

CMPUT 301: Lecture 14 Refactoring

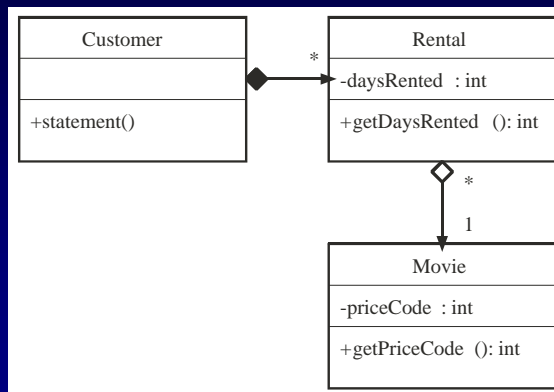
Lecturer: Sherif Ghali
Department of Computing Science
University of Alberta

Notes credits:
Ken Wong, Sherif Ghali
(Some slides removed by M. Hicks)

Refactoring

- Example:
 - a program to calculate and print a statement of a customer's charges at a video store

Class Diagram



3

Movie

```
public class Movie {
    public static final int CHILDRENS = 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;
    }
}
```

4

Rental

```
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;
    }
}
```

5

Customer

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

    public Customer( String name ) {
        _name = name;
    }
    public void addRental( Rental arg ) {
        _rentals.addElement( arg );
    }
    public String getName() {
        return _name;
    }
    ...
}
```

6

Customer::statement()

```
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();

        // determine amounts for each line
        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;
        }
        ...
    }
}
```

7

Customer::statement()

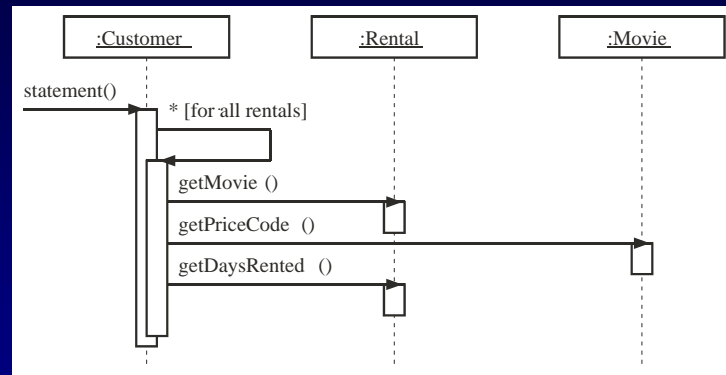
```
// add frequent renter points
frequentRenterPoints++;
// add bonus for a two day new release rental
if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
    each.getDaysRented() > 1) frequentRenterPoints++;

// show figures for this rental
result += "\t" + each.getMovie().getTitle() + "\t" +
    String.valueOf( thisAmount ) + "\n";
totalAmount += thisAmount;
}

// add footer lines
result += "Amount owed is " + String.valueOf( totalAmount ) + "\n";
result += "You earned " + String.valueOf( frequentRenterPoints ) +
    " frequent renter points";
return result;
}
}
```

8

Refactoring



9

Refactoring

- *Something is rotten in the state of Denmark*
 - What is it?

10

Refactoring

- Issues:
 - not object-oriented
 - statement() routine does too much
 - Customer class is a **blob**
 - potentially difficult to make changes
 - e.g., HTML output
 - e.g., new charging rules

11

Refactoring

- Idea:
 - If the code is not structured conveniently to add a feature, first **refactor** the program to make it easy to add the feature, then add the feature.

12

Refactoring

- First step:
 - Build self-checking tests.

13

Refactoring

- Decompose `statement()` method:
 - Extract logical chunk of code as a new method.
 - Apply Extract Method.

14

```

• 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()) {
            double thisAmount = 0;
            Rental each = (Rental)rentals.nextElement();

            // determine amounts for each line
            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;
            }
        }
    }
    ...
}

```

15

Refactoring

```

• 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()) {
            double thisAmount = 0;
            Rental each = (Rental)rentals.nextElement();

            thisAmount = amountFor( each );

            ...
        }
    }
}

```

16

Refactoring

```
• class Customer {  
    ...  
    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;  
    }  
    ...  
}
```

17

Refactoring

- Compile and test!
– small steps
- What do we do next?

18

Refactoring

- Rename variables in `amountFor()`:
 - Enhance readability.

19

Refactoring

```
• class Customer {  
    ...  
    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;  
    }  
    ...  
}
```

20

Refactoring

```
• 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;  
            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;  
    }  
    ...  
}
```

21

Refactoring

- Compile and test.
- What do we do next?

22

Refactoring

- Move `amountFor()` to `Rental` class:
 - Method uses `rental` information, but not `customer` information.
 - Move method to the right class.
 - Apply Move Method.

23

Refactoring

```
• 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;  
            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;  
    }  
    ...  
}
```

24

Refactoring

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

25

Refactoring

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

26

Refactoring

- Compile and test.

27

Refactoring

- Replace references to `amountFor()` with `getCharge()`:
 - Adjust references to old method to use new method.
 - Remove old method.

28

Refactoring

```
• 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()) {  
            double thisAmount = 0;  
            Rental each = (Rental)rentals.nextElement();  
  
            thisAmount = amountFor( each );  
  
            ...  
        }  
    }  
}
```

29

Refactoring

```
• 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()) {  
            double thisAmount = 0;  
            Rental each = (Rental)rentals.nextElement();  
  
            thisAmount = each.getCharge();  
  
            ...  
        }  
    }  
}
```

30

Refactoring

- Compile and test.

31

Refactoring

- Eliminate `thisAmount` temporary in `statement()`:
 - Replace redundant temporary variable with query.
 - Apply Replace Temp with Query.

32

Refactoring

```
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()) {
            double thisAmount = 0;
            Rental each = (Rental)rentals.nextElement();

            thisAmount = each.getCharge();

            // add frequent renter points
            frequentRenterPoints++;
            // add bonus for a two day new release rental
            if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
                each.getDaysRented() > 1) frequentRenterPoints++;

            // show figures for this rental
            result += "\t" + each.getMovie().getTitle() + "\t" +
                String.valueOf( thisAmount ) + "\n";
            totalAmount += thisAmount;
        }
        ...
    }
}
```

33

Refactoring

```
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();

            // add frequent renter points
            frequentRenterPoints++;
            // add bonus for a two day new release rental
            if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
                each.getDaysRented() > 1) frequentRenterPoints++;

            // show figures for this rental
            result += "\t" + each.getMovie().getTitle() + "\t" +
                String.valueOf( each.getCharge() ) + "\n";
            totalAmount += each.getCharge();
        }
        ...
    }
}
```

34

Refactoring

- Extract frequent renter points logic:
 - Applicable rules belong to the rental, not the customer.
 - Apply Extract Method and Move Method.

35

Refactoring

```
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();

            // add frequent renter points
            frequentRenterPoints++;
            // add bonus for a two day new release rental
            if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
                each.getDaysRented() > 1) frequentRenterPoints++;

            // show figures for this rental
            result += "\t" + each.getMovie().getTitle() + "\t" +
                String.valueOf( each.getCharge() ) + "\n";
            totalAmount += each.getCharge();
        }
        ...
    }
}
```

36

Refactoring

```
• 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();  
  
            // add frequent renter points  
            frequentRenterPoints += each.getFrequentRenterPoints();  
  
            // show figures for this rental  
            result += "\t" + each.getMovie().getTitle() + "\t" +  
                String.valueOf( each.getCharge() ) + "\n";  
            totalAmount += each.getCharge();  
        }  
        ...  
    }  
}
```

37

Refactoring

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

38

Refactoring

- Eliminate `totalAmount` temporary:
 - Apply Replace Temp with Query.

39

Refactoring

```
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();

            // add frequent renter points
            frequentRenterPoints += each.getFrequentRenterPoints();

            // show figures for this rental
            result += "\t" + each.getMovie().getTitle() + "\t" +
                String.valueOf( each.getCharge() ) + "\n";
            totalAmount += each.getCharge();
        }

        // add footer lines
        result += "Amount owed is " + String.valueOf( totalAmount ) + "\n";
        result += "You earned " + String.valueOf( frequentRenterPoints ) +
            " frequent renter points";
        return result;
    }
}
```

40

Refactoring

```
class Customer {
    ...
    public String statement() {
        int frequentRenterPoints = 0;
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + getName() + "\n";

        while (rentals.hasMoreElements()) {
            Rental each = (Rental)rentals.nextElement();

            // add frequent renter points
            frequentRenterPoints += each.getFrequentRenterPoints();

            // show figures for this rental
            result += "\t" + each.getMovie().getTitle() + "\t" +
                String.valueOf( each.getCharge() ) + "\n";
        }

        // add footer lines
        result += "Amount owed is " + String.valueOf( getTotalCharge() ) + "\n";
        result += "You earned " + String.valueOf( frequentRenterPoints ) +
            " frequent renter points";
        return result;
    }
}
```

41

Refactoring

```
• 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;
    }
    ...
}
```

42

Refactoring

- Eliminate `frequentRenterPoints` temporary:
 - Apply Replace Temp with Query.

43

Refactoring

```
class Customer {
    ...
    public String statement() {
        int frequentRenterPoints = 0;
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + getName() + "\n";

        while (rentals.hasMoreElements()) {
            Rental each = (Rental)rentals.nextElement();

            // add frequent renter points
            frequentRenterPoints += each.getFrequentRenterPoints();

            // show figures for this rental
            result += "\t" + each.getMovie().getTitle() + "\t" +
                String.valueOf( each.getCharge() ) + "\n";
        }

        // add footer lines
        result += "Amount owed is " + String.valueOf( getTotalCharge() ) + "\n";
        result += "You earned " + String.valueOf( frequentRenterPoints ) +
            " frequent renter points";
        return result;
    }
}
```

44

Refactoring

```
class Customer {
    ...
    public String statement() {
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + getName() + "\n";

        while (rentals.hasMoreElements()) {
            Rental each = (Rental)rentals.nextElement();

            // show figures for this rental
            result += "\t" + each.getMovie().getTitle() + "\t" +
                String.valueOf( each.getCharge() ) + "\n";
        }

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

45

Refactoring

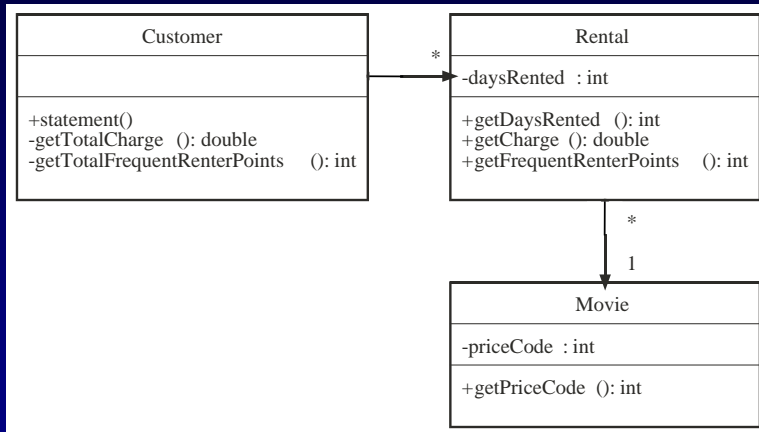
```
• class Customer {
    ...
    private int getTotalFrequentRenterPoints() {
        int result = 0;
        Enumeration rentals = _rentals.elements();

        while (rentals.hasMoreElements()) {
            Rental each = (Rental)rentals.nextElement();

            result += each.getFrequentRenterPoints();
        }
        return result;
    }
    ...
}
```

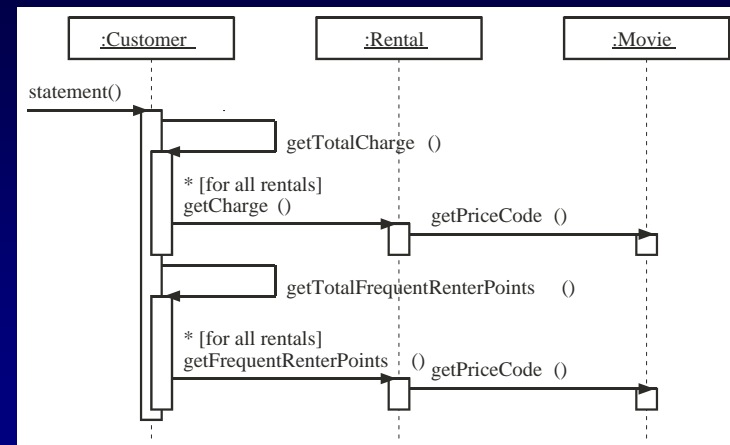
46

Refactoring

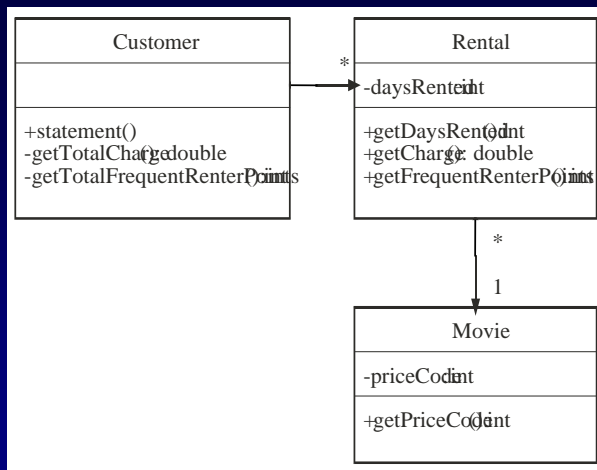


47

Refactoring



48



49

Refactoring

- Issues:
 - more code
 - slower performance?

50

Refactoring

- New feature (HTML output):

```
class Customer {
    ...
    public String htmlStatement() {
        Enumeration rentals = _rentals.elements();
        String result = "<H1>Rental Record for " + getName() + "</H1>\n";

        while (rentals.hasMoreElements()) {
            Rental each = (Rental)rentals.nextElement();

            // show figures for this rental
            result += each.getMovie().getTitle() + ": " +
                String.valueOf( each.getCharge() ) + "<BR>\n";
        }

        // add footer lines
        result += "<P>Amount owed is " +
            String.valueOf( getTotalCharge() ) + "</P>\n";
        result += "<P>You earned " +
            String.valueOf( getTotalFrequentRenterPoints() ) +
            " frequent renter points</P>";
        return result;
    }
}
```

51

Refactoring

- More needs:
 - new classifications of movies

52

Refactoring

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

53

Refactoring

- Replace conditional logic on price code with polymorphism:
 - Rental logic should not depend on specific movie types.
 - It is generally bad design to do a `switch` on another object's attribute.

54

Refactoring

```
• class Movie {  
    ...  
    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;  
    }  
    ...  
}
```

55

Refactoring

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

56

Refactoring

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

57

Refactoring

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

58

Refactoring

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

59

Refactoring

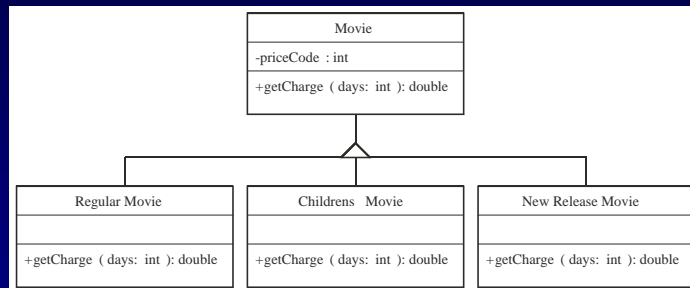
- Get rid of the switch statement:

```
class Movie {  
    ...  
    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;  
    }  
    ...  
}
```

60

Refactoring

- ?



61

Refactoring

- We have two flaws. What are they?
- Hint:
 - Is the movie classification static?

62

Refactoring

- Flaw:
 - A movie may change its classification during its lifetime.
 - An object cannot change its class during its lifetime.

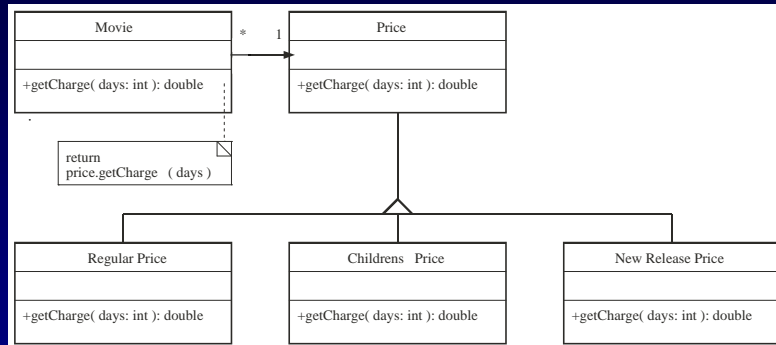
63

Refactoring

- Solution:
 - Use **State** design pattern.

64

Refactoring



65

Refactoring

- Replace price (type) code:
 - Apply Replace Type Code with State.
 - Compile and test after each step.

66

Refactoring

- Note:
 - Make sure uses of the price type code go through accessor methods ...

67

Refactoring

```
• public class Movie {  
    ...  
    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;  
    }  
    ...  
}
```

68

Refactoring

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

69

Refactoring

- Add new state classes:

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

70

Refactoring

- Replace price type codes with instances of price state classes ...

71

Refactoring

```
• public class Movie {  
    ...  
    private int _priceCode;  
    ...  
    public int getPriceCode() {  
        return _priceCode;  
    }  
    public void setPriceCode( int arg ) {  
        _priceCode = arg;  
    }  
    ...  
}
```

72

Refactoring

```
• public class Movie {  
    ...  
    private Price _price;  
    ...  
    public int getPriceCode() {  
        return _price.getPriceCode();  
    }  
    public void setPriceCode( int arg ) {  
        switch (arg) {  
            case REGULAR:  
                _price = new RegularPrice();  
                break;  
            case NEW_RELEASE:  
                _price = new NewReleasePrice();  
                break;  
            case CHILDRENS:  
                _price = new ChildrensPrice();  
                break;  
            default:  
                throw new IllegalArgumentException( "Incorrect price code" );  
        }  
    }  
}
```

73

Refactoring

- Move `getCharge()` to `Price` class:
 - Apply Move Method.

74

Refactoring

```
• class Movie {  
    ...  
    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;  
    }  
    ...  
}
```

75

Refactoring

```
• class Price {  
    ...  
    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;  
    }  
    ...  
}
```

76

Refactoring

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

77

Refactoring

- Replace switch statement in `getCharge()`:
 - For each case, add overriding method.
 - Define abstract method.
 - Apply Replace Conditional with Polymorphism.

78

Refactoring

```
• class Price {
    ...
    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;
    }
    ...
}
```

79

Refactoring

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

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

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

80

Refactoring

```
• class Price {  
  ...  
  abstract double getCharge( int daysRented );  
  ...  
}
```

81

Refactoring

- Move `getFrequentRenterPoints()` to `Price` class:
 - Apply Move Method.

82

Refactoring

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

83

Refactoring

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

84

Refactoring

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

85

Refactoring

- Replace if statement in
getFrequentRenterPoints():
 - Apply Replace Conditional with Polymorphism.

86

Refactoring

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

87

Refactoring

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

88

Refactoring

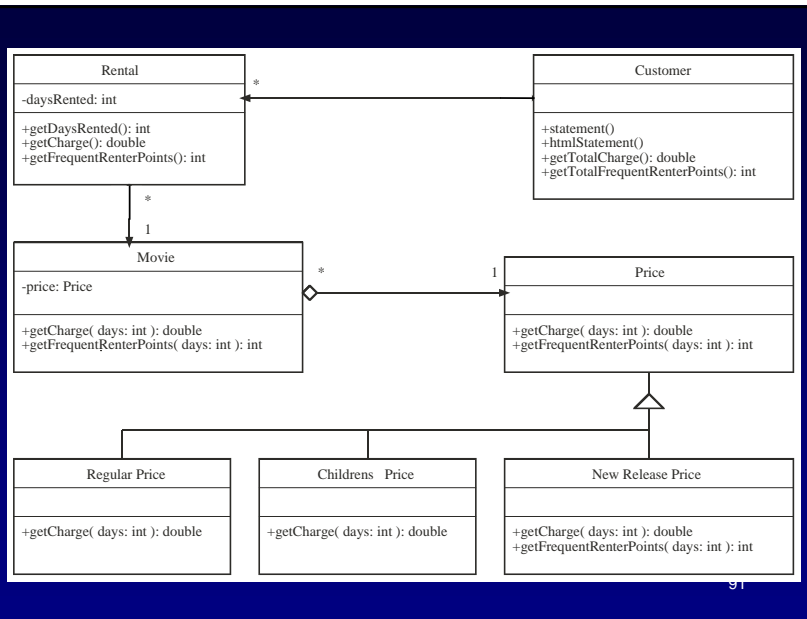
- Result of state design pattern:
 - easy to change price behavior
 - can add new price codes
 - rest of application does not know about this use of the state pattern

89

Refactoring

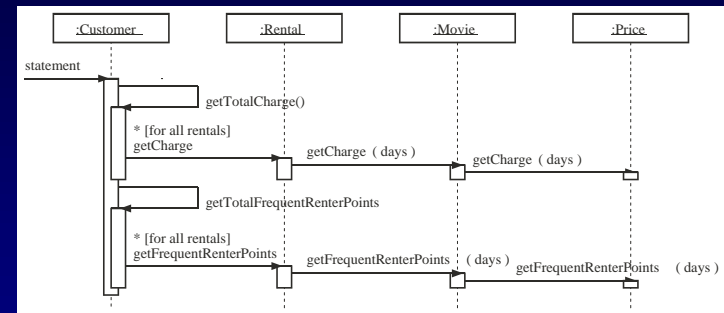
- Benefits of second refactoring:
 - easy to change movie classifications
 - easy to change rules for charging and frequent renter points

90



91

Refactoring



92

Bad Smells in Code

- Feature envy:
 - a method seems more interested in a class other than the one it is actually in
 - e.g., invoking lots of get methods
 - can use Move Method and Extract Method

93

Bad Smells in Code

- Data clumps:
 - groups of data appearing together in the fields of classes, parameters to methods, etc.
 - e.g., int x, int y, int z
 - move these groups into their own class
 - can use Extract Class and Introduce Parameter Object
 - Example: group (start: Date, end: Date) into (aRange: RangeDate)

94

Bad Smells in Code

- Primitive obsession:
 - using the built-in types of the language too much
 - reluctance to use small objects for small tasks
 - e.g., zip code string
 - use objects for individual data values
 - can use Replace Data Value with Object

95

Bad Smells in Code

- Switch statements:
 - consider using polymorphism instead
 - e.g., conditionals on type codes defined in other classes
 - can use Extract Method (on the switch), Move Method, Replace Type Code, and Replace Conditional with Polymorphism

96

Bad Smells in Code

- Speculative generality:
 - “I think we might need this someday.”
 - e.g., abstract classes without a real purpose
 - e.g., unused parameters
 - can use Collapse Hierarchy and Remove Parameter

97

Bad Smells in Code

- Message chains:
 - long chains of navigation to get to an object
 - e.g., client object talks to server object that delegates to another object that the client object must also know about
 - can use Hide Delegate

98

Bad Smells in Code

- Middle man:
 - a class that delegates many methods to another class
 - can use Remove Middle Man or Replace Delegation with Inheritance
 - but could be a legitimate adapter

99

Bad Smells in Code

- Don't stand so close:
 - two classes that depend too much on each other, with lots of bidirectional communication
 - separate the two classes
 - can use Move Method, Move Field, and Extract Class (factor out commonality)

100

Bad Smells in Code

- Alternative classes with different interfaces:
 - methods that do the same thing but have different signatures
 - e.g., put() versus add()
 - can use Rename Method

101

Bad Smells in Code

- Data class:
 - classes that are all data (manipulated by other classes)
 - e.g., a Point record that has other classes manipulating its coordinates
 - in early stages, it's all right to use public fields
 - study usage and move behavior into data classes
 - can use Encapsulate Field, Extract Method, Move Method

102

Bad Smells in Code

- Refused bequest:
 - when a subclass inherits something that is not needed
 - when a superclass does not contain truly common state/behavior
 - can use Push Down Method and Push Down Field
 - can use Replace Inheritance with Delegation (e.g., Square versus Rectangle)

103

Bad Smells in Code

- Comments:
 - often deodorant for bad smelling code
 - refactor code so that the comment becomes extraneous

104

References

- Refactoring
 - M. Fowler et al.
 - Addison-Wesley, 1999
- UML Distilled
 - M. Fowler
 - Addison-Wesley, 2000



This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.