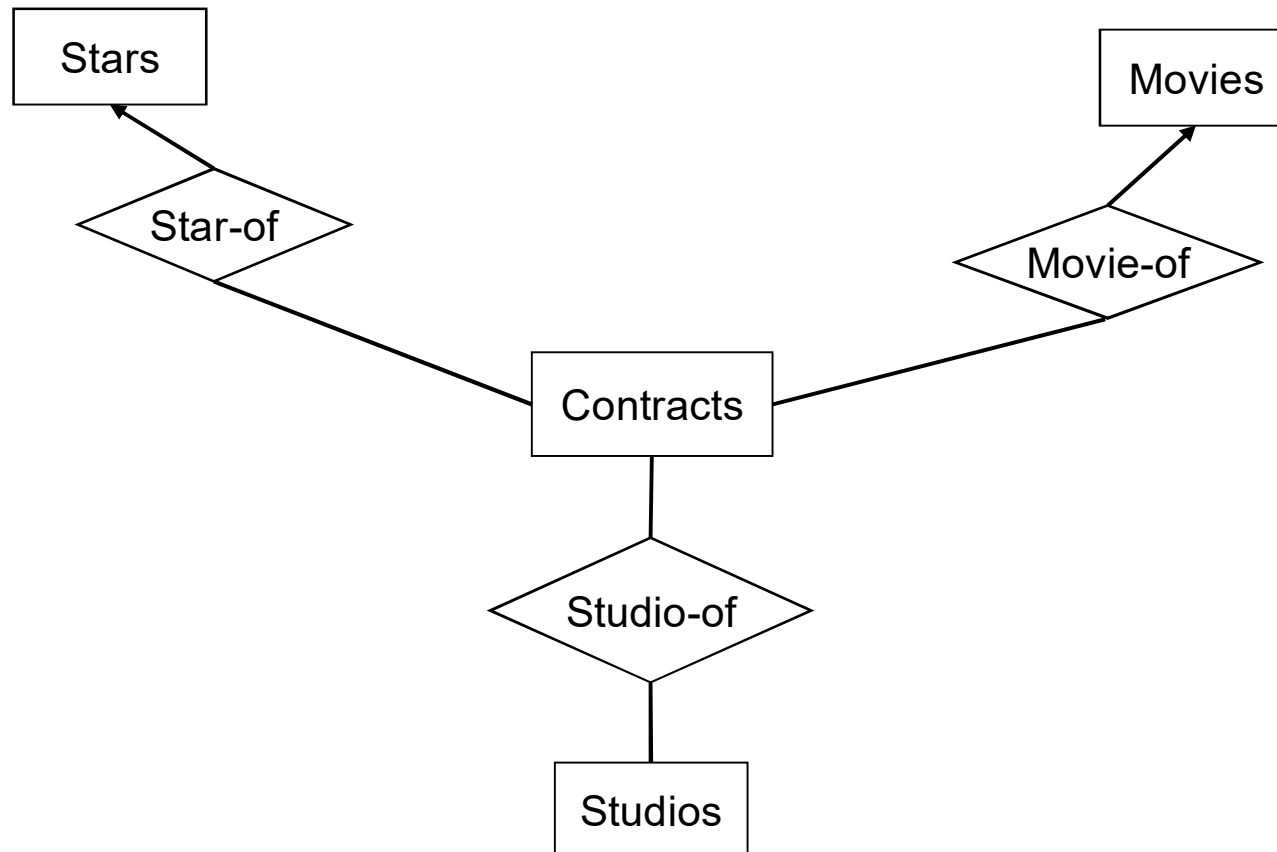
- **Movie, Star, Set<Movie>, List<Studio>**

## ◆ 규칙에 위배 되는 예제

- 애트리뷰트 : **Movie, Set<Array<real>>**
- 관계성 : **Set<integer>, Struct N <Movie f1, Star f2>, Set<Bag<Movie>>**

# E/R Diagram 예제

---



# ODL의 다중 방향 관계성

---

- ◆ 연결 엔티티 집합을 이용하여 **ODL**에서도 쉽게 다중 방향 관계성을 정의

```
Class Contract {  
    attribute integer salary;  
    relationship Star theStar  
        inverse Star::contractsFor;  
    relationship Movie theMovie  
        inverse Movie::contractsFor;  
    relationship Set<Studio> studios  
        inverse Studio::contractsFor;  
};
```

- **Star, Movie, Studio** 등의 클래스에도 **Contract** 클래스에 대한 관계성을 추가한다. 예를 들어 **Movie** 클래스에는 다음과 같은 선언문이 추가된다.
  - » **relationship Set<Contract> contractsFor inverse Contract::theMovie;**

# ODL에서의 서브클래스

---

## ◆ 서브클래스(subclass)

- 슈퍼클래스(superclass)의 모든 애트리뷰트와 자신의 고유한 애트리뷰트를 상속받는(inherit)다.

```
Class Cartoon extends Movie {  
    relationship Set<Star> voices;  
};
```

- Cartoon은 Movie의 서브클래스이다.
- 각 만화 영화 객체는 자신의 voices 관계성과 더불어 Movie로부터 상속된 title, year, length, filmType 애트리뷰트와 stars, ownedBy 관계성을 가지게 된다.



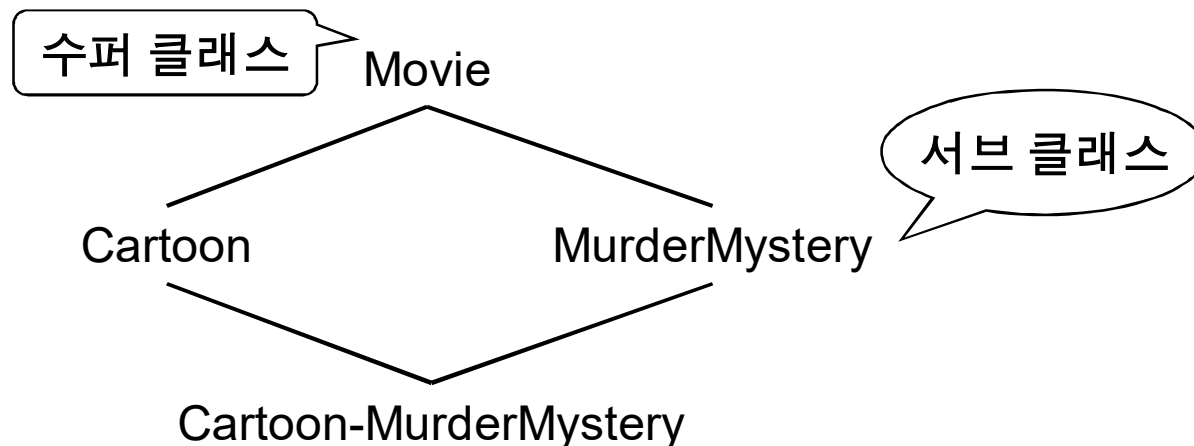
# ODL에서의 다중 상속

## ◆ 다중 상속(multiple inheritance)

- 클래스는 하나 이상의 수퍼클래스들을 가질 수 있다.

```
Class MurderMystery extends Movie {  
    attribute string weapon;  
};
```

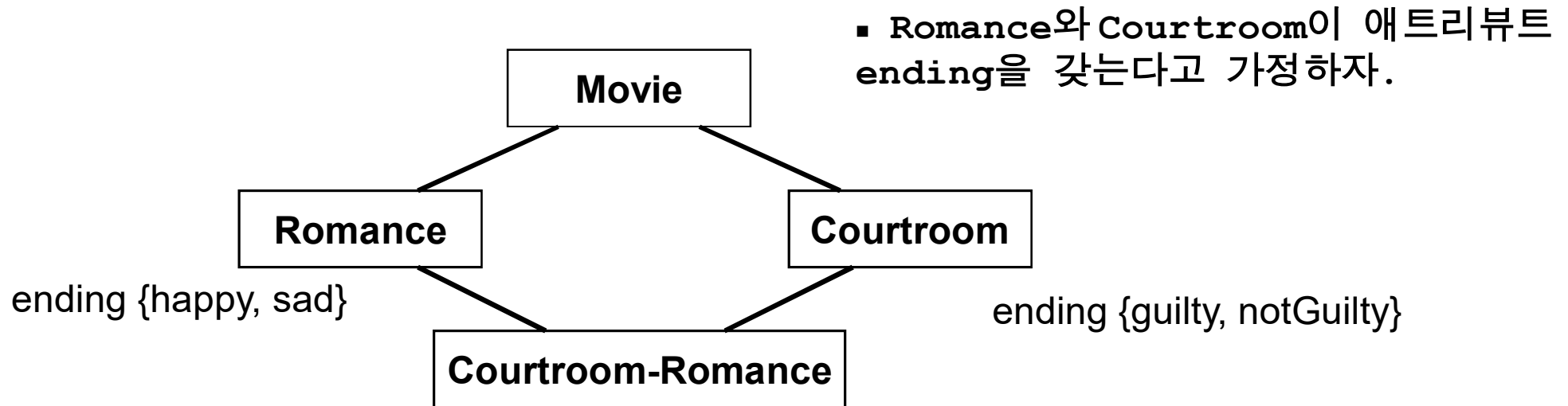
```
Class Cartoon-MurderMystery  
    extends Cartoon : MurderMystery {};
```



# 다중 상속에 의한 충돌(conflict)

---

- ◆ 둘 이상의 슈퍼클래스가 같은 이름의 애트리뷰트나 관계성을 가질 수도 있고, 이 특성들의 타입이 다를 수도 있다.



- Courtroom-Romance 클래스에 상속된 애트리뷰트 ending의 타입이 불명확하다.

# 다중 상속에 의한 충돌의 해결방법

---

- ① 하나만 상속: 어느 것을 서브클래스에 적용할 것인지를 명시(selection)
  - Courtroom-Romance는 슈퍼클래스 Romance의 애트리뷰트 ending을 상속 받도록 명시한다.
- ② 둘 다 상속: 서브클래스에서 충돌이 일어나는 애트리뷰트 중 하나에 새로운 이름을 부여(rename)
  - Courtroom-Romance가 애트리뷰트 ending을 슈퍼클래스 Romance로부터 상속 받는다면, Courtroom-Romance는 슈퍼클래스 Courtroom 으로부터 애트리뷰트 ending을 verdict라는 이름으로 변경하여 상속 받는다고 명시한다.
- ③ 상속받지 않음: 슈퍼클래스에서 정의된 특성들을 다시 정의(redefine)
  - 애트리뷰트 ending을 슈퍼클래스로부터 직접 상속 받지 않도록 한다. 클래스 Courtroom-Romance에서 애트리뷰트 ending을 관객의 만족도를 나타내는 정수 타입으로 재정의한다.

# 익스텐트(Extent)

---

## ◆ 의미

- 데이터베이스의 일부로 **ODL** 클래스를 정의할 때 클래스 정의와 클래스에 속하는 객체들의 집합을 분리
  - » 릴레이션 스키마와 릴레이션 인스턴스 사이의 관계
  - » 엔티티 정의와 엔티티 집합 사이의 관계
- 해당 클래스에 현재 존재하는 모든 객체들의 집합
- **OQL** 질의는 (클래스 이름이 아니라) 클래스의 익스텐트를 참조

# ODL에서 키의 선언

---

## ◆ ODL에서 키의 사용

- **OID**가 모든 객체들에 대해서 유일한 객체 식별자 기능을 한다.
- 키 선언은 선택 사항

## ◆ 키의 선언

- **Class Movie (extent Movies key (title, year)) { ... } : (title, year)가 키**
- **Class Star (extent Stars key name) { ... } : (name)이 키**
- **Class Employee (extent Employees key empID, ssNo) { ... } :**  
**(empID), (ssNo) 등의 두 개의 후보 키 선언**
- **Class Employee (extent Employees key (empID, ssNo)) { ... } :**  
**(empID, ssNo)가 키**

# ODL에서 키의 선언 (계속)

---

## ◆ 메소드나 관계성을 키로 선언 가능

- 키로 선언된 메소드의 경우 결과값이 유일하다는 의미
- 관계성을 키로 선언하는 경우 E/R 다이어그램의 약 엔티티 집합의 표현에 적용할 수 있다.

**Class Crew**

**(extent Crews key (number, partOf))**

**{**

**attribute integer number;**

**relationship Studio partOf**

**inverse Studio:: crewsOf;**

**}**

# ODL에서 관계 모델로의 변환

---

## ◆ ODL 애트리뷰트에서 관계 애트리뷰트로 [간단한 경우]

- 클래스의 모든 특성은 애트리뷰트뿐이다.
  - » 관계성이나 메소드는 없음
- 애트리뷰트의 타입은 모두 원자적 타입이다.
  - » 구조나 집합이 아님

[예] `Movies(title, year, length, filmType)`

# ODL에서 관계 모델로의 변환 (계속)

## ◆ 클래스에 비원자적 타입을 갖는 애트리뷰트

- 구조, 집합, 백, 리스트, 배열

☞ 릴레이션의 애트리뷰트는 원자적 타입을 가진다.

## ① 레코드 구조

- 구조의 각 필드에 대해

하나의 애트리뷰트를 생성

```
Class Star (extent Stars) {  
    attribute string name;  
  
    attribute Struct Addr  
        {string street, string city} address;  
  
};  
  
Stars(name, street, city)
```

name	street	city
Carie Fisher	123 Maple St.	Hollywood
Mark Hamill	456 Oak Rd.	Brentwood
Harrison Ford	789 Palm Dr.	Beverly Hills



# ODL에서 관계 모델로의 변환 (계속)

## ② 집합

- 각 원소에 대해 하나의 튜플을 생성

```
Class Star (extent Stars) {  
    attribute string name;  
    attribute Set  
        <Struct Addr {string street, string city}> address;  
    attribute Date birthdate;  
};
```

name	street	city	birthdate
Carrie Fisher	123 Maple St.	Hollywood	9/9/99
Carrie Fisher	5 Locust Ln.	Malibu	9/9/99
Carrie Fisher	110 Daeyun	Pusan	9/9/99
Mark Hamill	456 Oak Rd.	Brentwood	8/8/88

정보의 중복

- name → birthdate 가 성립하므로 BCNF 위반이다.

# ODL에서 관계 모델로의 변환 (계속)

---

## ③ 백

- 백의 각 원소에 대해 백에 그 원소가 나타나는 빈도를 표시하는 새로운 애트리뷰트 *count* 를 릴레이션 스키마에 추가

name	street	city	count
Carrie Fisher	123 Maple St.	Hollywood	2
Carrie Fisher	5 Locust Ln.	Malibu	3

## ④ 리스트

- 리스트 내부의 위치를 표시하는 새로운 애트리뷰트 *position* 을 추가

name	street	city	birthdate	position
Carrie Fisher	123 Maple St.	Hollywood	9/9/99	1
Carrie Fisher	5 Locust Ln.	Malibu	9/9/99	2
Mark Hamill	456 Oak Rd.	Brentwood	8/8/88	1

# ODL에서 관계 모델로의 변환 (계속)

---

## ◆ 관계성의 표현

- 단일값(single-valued) 관계성과 다중값(multi-valued) 관계성

```
Class Movie (extent Movies key (title, year)) {  
    attribute string title;  
    attribute integer year;  
    attribute integer length;  
    attribute enumeration(color, blackAndWhite) filmType;  
    relationship Set<Star> stars                /* 다중값 */  
        inverse Star::starredIn;  
    relationship Studio ownedBy                  /* 단일값 */  
        inverse Studio::owns  
};
```

- ☞ 관계 모델에는 포인터의 개념이 없다 : 각 튜플을 유일하게 결정할 수 있는 역할은 주키를 이루는 애트리뷰트가 담당

# ODL에서 관계 모델로의 변환 (계속)

## ◆ 단일값 관계성

- 연관된 클래스에서 각 객체를 나타내는 키를 찾고,  
그 클래스의 키에 해당하는 애트리뷰트를 추가한다.  
» 연관된 객체를 나타내는 값이 포인터의 효과를 낸다.

title	year	length	film Type	studioName
Star Wars	1977	124	color	Fox
Mighty Ducks	1991	104	color	Disney
Wayne's World	1991	95	color	Paramount

Studio 튜플에  
대한 포인터(주키)

ownedBy 정보를 가진 릴레이션 **Movie**

# ODL에서 관계 모델로의 변환 (계속)

## ◆ 다중값 관계성

- 연관된 각 객체를 나타내는 키를 찾고,
- 각 값에 대하여 하나의 튜플을 생성하여 연관된 객체들의 집합을 표현한다.

title	year	length	film Type	studioName	starName
Star Wars	1977	124	color	Fox	Carrie Fisher
Star Wars	1977	124	color	Fox	Mark Hamill
Star Wars	1977	124	color	Fox	Harrison Ford
Mighty Ducks	1991	104	color	Disney	Emilio Estevez
Wayne's World	1991	95	color	Paramount	Dana Carvey
Wayne's World	1991	95	color	Paramount	Mark Hamill

Star 튜플에  
대한 포인터(주키)

☞ 한 객체가  $n_1$ 개의 객체를  $R_1$ 을 통해 연결되어 있고,  $n_2$ 개의 객체를  $R_2$  등으로 연결되어 있다고 하자. 이러한 경우 각 객체에 대해서  $n_1 \times n_2 \times \dots \times n_k$  개의 튜플들이 생성된다.



# ODL에서 관계 모델로의 변환 (계속)

---

## ◆ 관계성과 그 역관계성에 대한 표현

### ① 관계성을 양 방향(즉, 두 테이블에)으로 표현

» 다대일 관계성에 대해서는 중복 (redundancy)이 발생한다.

[예] **Movie(title,year,length,filmType,starName)**와 **Star(name,address,title,year)**에서 **starName**과 **(title,year)** 둘 모두 있을 필요없음.

### ② 관계성을 한 방향(즉, 한 테이블에만)으로 표현

» 다대일 관계성에 대해서는 “다”에 해당하는 테이블에 관계성을 포함시킨다.

[예] 관계성 **ownedBy: (Studio 테이블이 아닌) Movie**에 표현

**Movie(title, year,length,filmType,studioName)** : 한 영화는 한 스튜디오를 가짐

**Studio(name,address,title,year)** : 한 스튜디오는 여러 개의 영화를 가짐. **name**과 **address**가 중복됨

### ③ 관계성과 그 역을 새로운 테이블로 분리하여 표현 : E/R에서 사용

### ④ 정규화(normalization) 과정

# ODL에서 관계 모델로의 변환 (계속)

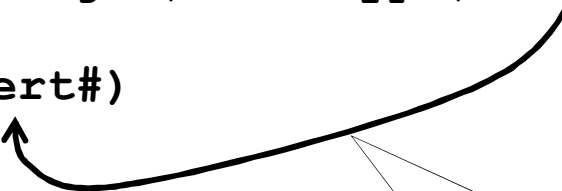
---

## ◆ 키가 존재하지 않는 클래스

- OO 모델에서는 한 클래스에 있는 두 객체들이 모든 특성에 동일한 값을 가지는 것을 허용 : **OID**로 객체의 유일성 보장
- 관계 모델에서 유일성(**uniqueness**)을 보장하기 위해서는 “객체 식별자(**object identifier**)”에 해당하는 새로운 애트리뷰트를 릴레이션에 추가해야 할 경우가 있다.

`Movie(title, year, length, filmType, cert#)`

`Star(name, address, cert#)`



각 스타에 대한  
인증 번호

# E/R에서 관계 모델로의 변환

---

## ◆ E/R 설계와 ODL 설계의 차이점

	E/R 설계	ODL 설계
관계성	독립적인 개념으로 취급	클래스내의 특성으로 표현
애트리뷰트 타입	레코드 구조 타입은 허용되나, 집합이나 다른 컬렉션 타입들은 명확하지 않다.	구조, 컬렉션 타입이 허용된다.
관계성의 애트리뷰트	애트리뷰트를 가질 수 있다.	관계성에 애트리뷰트에 해당하는 개념이 없다.



# 객체 관계형 모델

---

## ◆ 개념

- 관계형 모델 + 객체 지향 모델
- 90년대에 객체 지향형 **DBMS**가 개발되었으나 관계형 **DBMS**의 시장을 대체하지는 못 함
  - » 상당 기간 관계형 **DBMS**로 개발된 많은 프로그램과 **DB**의 전환 부담
  - » 기존 관계형 **DBMS**와의 호환성의 중요성
- 관계형 모델에 객체 지향 개념을 접목하는 방식

# 객체 관계형 모델의 특징

---

## ◆ 구조화된 타입의 애트리뷰트

- 원자적 타입과 사용자 정의 타입 지원
  - » 원자적 타입들로 구성된 타입과 구조체, 집합, 백과 같은 타입 구성자들로 이루어진 타입
- 구조체의 집합과 같은 타입이 애트리뷰트의 타입으로 가능

## ◆ 메소드

- 타입에 대한 연산자 또는 함수 정의 가능
- **ADT** 지원

## ◆ 튜플 식별자

- 객체 관계형 **DBMS**의 튜플이 객체의 역할 수행
- 각 튜플은 **OID**와 유사한 유일한 식별자를 가진다.
- 원칙적으로 내부적 용도

## ◆ 참조

- 다른 데이터를 다양한 방법으로 참조 가능

# 중첩 릴레이션

---

## ◆ 개념

- 릴레이션의 구성요소로 또 다른 릴레이션을 포함
  - » 애트리뷰트 타입이 릴레이션 스키마로 가능
- 비 원자적인 형태의 애트리뷰트 표현 가능
- **BASIS** : 원자적 타입(정수, 실수, 문자열 등)은 애트리뷰트 타입으로 가능
- **INDUCTION** : 한 릴레이션 타입은 여러개의 애트리뷰트로 구성된 스키마가 되며, 임의의 애트리뷰트의 가능한 타입
  - » 스키마는 어떤 애트리뷰트의 타입으로 가능

## ◆ 중첩 릴레이션의 예제

- **Stars(name, address(street, city), birthdate, movies(title, year, length))**
  - » 애트리뷰트 **address**는 **street**와 **city**등으로 구성된 릴레이션 타입
  - » 애트리뷰트 **movies**는 **title, year, length**로 구성된 릴레이션 타입

# 중첩 릴레이션의 예제

<i>name</i>	<i>address</i>	<i>birthdate</i>	<i>movies</i>		
Fisher	<i>street</i>	<i>city</i>	<i>title</i>	<i>year</i>	<i>lenth</i>
	Maple	H'wood	Star Wars	1997	124
	Locust	Malibu	Empire	1980	127
			Return	1983	133
Hamill	<i>street</i>	<i>city</i>	<i>title</i>	<i>year</i>	<i>lenth</i>
	Oak	B'wood	Star Wars	1997	124
			Empire	1980	127
			Return	1983	133

스타와 스타가 출연한 영화를 위한 중첩 릴레이션 **Stars**의 예

# 참조 (Reference)

---

## ◆ 중첩 릴레이션의 문제점

- 중첩 릴레이션 **Stars**의 애트리뷰트 **movies**의 값인 **Star Wars**는 중복 가능
- **BCNF** 분해로는 해결 불가
- 참조 타입을 사용

## ◆ 개념

- 튜플 **t**를 튜플 **s**에 포함시키지 않고 튜플 **s**가 튜플 **t**를 참조하게 한다.
- 애트리뷰트의 타입은 스키마를 갖는 튜플에 대한 참조가 가능
- 애트리뷰트 **A**가 릴레이션 스키마 **R**에 해당하는 각 튜플에 대한 참조 타입인 경우
  1. 단일값 형태의 참조 :  $A(*R)$ 로 표현, 스키마 **R**의 튜플에 대한 참조 타입
  2. 다중값 형태의 참조 :  $A(\{*R\})$ 로 표현, 스키마 **R**의 튜플들의 집합에 대한 참조 타입
    - » **ODL**의 **Set<R>**와 유사한 형태

# 참조 값의 예

## ◆ 스키마 정의

- **Movies(title, year, length)**
- **Stars(name, address(street, city), birthdate, movies(\*Movies))**

<i>name</i>	<i>address</i>	<i>birthdate</i>	<i>movies</i>	
Fisher	<i>street</i>	9/9/99		<i>title</i>
	Maple			Star Wars
	Locust			1997
Hamill	<i>city</i>	8/8/88		124
	Oak			Empire
	B'wood			1980
				127
				Return
				1983
				133

# 객체 지향과 객체 관계형의 비교

---

## ◆ 객체와 튜플

- 객체 : 애트리뷰트와 관계성의 완전한 구조체
  - » ODL에서 관계성의 구현은 포인터로 가능
- 튜플 : 관계형 모델에서는 애트리뷰트만으로 구성
  - » 객체 관계형 모델에서는 참조를 이용하여 관계성도 표현 가능

## ◆ 익스텐트와 릴레이션

- 한 클래스에 속하는 객체들을 익스텐트로 관리
  - » 하나의 클래스 정의에 대해서 하나의 익스텐트만 존재
  - » 서브 클래스를 정의한다면 비슷한 클래스 정의에 대해서 여러개의 익스텐트가 존재 가능
- 객체 관계형 모델에서는 동일한 스키마에 해당하는 여러개의 다른 릴레이션이 존재 가능

## ◆ 메소드

- **SQL-99** 및 대부분의 객체 관계형 **DBMS**에서 메소드 정의 가능

# 객체 지향과 객체 관계형의 비교 (계속)

---

## ◆ 타입 시스템

- 원자적 타입과 구조체나 콜렉션 타입 구성자로 새로운 타입 정의
- **ODL** : 클래스 타입, 객체 관계형 모델 : 릴레이션 타입

## ◆ 참조와 객체 식별자

- 순수한 객체 지향 모델은 **OID**를 사용하지 못한다.
- 객체 관계형 모델은 참조 타입을 지원하면서 참조값을 보거나 변경할 수 있다.

## ◆ 이전 버전과의 호환성

- 오랜 기간 사용되어 온 관계형 **DBMS**와의 호환성으로 객체 관계형 **DBMS**는 완전한 객체 지향 모델을 지원하기 힘들다.