[우리 학과 취업스쿨 Day09]

소프트웨어 디자인 패턴과 리팩토링

리팩토링(Refactoring)

(프로그램의 가치를 높이는 코드 정리 기술)

기능 추가 - htmlStatement()

- htmlStatement() 작성하기
  - 기존의 statement()에서 계산 코드를 method로 뽑아내었기 때문에 이를 이용하여 htmlStatement() 를 쉽게 구성할 수 있다.

```
class Customer ...
  public String htmlStatement() {
    Enumeration rentals = _rentals.elements();
    String result = "<H1>Rentals for <EM>" + getName() + "</EM></H1><P>\n";
    while ( rentals.hasMoreElements() ) {
      Rental each = (Rental) rentals.nextElement();

      // 대여에 대한 요금 계산결과 표시
      result += each.getMovie().getTitle() + ": " + String.valueOf(each.getCharge()) + "<BR>\n";
    }
    // 꼬리말 달기
    result += "<P> You owe <EM> + String.valueOf(getTotalCharge()) + "</EM><P>\n";
    result += "On this rental you earned <EM> " + String.valueOf(getTotalFrequentRenterPoints()) +
          "</EM> frequent renter points<P>";
    return result;
  }
```

다시 Refactoring 실행

- 대여점에서
  - 영화를 분류하는 방법을 바꾸려고 한다.
  - 새로운 분류방법에 따라 요금과 포인트 계산방법이 달라질 것이다.
- 현재 시점에서 이런 식으로 변경하는 것은 어렵다.
- 다시 Refactoring 실행.

Refactoring 적용-Move Method

- Rental.getCharge() 에서
  - switch 문장이 다른 class 객체의 속성값에 기반하는 것은 별로 좋지 못하다.
  - Rental.getCharge() → Movie.getCharge(int daysRented) 로 이동
  - Rental.getCharge()는 남겨두고 , 그 구현은 delegation으로

Refactoring 적용-Move Method -2

```
class Rental …
  public double getCharge()
  {
    double result = 0;
    switch (getMovie().getPriceCode()) {
      case Movie.REGULAR:
        result += 2;
        if (getDaysRented() > 2)
          result += (getDaysRented()-2) * 1.5;
        break;
       …
    }
    return result ;
  }
}
```

```
class Movie …
  public double getCharge(int daysRented)
  {
    double result = 0;
    switch (getPriceCode()) {
      case Movie.REGULAR:
        result += 2;
        if (daysRented > 2)
          result += (daysRented-2) * 1.5;
        break;
       …
    }
    return result ;
  }
}
```

```
class Rental …
  public double getCharge()  {
     return _movie.getCharge(_daysRented);
  }
}
```

Move Method & delegation

Refactoring 적용-Move Method

- Rental.getFrequentRenterPoints() 에서
  - 앞에 것과 마찬가지로 외부 객체의 속성값에 기반하여 반환값을 결정하는 것은 좋지 못하다.
  - → Movie.getFrequentRenterPoints(int daysRented)
  - Rental.getFrequentRenterPoints()는 남겨두고, 그 구현은 delegation으로

Refactoring 적용-Move Method -3

```
class Rental …
   public int getFrequentRenterPoints() {
      if ((getMovie().getPriceCode() == Movie.NEW_RELEASE) && getDaysRented() > 1)
         return 2;
      else
         return 1;
   }
}
```

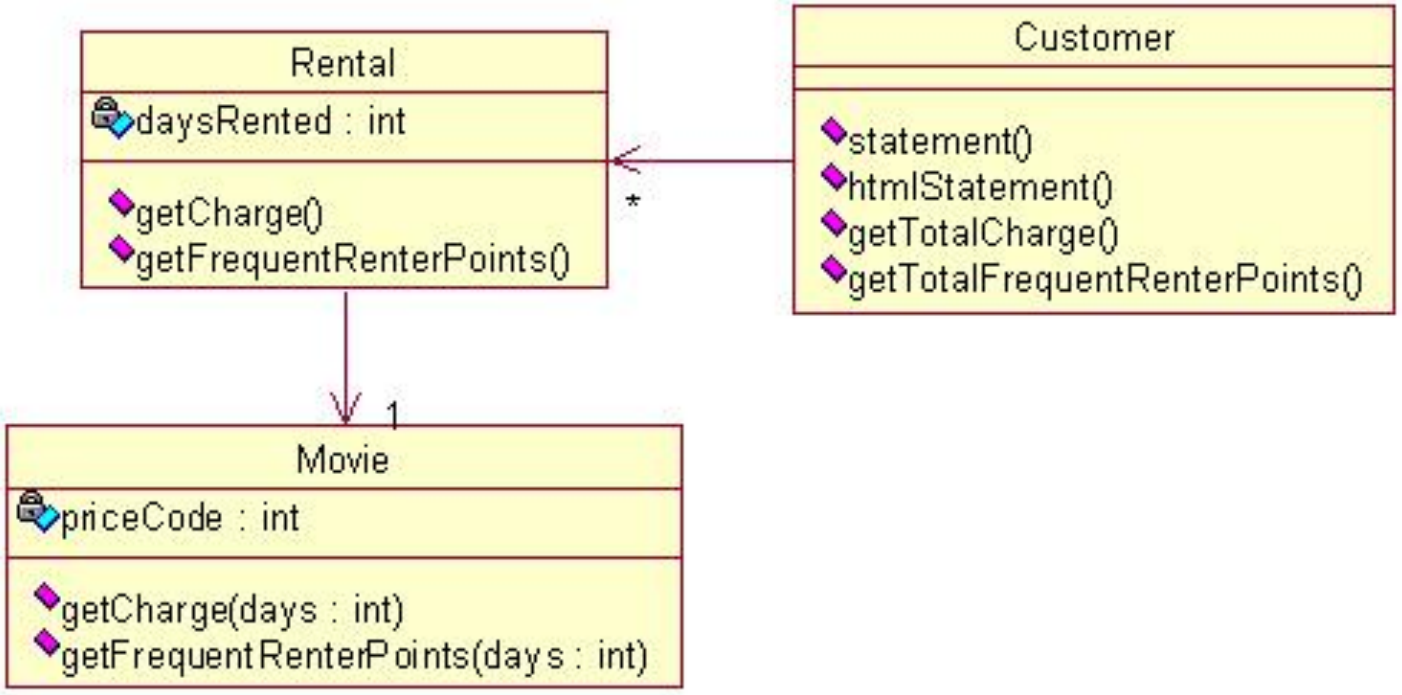
Move Method & delegation

```
class Rental …
   public int getFrequentRenterPoints() {
      return _movie.getFrequentRenterPoints(_daysRented);
   }
}

class Movie…
 public int getFrequentRenterPoints(int daysRented) {
      if ((getPriceCode() == Movie.NEW_RELEASE) && daysRented > 1)
         return 2;
      else
         return 1;
   }
}
```

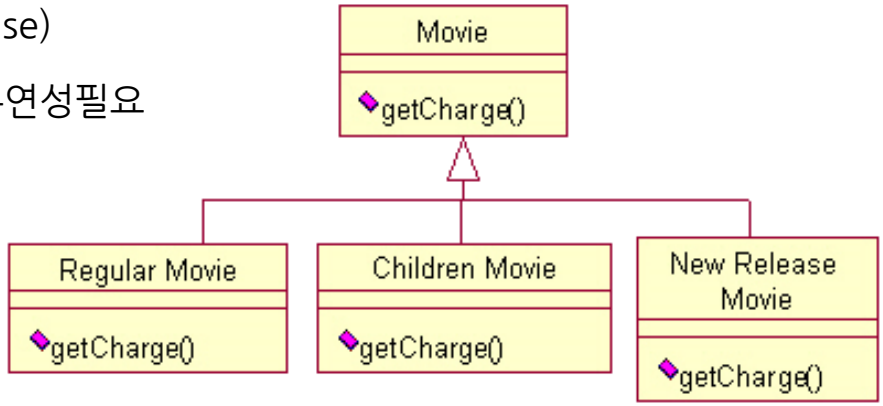
Refactoring 적용-Move Method -4

❖ [옮긴 후 Class diagram]



조건문을 다형성으로

- 영화 분류에 대하여…
  - 현재는 3가지(Regular, Children, New Release)
  - 바뀔 수 있으나, 아직 결정되지는 않았음. → 유연성필요
  - 각 종류별로 subclassing

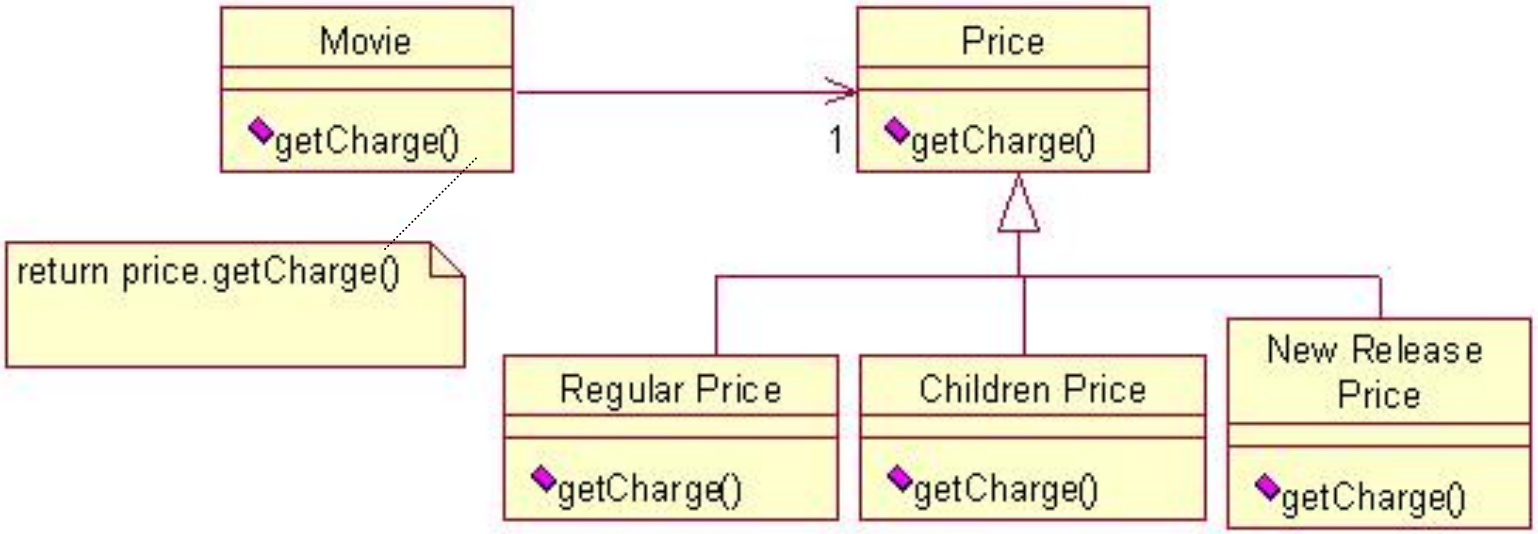


  - [문제점]
    - 한 비디오 객체가 생성될 때 종류가 고정되어 버림
    - 기간이 지나면 바뀌어야 되는 종류도 (예: New Release) class를 바꿀 수 없음.

조건문을 다형성으로 -2

- 객체가 동적으로 자신의 상태를 바꿀 수 있게 하려면?
  - → “State pattern 적용”

[Movie에 State pattern 사용]



조건문을 다형성으로 -3

- Movie에 State pattern 적용
  - priceCode 를 Price class 로 변환
    - Replace Type Code with State/Strategy
  - 2. switch문 있는 Movie.getCharge()를 Price class로 이동
    - Move Method
  - 3. Price.getCharge()에서 switch 문 제거
    - Replace Conditional with Polymorphism

[priceCode 관련]

```
class Movie…
  public Movie(String name, int priceCode) {
    _name = name;
    _priceCode = priceCode;
  }
  private int _priceCode;
```

조건문을 다형성으로 -4

1. priceCode 를 Price class 로 변환

```
class Movie…
  public Movie(String name, int priceCode) {
    _name = name;
    setPriceCode(priceCode);
  }
  public setPriceCode(int arg
    swithc(arg) {
      case REGULAR:
        _price = new RegularPrice();
        break;
      case CHILDRENS:
        _price = new ChildrensPrice();
        break;
      case NEW_RELEASE:
        _price = new NewReleasePrice();
        break;
       default:
        throw new IllegalArgumentException("..");
    }
  }
  public int getPriceCode() {
    return _price.getPriceCode();
  }
  private Price _price;
```

Replace Type Code with State/Strategy

기존 priceCode접근을 set/get 함수로

priceCode는 price class로

[Price classes]

```
abstract class Price {
  abstract int getPriceCode();
}
class ChildrensPrice extends Price {
  int getPriceCode() {
      return Movie.CHILDRENS;
  }
}
class NewReleasePrice extends Price {
  int getPriceCode() {
      return Movie.NEW_RELEASE;
  }
}
class RegularPrice extends Price {
  int getPriceCode() {
      return Movie.REGULAR;
  }
}
```

조건문을 다형성으로 -5

2. switch문 있는 Movie.getCharge()를 Price class로 이동

```
class Movie …
  public double getCharge(int daysRented)  {
    double result = 0;
    switch (getPriceCode()) {
      case Movie.REGULAR:
        result += 2;
        if (daysRented > 2)
          result += (daysRented-2) * 1.5;
        break;
       …
    }
    return result ;
  }
}
```

Move Method

```
class Movie …
  public double getCharge(int daysRented) {
    return _price.getCharge(daysRented);
  }
```

```
class Price… {
  public double getCharge(int daysRented) {
    double result = 0;
    switch (getPriceCode()) {
      case Movie.REGULAR:
        result += 2;
        if (daysRented > 2)
          result += (daysRented-2) * 1.5;
        break;
       …
    }
    return result ;
  }
```

조건문을 다형성으로 -6

3. Price.getCharge()에서 switch 문 제거

```
class Price…
  public double getCharge(int daysRented) {
    double result = 0;
    switch (getPriceCode()) {
      case Movie.REGULAR:
        result += 2;
        if (daysRented > 2)
          result += (daysRented-2) * 1.5;
        break;
      case Movie.NEW_RELEASE:
        result += daysRented*3;
        break;
      case Movie.CHILDRENS:
        result += 1.5;
        if (daysRented > 3)
          result += (daysRented-3)*1.5;
        break;
    }
    return result ;
  }
```

Replace Conditional with Polymorphism

```
class Price…
  abstract double getCharge(int daysRented);
```

```
class RegularPrice …
  public double getCharge(int daysRented) {
    double result = 2;
    if (daysRented > 2)
      result += (daysRented-2) * 1.5;
    return result;
  }
```

```
class NewReleasePrice …
  public double getCharge(int daysRented) {
    return daysRented*3;
  }
```

```
class ChildrensPrice…
  public double getCharge(int daysRented) {
    double result = 1.5;
    if (daysRented > 3)
      result += (daysRented-3) * 1.5;
    return result;
  }
```

조건문을 다형성으로 -7

- Movie.getFrequentRenterPoints()에서 if 문장을 다형성으로

```
class Movie…
  int getFrequentRenterPoints(int daysRented) {
    if ((getPriceCode()==Movie.NEW_RELEASE) && daysRented > 1 )
      return 2;
    else
      return 1;
  }
```

Replace Conditional with Polymorphism

```
class Movie…
  int getFrequentRenterPoints(int daysRented) {
    return _price.getFrequentRenterPoints(daysRented);
  }
```

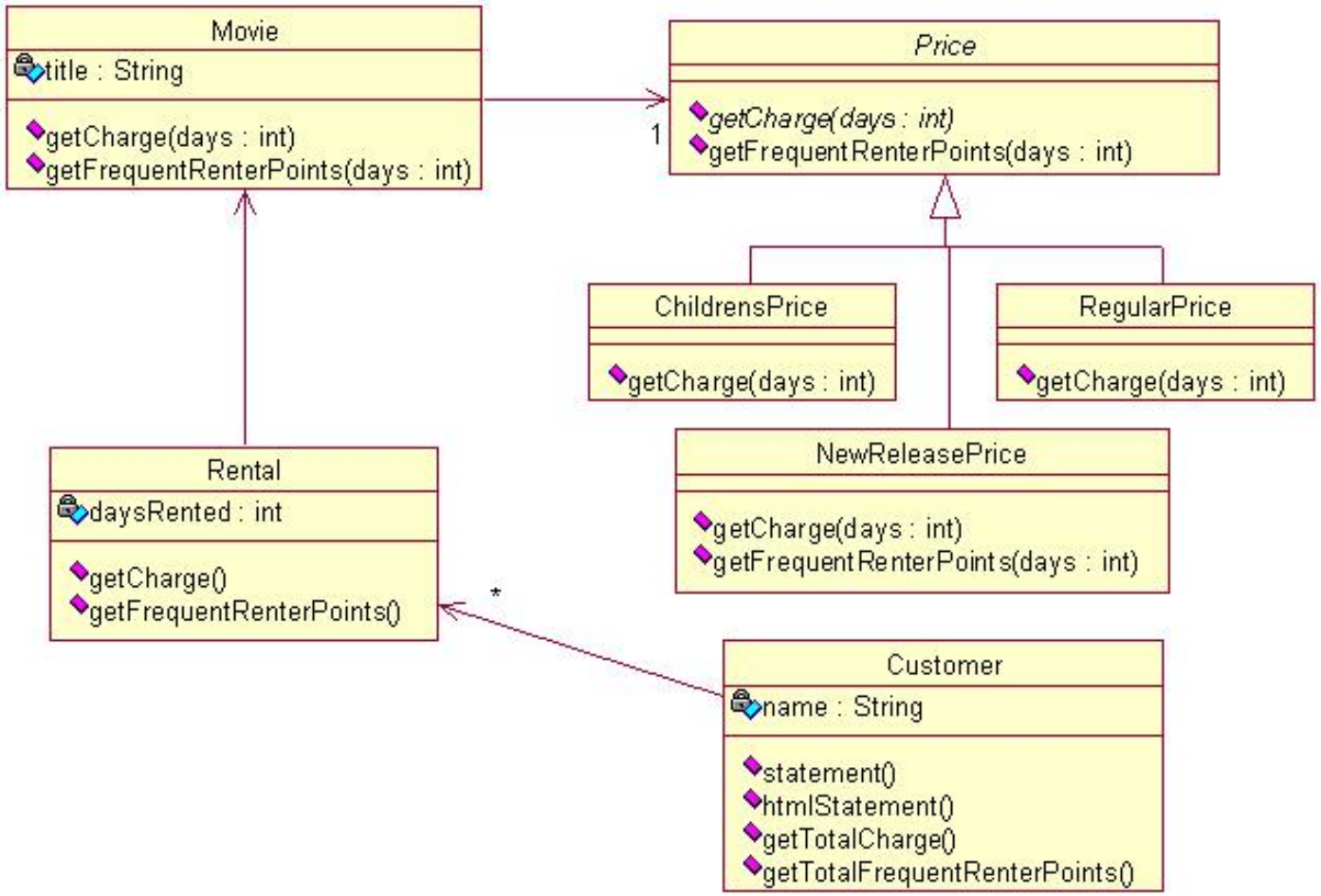
```
class Price…
  int getFrequentRenterPoints(int daysRented) {
    return 1;
  }

class NewReleasePrice…
   int getFrequentRenterPoints(int daysRented) {
     return (daysRented >1) ? 2: 1;
}
```

NewReleasePrice에만 method를 override

조건문을 다형성으로 -8

[State pattern을 적용한 후의 Class diagram]



조건문을 다형성으로 -9

[State pattern을 적용한 후의 Sequence diagram]