[우리 학과 취업스쿨 Day08]

소프트웨어 디자인 패턴과 리팩토링

리팩토링(Refactoring)

(프로그램의 가치를 높이는 코드 정리 기술)

데이터를 조직하기 관련 리팩토링

1. 속성을 캡슐화 하기(Self Encapsulate Field)
2. 데이터 값을 객체로 바꾸기(Replace Data Value with Object)
3. 값을 레퍼런스로 변경하기(Change Value to Reference)
4. 레퍼런스를 값으로 바꾸기 (Change Reference to Value)
5. 배열을 객체로 바꾸기
6. 관찰 데이터를 분리하기
7. 단방향 연관성을 양반향으로 변경하기
8. 양방향 연관성을 단방향으로 변경하기
9. Public 속성을 private로 캡슐화 하기
10. 컬렉션을 캡슐화 하기
11. 레코드를 데이터 클래스로 바꾸기
12. 타입코드를 클래스로 바꾸기
13. 타입코드를 하위 클래스들로 바꾸기
14. 하위클래스를 속성으로 바꾸기

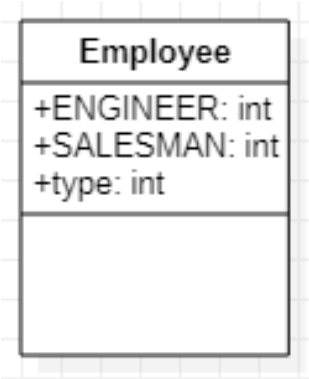
타입코드를 하위 클래스로 바꾸기

Replace Type Code with Subclasses

타입코드를 하위 클래스로 바꾸기

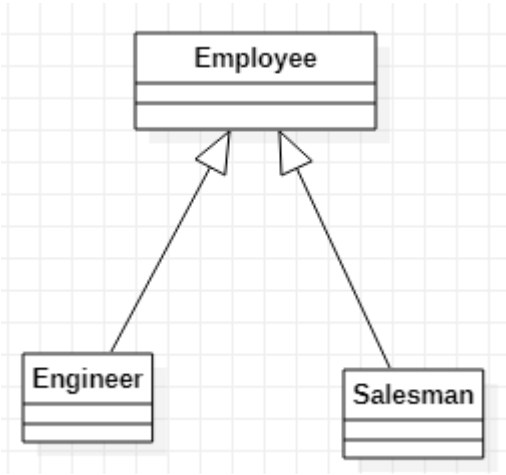
❖ 개요

- 특정 클래스의 속성이 변하지 않는 타입 코드로 **클래스의 행위에 영향을 줄 때 적용한다.**
- **타입 코드 대신에 하위 클래스를 정의하여 대체한다**.
- 예시



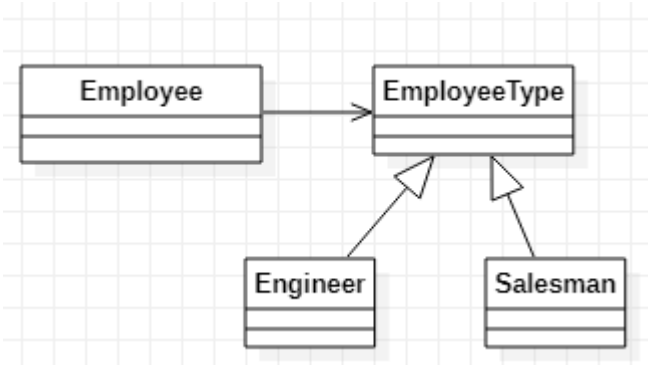
Employee 클래스의 **type속성은 직업의 종류를 나타낸다.**
이 값은 static으로 선언된 ENGINEER,SALESMA 중 하나이다.
Employee 객체가 생성될 때. 직업의 종류가 결정되고 나중에 바뀌는 일이 없으며, 이 클래스의 **메소드는 type속성의 값에 따라 행위가 달라진다고** 가정하자.

타입코드를 하위 클래스로 바꾸기



Type 값은 별도의 클래스로 정의할 수 있다.(확장성 고려)

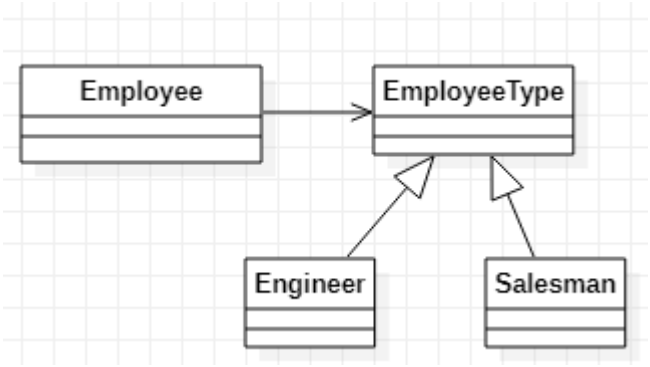
- 기존 Employee 클래스의 type 속성은 필요 없어 지며 이해하기 쉬워지고, 확장성도 좋다.



하지만, 반드시 타입값을 상징하는 **속성값에 따라 클래스의 행위가 변경되어야 한다면, 타입코드를** **State/Strategy 패턴으로 바꾸기도 가능**

Employee객체는 엔지니어, 세일즈맨으로 역할을 바꿀 수 있도록 state패턴처럼 객체의 상태(엔지니어, 세일즈맨)를 나타내는 state객체를 도입한다.

타입코드를 하위 클래스로 바꾸기



하지만, 반드시 타입값을 상징하는 **속성값에 따라 클래스의 행위가 변경되어야 한다면, 타입코드를** State/Strategy 패턴으로 바꾸기도 가능

Employee객체는 엔지니어, 세일즈맨으로 역할을 바꿀 수 있도록 state패턴처럼 객체의 상태(엔지니어, 세일즈맨)를 나타내는 state객체를 도입한다.
EmployeeType 이라는 클래스가 State패턴의 상태 패턴의 상태 객체를 상징한다.
기존의 Employee클래스의 type속성값에 따라 EmployeeType의 하위클래스를 정의한다.

조건문을 단순화 하기 관련 리팩토링

1. 조건문을 단순화 하기(Decompose Conditional)
2. 조건 표현식을 다듬기(Consolidate Conditional Expression)
3. 중복된 조건문을 다듬기(Consolidate Duplicate Conditional Fragement)
4. 컨트롤 플래그를 제거하기
5. 중복 조건문들을 가드로 바꾸기
6. 조건문을 폴리모르피즘으로 바꾸기
7. 널 객체를 도입하기

중복된 조건문 내부 코드로 다듬기

Consolidate Duplicate Conditional Fragement

중복된 조건문 내부 코드로 다듬기

❖ 개요

- 조건문 내부에 중복된 코드 조각이 존재하는 경우에 적용한다. 중복되는 부분을 조건문 밖으로 이동한다.

```
if(isSpecialDeal()){
   total = price * 0.95;
   send();
}
else{
    total = price * 0.98;
    send();
}
```

```
if(isSpecialDeal())
 total = price * 0.95;
else
    total = price * 0.98;
send();
}
```

조건문을 폴리모피즘으로 바꾸기

Replace Conditional with Polymorphism

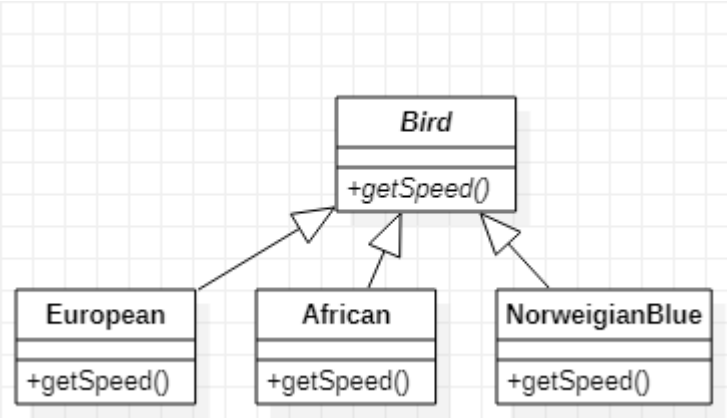
조건문을 폴리모피즘으로 바꾸기

❖ 개요

- 특정 메소드 내에 객체의 타입값에 따라 다른 행위를 선택하는 조건문이 있는 경우에 적용한다. 이 메소드를 추상 메소드로 선언하고 타입별 하위 클래스를 정의한 후 오버라이딩한다.

```
double getSpeed(){
    switch(type){
      case EUROPEAN:
           return getBaseSpeed();
      case AFRICAN:
           return getBaseSpeed()-getLoadFactor()*numberOfcoconuts;
      case NORWEIGIAN_BLUE:
           return (isNalied) ? 0 : getBaseSpeed(voltage);
}
throw new RuntimeException(Should be Unreachable");
```

조건문을 폴리모피즘으로 바꾸기




```
class European{
    @override
double getSpeed(){
           return getBaseSpeed();}
class AFRICAN{
    @override
double getSpeed(){
return getBaseSpeed()-getLoadFactor()*numberOfcoconuts;
}
class NORWEIGIAN_BLUE{
 @override
double getSpeed(){
return (isNalied) ? 0 : getBaseSpeed(voltage);
}
```

getSpeed()메소드는 Bird 클래스에 속해 있으며 swtich문의 type의 새 종류를 상징하는 속성이다.

이메소드의 switch 조건문은 새의 종류에 따라 속도를 다르게 계산한다. 즉 새의 타입이 EUROPEAN, AFRICAN, NORWEIGIAN_BLUE의 여부에 따라 getSpeed() 의 행위가 달라지게 되는 것이다. 이런 코드는 새로운 종류의 새가 추가될때 마다 getSpeed()내부의 코드를 변경하는 단점이 있다. 즉 OCP 위배된다.

예: 비디오 가게 고객 관리 프로그램

❖ 프로그램 기능

- 고객이 대여한 비디오 목록(영화제목, 종류, 대여기간)을 보여주고, 고객이 지불해야 하는 비용을 계산함
- 영화 종류: 일반, 아동, 최신
- 영화 종류에 따라 대여 비용의 차이가 있음
  - 일반: 기본 2000원 + 2일 경과 후 부터 1일 1500원
  - 아동: 기본 1500원 + 3일 경과 후 부터 1일 1500원
  - 최신: 대여일자 x 2000원
- 적립 포인트 : 영화 1편당 100원
  - 최신 영화는 2일 이상 대여할 경우 100원 추가 적립

이 프로그램의 문제는?

- 이 프로그램은 원하는 형태로 동작은 함. 그러면 충분한가?
  - 매우 단순한 프로그램일 경우에는 충분할 수 있음
  - 매우 큰 프로그램의 일부이면 유지보수 가능성, 가독성 등이 매우 중요함
- Customer클래스의 문제?
  - 웹으로 결과 출력을 원할 경우에는?
  - 비용계산 방법이 바뀔 경우?
  - 영화 종류가 추가 되면?

예제 프로그램 -2

[Movie.java]

```
public class Movie {
   public static final int CHILDREN =2;
   public static final int REGULAR =0;
   public static final int NEW_RELEASE =1;

   private String _title;
   private int _priceCode;

   public Movie(String title, int priceCode) {
      _title = title;
      _priceCode = priceCode;
   }
   public int getPriceCode() {
      return _priceCode;
   }
   public void setPriceCode(int arg) {
      _priceCode = arg;
   }
   public String getTitle() {
      return _title;
   }
}
```

[Rental.java]

```
class Rental {
   private Movie _movie;
   private int _daysRented;

   public Rental(Movie movie, int daysRented) {
      _movie = movie;
      _daysRented = daysRented;
   }
   public int getDaysRented() {
      return _daysRented;
   }
   public Movie getMovie() {
      return _movie;
   }
}
```

예제 프로그램 -3

[Customer.java]

```
class Customer {
   private String _name;
   private Vetor _rentals = new Vector();

   public Customer(String name) {
      _name = name;
   }
   public void addRental(Rental arg) {
      _rentals.addElement(arg);
   }
   public String getName() {
      return _name;
   }
   public String statement() {
      double totalAmount =0;
      int frequentRenterPoints =0;
      Enumeration rentals = _rentals.elements();
      String result = "Rental Record for " + getName() + "\n";
      while ( rentals.hasMoreElements() ) {
         double thisAmount = 0;
         Rental each = (Rental) rentals.nextElement();

         // 각 영화에 대한 요금 결정
         switch (each.getMovie().getPriceCode()) {
            case Movie.REGULAR:
               thisAmount += 2;
               if (each.getDaysRented() > 2)
                  thisAmount += (each.getDaysRented()-2) * 1.5;
               break;
      // Cont'd
```

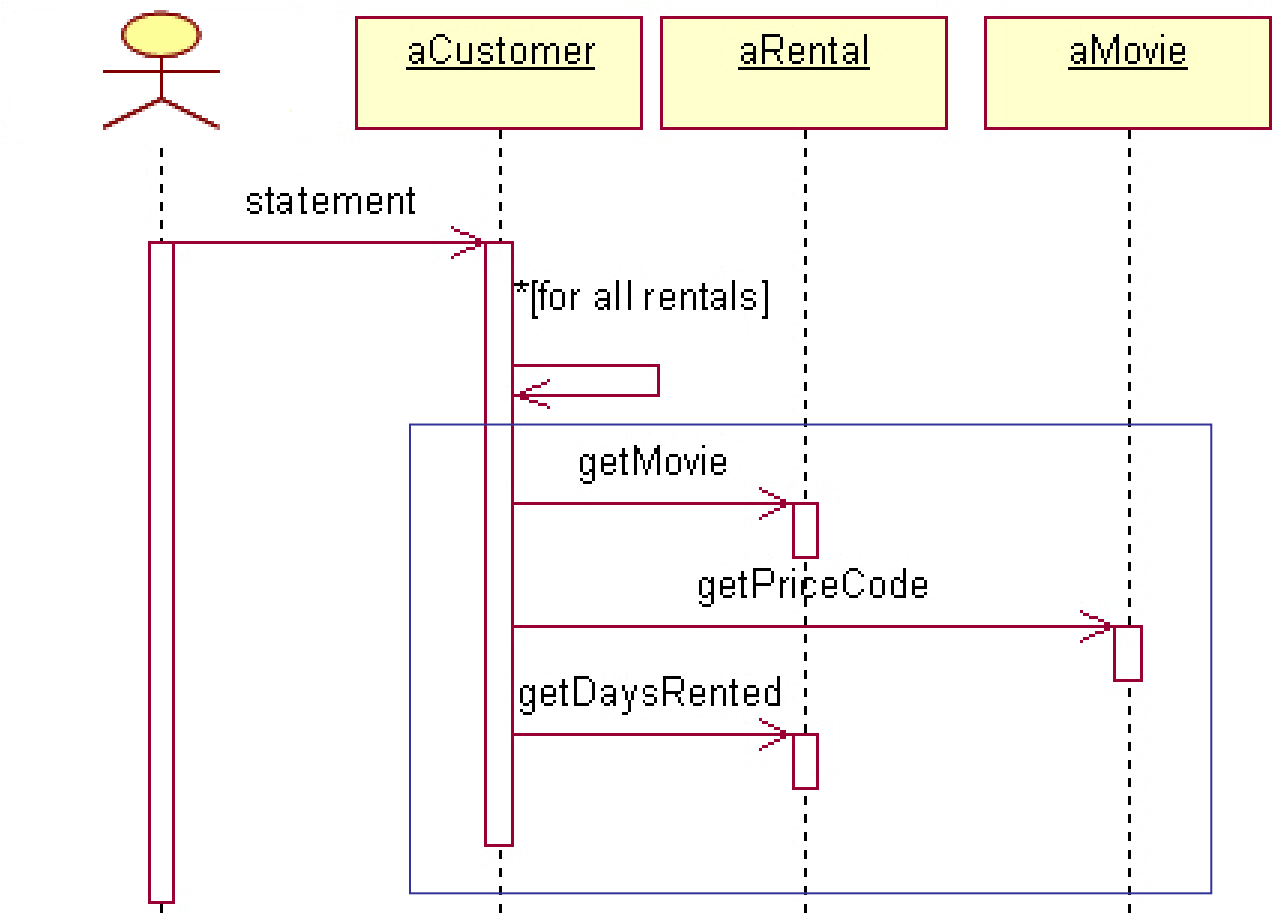
예제 프로그램 -4

[Customer.java]

```
            case Movie.NEW_RELEASE:
                  thisAmount += each.getDaysRented() * 3;
                  break;
            case Movie.CHILDRENS:
                  thisAmount += 1.5;
                  if (each.getDaysRented() > 3 )
                     thisAmount += (each.getDaysRented()-3)*1.5;
                  break;
            }

            // 포인터(frequent renter points) 추가
            frequentRenterPoints++;

            // 최신비디오를 이틀이상 대여하는 경우 추가 포인트 제공
            if ( (each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
                  each.getDaysRented() > 1)
               frequentRenterPoints++;

            // 대여에 대한 요금 계산결과 표시
             result += "\t" + each.getMovie().getTitle() + "\t" +
               String.valueOf(thisAmount) +"\n";
             totalAmount += thisAmount;
         }
         // 꼬리말 달기
         result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
         result += "You earned " + String.valueOf(frequentRenterPoints) +
            "frequent renter points";
         return result;
      }
}
```

예제 프로그램 -5

[statement()'s sequence diagram]



예제 프로그램 -6

❖ 프로그램에 대한 Comment

- 객체지향적이지 않다.
  - Customer class의 함수가 너무 길다.
- 요구사항 변경에 제대로 대처할 수 있는가?
  - 계산서가 HTML로 출력되어 웹에서도 볼 수 있기를 바란다면?
  - 요금 계산 방법이 바뀐다면?
  - 영화 분류하는 방법을 변경하고 싶다면?

TIP

새로운 기능을 추가하기 쉽도록 프로그램이 구조화 되어 있지 않다면 먼저 Refactoring을 해서 재구조화 한 다음 기능을 추가한다.

Refactoring 시작

- Refactoring 할 부분의 코드에 대한 견고한 Test set을 작성
  - Refactoring은 기능을 변경하는 것이 아니다.
    - 기존 기능이 유지되는지 확인 필요
  - statement() testing 예.
    - 두세 명의 고객을 만들고, 각각의 고객이 여러 영화를 빌리게 한 다음 계산서 문자열을 생성한다.
    - 생성된 문자열이 미리 확인한 기준 문자열과 비교한다.
    - 프로그램이 테스트를 확인하도록 작성 (self-checking)

TIP

리팩토링을 시작하기 전에 견고한 테스트 세트를 가지고 있는지 확인하라. 이 테스트는 자체 검사 여야 한다.

Refactoring 적용

- Refactoring 목표 설정
  - 지나치게 긴 statement()를 분해하는 것이 첫 목표
  - 중복을 최소화하면서 HTML statement() 를 만들기 쉽도록 하는 것
- Extract method 적용
  - 논리적으로 연관이 있는 코드 덩어리를 외부 method로 추출
  - statement() 내의 switch 문장
  - 지역변수와 함수인자에 주의
    - each: 값이 수정되지 않으므로 함수인자로 넘길 수 있다.
    - thisAmount: 값이 수정된다. 변수가 하나일땐 반환하면 된다.

TIP

리팩토링은 작은 단계로 나누어 프로그램을 변경한다.
실수를 하게 되더라도 쉽게 버그를 찾을 수 있다.

Refactoring 적용-Extract Method

```
class Customer {
   …
   public String statement() {
      …
      while ( rentals.hasMoreElements() ) {
         double thisAmount = 0;
         Rental each = (Rental) rentals.nextElement();

         // 각 영화에 대한 요금 결정
         switch (each.getMovie().getPriceCode()) {
            case Movie.REGULAR:
               thisAmount += 2;
               if (each.getDaysRented() > 2)
                  thisAmount += (each.getDaysRented()-2) * 1.5;
               break;
            case Movie.NEW_RELEASE:
               thisAmount += each.getDaysRented() * 3;
               break;
            case Movie.CHILDRENS:
               thisAmount += 1.5;
               if (each.getDaysRented() > 3 )
                  thisAmount += (each.getDaysRented()-3)*1.5;
               break;
         }
         …
      }
      …
   }
}
```

Extract Method

```
class Customer {
   …
   public String statement() {
      …
      while ( rentals.hasMoreElements() ) {
         double thisAmount = 0;
         Rental each = (Rental) rentals.nextElement();

         thisAmount = amountFor(each);
         …
      }
      …
   }
   private int amountFor(Rental each) {
   {
      double thisAmount = 0;
      switch (each.getMovie().getPriceCode()) {
           case Movie.REGULAR:
                …
           case Movie.CHILDRENS:
                …
      }
      return thisAmount;
   }
}
```

Refactoring 적용-Extract Method -2

❖ 새로 만든 amountFor() 의 내부 변수 이름 바꾸기

```
private double amountFor(Rental each)
{
   double thisAmount = 0;
   switch (each.getMovie().getPriceCode()) {
      case Movie.REGULAR:
         thisAmount += 2;
         if (each.getDaysRented() > 2)
            thisAmount += (each.getDaysRented()-2) * 1.5;
         break;
      case Movie.NEW_RELEASE:
         thisAmount += each.getDaysRented() * 3;
         break;
      case Movie.CHILDRENS:
         thisAmount += 1.5;
         if (each.getDaysRented() > 3 )
            thisAmount += (each.getDaysRented()-3)*1.5;
         break;
   }
   return thisAmount;
}
```

```
private double amountFor(Rental aRental)
{
   double result = 0;
   switch (aRental.getMovie().getPriceCode()) {
      case Movie.REGULAR:
         result += 2;
         if (aRental.getDaysRented() > 2)
            result += (aRental.getDaysRented()-2) * 1.5;
         break;
      case Movie.NEW_RELEASE:
         result += aRental.getDaysRented() * 3;
         break;
      case Movie.CHILDRENS:
         result += 1.5;
         if (aRental.getDaysRented() > 3 )
           result += (aRental.getDaysRented()-3)*1.5;
         break;
   }
   return result ;
}
```

TIP

컴퓨터가 이해할 수 있는 코드는 어느 바보나 다 짤 수 있다

좋은 프로그래머는 사람이 이해할 수 있는 코드를 짠다

Refactoring 적용-Move Method

- amountFor() 를 Customer에서 Rental로 이동
  - amountFor()가 사용하는 정보는 Rental class 에 있는 정보
  - 대부분의 경우 method는 그것이 사용하는 data가 있는 class에 있어야 한다.

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

Move Method

- aRental 인자삭제
- 이름 변경

```
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 ;
   }
}
```

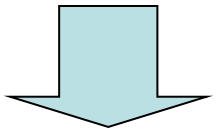
Refactoring 적용-Move Method -2

- 변경 요소
  - 함수이름 변경: amountFor() → getCharge()
  - 함수인자 삭제: amountFor(Rental aRental) → getCharge()
- Move Method 가 제대로 되었는지 확인
  - 1. Customer의 amountFor() 를 다음과 같이 변경

```
class Customer…
   private double amountFor(Rental aRental) {
      return aRental.getCharge();
   }
…
```

  - 2. 컴파일과 테스트를 해보고 잘못된 부분이 없는지 확인
  - 3. 기존에 Customer.amountFor() 사용하는 코드를 Rental.getCharge() 로 수정
  - 4. Customer.amountFor() 삭제

Refactoring 적용-Move Method -3



[Move Method 후의 클래스 상태]

Refactoring 적용-Replace Temp with Query

- Customer.statement() 내의 thisAmount 임시변수 제거
  - thisAmount 를 Rental.getCharge() 로 대체
  - 임시변수는 가능하면 제거하는 것이 좋다.
  - 일반적으로 performance 측면에서는 손해지만, 코드가 적절히 분해되어진다면 향후 최적화할 때 유리하다.

Refactoring 적용-Replace Temp with Query -2

```
public String statement() {
   double totalAmount =0;
   int frequentRenterPoints =0;
   Enumeration rentals = _rentals.elements();
   String result = "Rental Record for " + getName() + "\n";
   while ( rentals.hasMoreElements() ) {
      double thisAmount = 0;
      Rental each = (Rental) rentals.nextElement();

      thisAmount = each.getCharge();

      // 포인터(frequent renter points) 추가
      frequentRenterPoints++;
      // 최신비디오를 이틀이상 대여하는 경우 추가 포인트 제공
      …

      // 대여에 대한 요금 계산결과 표시
      result += "\t" + each.getMovie().getTitle() + "\t" +
                String.valueOf(thisAmount) +"\n";
      totalAmount += thisAmount;
   }
   ..
}
```

Replace Temp with Query

```
public String statement() {
   double totalAmount =0;
   int frequentRenterPoints =0;
   Enumeration rentals = _rentals.elements();
   String result = "Rental Record for " + getName() + "\n";
   while ( rentals.hasMoreElements() ) {
      Rental each = (Rental) rentals.nextElement();

      // 포인터(frequent renter points) 추가
      frequentRenterPoints++;
      // 최신비디오를 이틀이상 대여하는 경우 추가 포인트
      …

      // 대여에 대한 요금 계산결과 표시
      result += "\t" + each.getMovie().getTitle() + "\t" +
                String.valueOf(each.getCharge()) +"\n";
      totalAmount += each.getCharge();
   }
   ..
}
```

Refactoring 적용-Extract Method

- statement() 의 포인트 계산부분 추출
  - swtich 문장을 뽑아내어 Rental.getCharge()로 만든 방법대로, 포인트 계산부분을 Rental.getFrequentRenterPoints() 로 추출

```
public String statement() {
    …
   while ( rentals.hasMoreElements() ) {
      Rental each = (Rental) rentals.nextElement();

      // 포인터(frequent renter points) 추가
      frequentRenterPoints++;
      // 최신비디오를 이틀이상 대여하는 경우 추가 포인트 제공
      if ( (each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
            each.getDaysRented() > 1)
         frequentRenterPoints++;

      // 대여에 대한 요금 계산결과 표시
      result += "\t" + each.getMovie().getTitle() + "\t" +
          String.valueOf(each.getCharge()) +"\n";
      totalAmount += each.getCharge();
   }
   …
}
```

Extract Method &
Move Method

Refactoring 적용-Extract Method -2

```
class Customer ...
   public String statement() {
      double totalAmount =0;
      int frequentRenterPoints =0;
      Enumeration rentals = _rentals.elements();
      String result = "Rental Record for " + getName() + "\n";
      while ( rentals.hasMoreElements() ) {
         Rental each = (Rental) rentals.nextElement();
         frequentRenterPoints += each.getFrequentRenterPoints();
         // 대여에 대한 요금 계산결과 표시
         result += "\t" + each.getMovie().getTitle() + "\t" +
            String.valueOf(each.getCharge()) +"\n";
         totalAmount += each.getCharge();
      }
      ...
   }
```

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

Refactoring 적용-Replace Temp with Query

- Customer.statement()에서 totalAmount, frequentRenterPoints 임시 변수 제거
  - 임시변수를 query method로 대체
    - totalAmount → Customer.getTotalCharge()
    - frequentRenterPoints → Customer.getTotalFrequentRenterPoints()
  - loop 안에서 사용되는 임시변수 값을 계산하기 위하여 query method 도 loop를 복사하여야 한다.

Refactoring 적용- Replace Temp with Query -2

```
class Customer ...
  public String statement() {
    …
    while ( rentals.hasMoreElements() ) {
      Rental each = (Rental) rentals.nextElement();

      frequentRenterPoints += each.getFrequentRenterPoints();

      // 대여에 대한 요금 계산결과 표시
      result += "\t" + each.getMovie().getTitle() + "\t" +
          String.valueOf(each.getCharge()) +"\n";
      totalAmount += each.getCharge();
    }
    // 꼬리말 달기
    result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
    result += "You earned " + String.valueOf(frequentRenterPoints) +
      "frequent renter points";
    return result;
  }
```

Replace Temp with Query

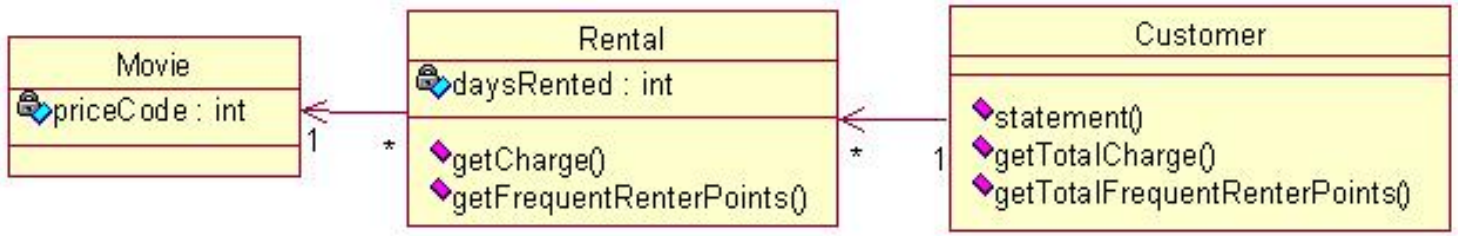
```
er ...
g statement() {

tals.hasMoreElements() ) {
ach = (Rental) rentals.nextElement();

대한 요금 계산결과 표시
+"\t" + each.getMovie().getTitle() + "\t" +
g.valueOf(each.getCharge()) +"\n";

// 꼬리말 달기
result += "Amount owed is " +
      String.valueOf(getTotalCharge()) + "\n";
result += "You earned " + String.valueOf(
      getTotalFrequentRenterPoints()) +
      "frequent renter points";
return result;
}
```

Refactoring 적용-Replace Temp with Query -3

```
class Customer...
  private double getTotalCharge() {
    double result =0;
    Enumeration rentals = _rentals.elements();
    while ( rentals.hasMoreElements()) {
      Rental each = (Rental) rentals.nextElement();
      result += each.getCharge();
    }
    return result;
  }
  private int getTotalFrequentRenterPoints() {
    int result =0;
    Enumeration rentals = _rentals.elements();
    while(rentals.hasMoreElements()) {
      Rental each= (Rental) rentals.nextElement();
      result += each.getFrequentRenterPoints();
    }
    return result;
  }
```

Refactoring 적용-Replace Temp with Query -4

❖ [변경된 Class diagram]



[변경된 Sequence diagram]