**Code Smell #1:** Data Clumps

**Explanation:** The *Customer* class uses too many attributes from other classes.

**Example:**

* Rental each = (Rental) rentals.nexElement(); - Line# 33
* each.getMovie().getPriceCode(); - Line# 38 & 60
* each.getDaysRented(); - Line# 41
* MOVIE.REGULAR – Line# 39
* MOVIE.NEW\_RELEASE – Line# 45 & 60
* MOVIE.CHILDREN – Line# 48

Line 13 - 15

public void addRental(Rental arg) {

\_rentals.addElement(arg);

}

**Code Smell #2:** Long method

**Explanation**: statement() is a long method with too many tasks to accomplish.

**Example**: statement() is doing to the following tasks:

* Collecting movie rentals

Enumeration rentals = \_rentals.elements(); - Line#27

* Retrieving price code for each movie

switch (each.getMovie().getPriceCode()); - Line#38

* Calculating total rental price based on movie category –Line 38-54
* Calculating rental reward points -Line# 60-63

if ((each.getMovie().getPriceCode() == Movie.NEW\_RELEASE) && //Code Smell: Message Chaining.

(each.getDaysRented() > 1)) {

frequentRenterPoints++;

}

* Displaying the result summary –Line# 66-73

result += "\t" + each.getMovie().getTitle() + //Code Smell: Message Chaining

"\t" + String.valueOf(thisAmount) +

result += "Amount owed is " + String.valueOf(totalAmount) + "\n";

result += "You earned " + String.valueOf(frequentRenterPoints)

**Code Smell #3:** Feature Envy

**Explanation**: statement() requires too many information from other classes.

**Example**: Here’s a list of data that this method collects from the *Rental* and *Movie* classes:

*Rental*:

* Movie Objects -> (each.getMovie().getPriceCode() – Line# 38 & 60
* # of days rented -> each.getDaysRented() –Line# 41, 42, 46, 50, 51, & 61

*Movie*:

* Price Code -> each.getMovie().getPriceCode()) - Line# 38 & 60
* Movie Categories -> case Movie.REGULAR: -Line# 39
* Movie Titles -> each.getMovie().getTitle() – Line# 66

**Code Smell #4:** Switch Statement

**Explanation**: switch statement should be avoided as they are often duplicated in the code

**Example**:

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;

}

**Code Smell #5:** Meaningless variable name

**Explanation**: The variable name *each* is rather unclear. Need more meaning name. It could be renamed to rental.

**Example**: Rental each = (Rental) rentals.nextElement(); - Line# 33

**Code Smell #6:** Message Chaining

**Explanation**: The *Customer* class is calling *Rental* class and *Rental* class is calling *Movie* class for each rental item.

*Customer => Rental => Movie*

**Example**:

switch (each.getMovie().getPriceCode()) – Line# 33

**Code Smell #7:** Message Chaining

**Example**:

((each.getMovie().getPriceCode() == Movie.NEW\_RELEASE) - Line# 60

**Code Smell #8:** Message Chaining

**Example**:

result += "\t" + each.getMovie().getTitle() – Line# 66

**Code Smell #9:** Magic Number

**Explanation**: Using hard coded value is a bad practice as it is difficult to understand the business logic. It is also prone to error as there is always a possibility of generating out of range exception when *Movie* class with a wrong value.

**Example**: Movie Class:

* public static final int CHILDRENS = 2; -Line# 4

**Code Smell #10:** Magic Number

* public static final int REGULAR = 0; - Line# 5

**Code Smell #11:** Magic Number

* public static final int NEW\_RELEASE = 1; -Line #6

**Code Smell #12:** Data Class

**Explanation**: *Movie* class is a Data Class as it only contains fields, getter and setter methods for the fields and no behavior

**Example**: Line 16 - 27

public int getPriceCode() {

return \_priceCode;

}

public void setPriceCode(int arg) {

\_priceCode = arg;

}

public String getTitle() {

return \_title;

}

**Code Smell #13:** Unhandled Exception

**Explanation**: Potential for an out of range exception. Price code could be set with an invalid value. Also, anyone could exploit this code vulnerability with malicious intent.

**Example**: *Movie* Class, Line 21-23.

public void setPriceCode(int arg) {

\_priceCode = arg;

}

**Code Smell #14:** Data Class

**Explanation**: *Rental* class is a Data Class as it only contains fields, getter and setter methods for the fields and no behavior

**Example**: Line 12 – 18

public int getDaysRented() {

return \_daysRented;

}

public Movie getMovie() {

return \_movie;

}

**Code Smell #15:** Incomplete class

**Explanation**: *Rental* class is considered an incomplete class. This class is missing rental price calculation method.

**Code Smell #16:** Magic number

**Explanation**: *number 1.5 in the Customer class inside the switch statement on line 37 and line 46. This number has no obvious meaning. It’s harder to understand why choosing that equation.*

*Example:*

*Line 37:* thisAmount += (each.getDaysRented() - 2) \* 1.5;

Line 46: thisAmount += (each.getDaysRented() - 3) \* 1.5;

**Code Smell #17:** *Meaningless variable name*

**Explanation***: In the Customer class on line 22, the variable totalAmount does not explain what it’s for. It could be total rental price, total customer credit, or anything.*

*Example:*

*Line 22:* double totalAmount = 0;

**Code Smell #18:** Lazy class

**Explanation**: *The Rental class is not doing anything. The rental class could do the rental price calculation instead of the Customer class.*

*Example: N/A*

### Code Smell #19: Code Comments

**Explanation**: *In the Customer class on line 51-52 and line 66-96, these code are already self-explanatory.*

*Example:*

*Line 51-52*

// add frequent renter points  
 frequentRenterPoints++;

*Line 66-69*

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

### Code Smell #20: Large class

**Explanation**: The *Customer class are doing too many tasks: calculating rental price, calculating reward points, showing statement summary. It could be break down to smaller classes.*

*Example: N/A*