Aadhi Sivakumar

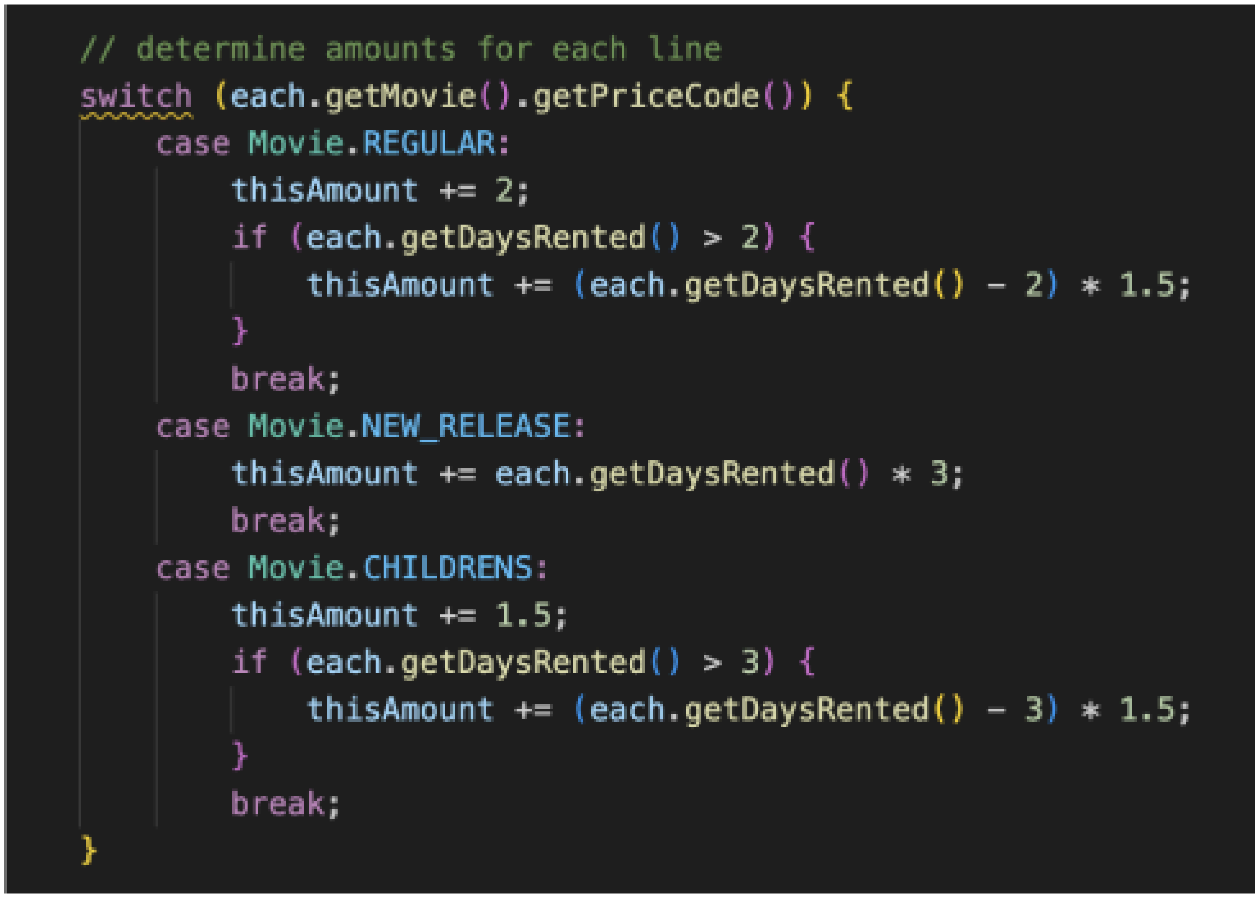
CS 3354.001

Homework #3

1)

- Public class Customer {} in the customer.java file is a code smell because it is a very large class, which reduces cohesion and has too many responsibilities/things to do.

2)



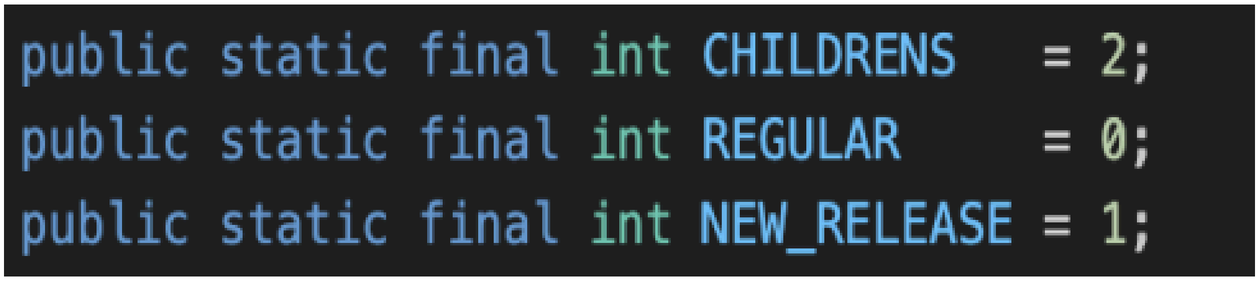
```java
// determine amounts for each line
switch (each.getMovie().getPriceCode()) {
    case Movie.REGULAR:
        thisAmount += 2;
        if (each.getDaysRented() > 2) {
            thisAmount += (each.getDaysRented() - 2) * 1.5;
        }
        break;
    case Movie.NEW_RELEASE:
        thisAmount += each.getDaysRented() * 3;
        break;
    case Movie.CHILDRENS:
        thisAmount += 1.5;
        if (each.getDaysRented() > 3) {
            thisAmount += (each.getDaysRented() - 3) * 1.5;
        }
        break;
}
```

- This piece of code uses switch statements, which is a code smell. Switch statements are often duplicated in code and should be replaced by the use of polymorphism.

3)

- Public String statement {} in Public class Customer {} is a code smell because it is a very long method which is difficult to understand. This method handles a lot including calculating rental amounts, frequent renter points, and formatting the statement string.

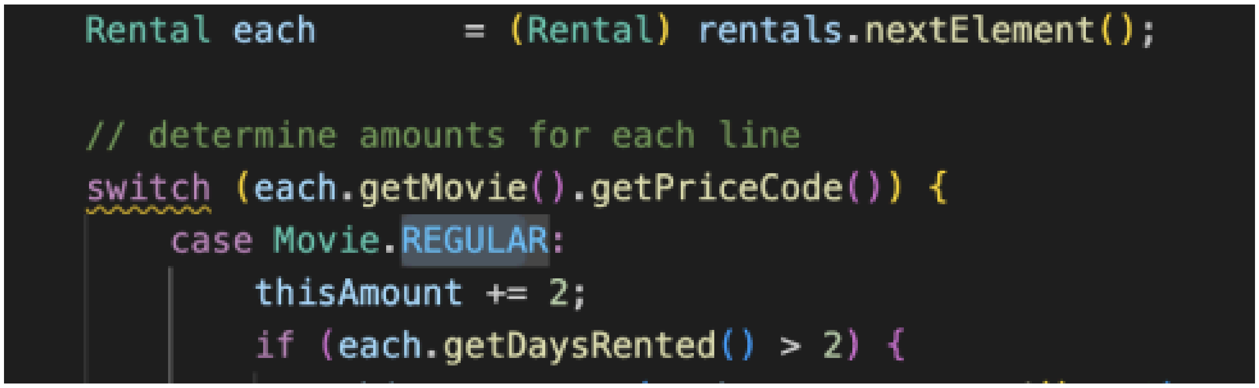
4)



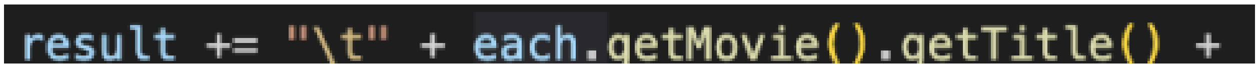
```
public static final int CHILDRENS   = 2;
public static final int REGULAR     = 0;
public static final int NEW_RELEASE = 1;
```

- This piece of code uses primitive integers (price codes) to represent movie types and is an example of primitive obsession. We can instead encapsulate these into objects and put them in a class.

5)



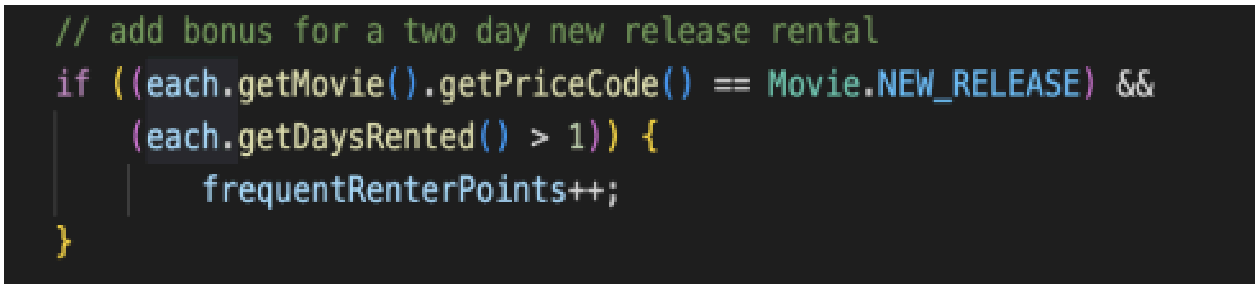
```
    Rental each       = (Rental) rentals.nextElement();

    // determine amounts for each line
    switch (each.getMovie().getPriceCode()) {
        case Movie.REGULAR:
            thisAmount += 2;
            if (each.getDaysRented() > 2) {
```



```
result += "\t" + each.getMovie().getTitle() +
```

- This piece of code is a code smell because each in statement method interacts and requires a lot of information from the Rental and Movie class's methods. This is an example of feature envy.

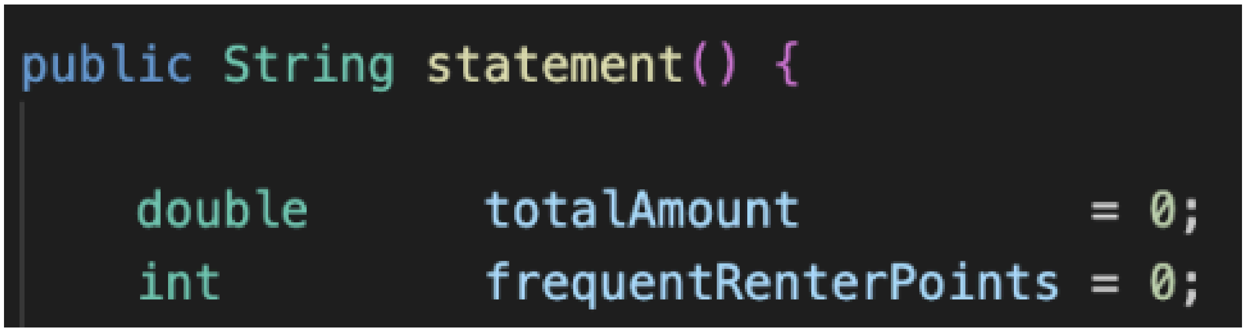
6)



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

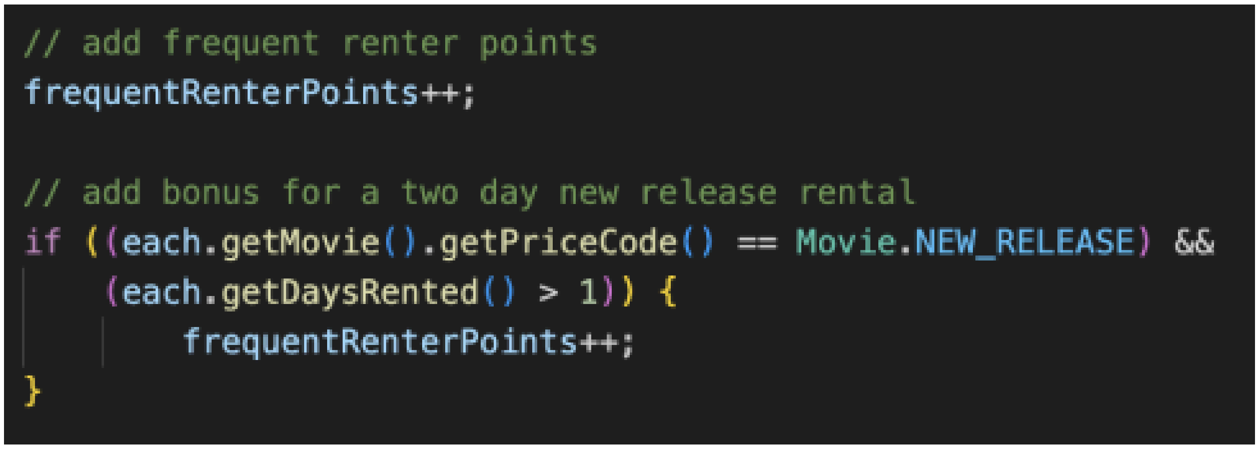
- This piece of code is a code smell because multiple attributes: price code and days rented, are frequently used together but are not part of the same class. Price code is part of the Movie class and days rented is part of the Rental class. This is an example of a data clump.

7)


```
public String statement() {

    double      totalAmount          = 0;
    int         frequentRenterPoints = 0;
```

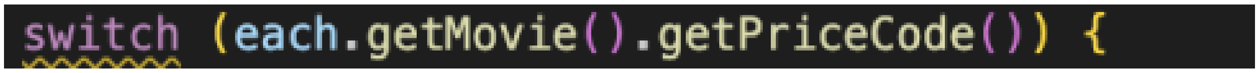
- totalAmount and frequentRenterPoints variables are code smells because they are an example of <u>temporary fields.</u> They exist only within the scope of the statement() method and serve just as accumulators for generating the rental statement. They do not do anything else outside of the specific method's execution.

8)


```
// 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++;
}
```

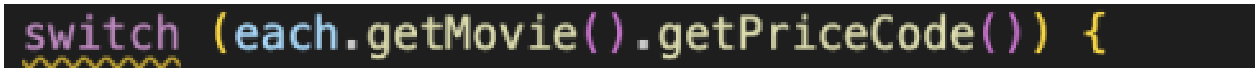
- This piece of code is a code smell because it contains <u>duplicated code</u> where there are two separate calculations for frequentRenterPoints.

9)


```
switch (each.getMovie().getPriceCode()) {
```

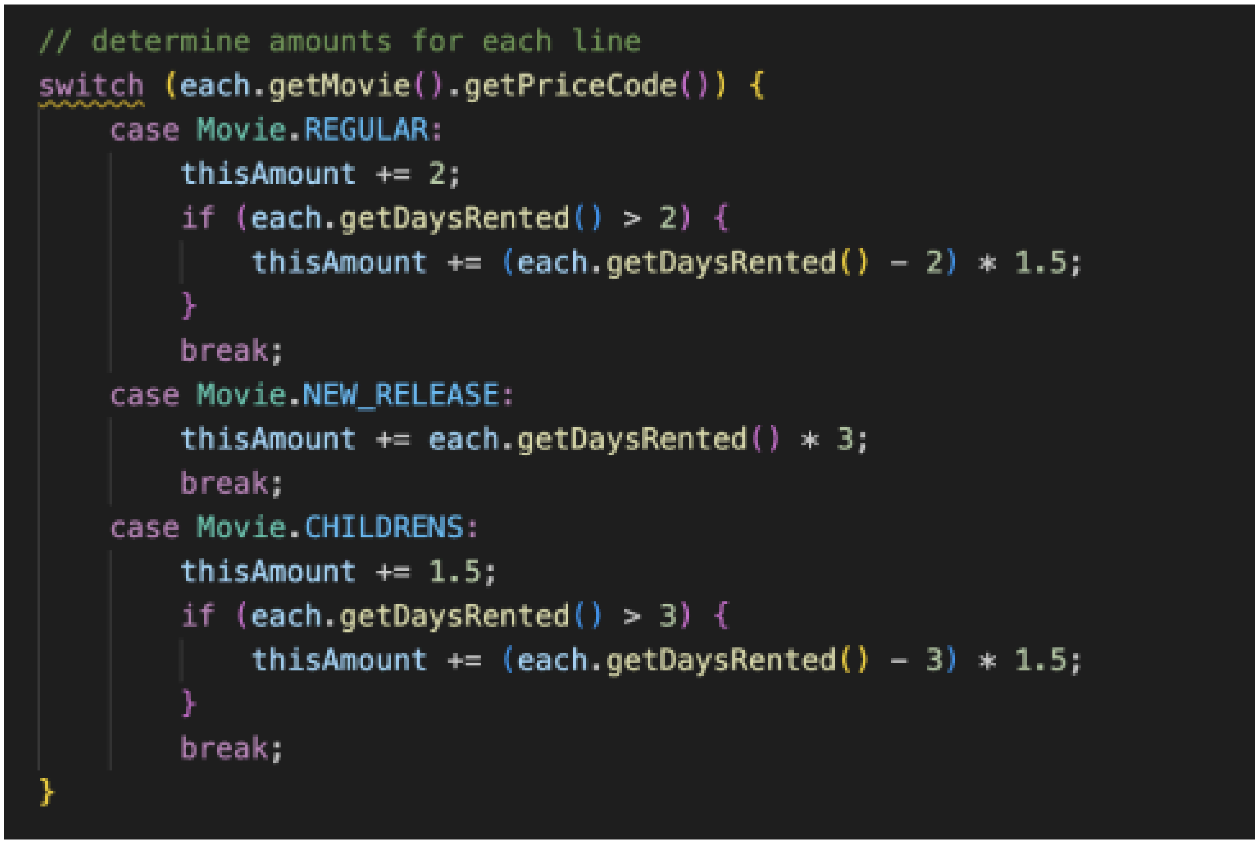
- This piece of code is a code smell because it contains a <u>message chain</u> where each object is calling for the getMovie() method which is calling for the getPriceCode() method.

10)


```
switch (each.getMovie().getPriceCode()) {
```

- This piece of code is a code smell because of <u>inappropriate intimacy</u>. The customer class knows too much about the Rental and Movie class's internal methods and variables.

11)

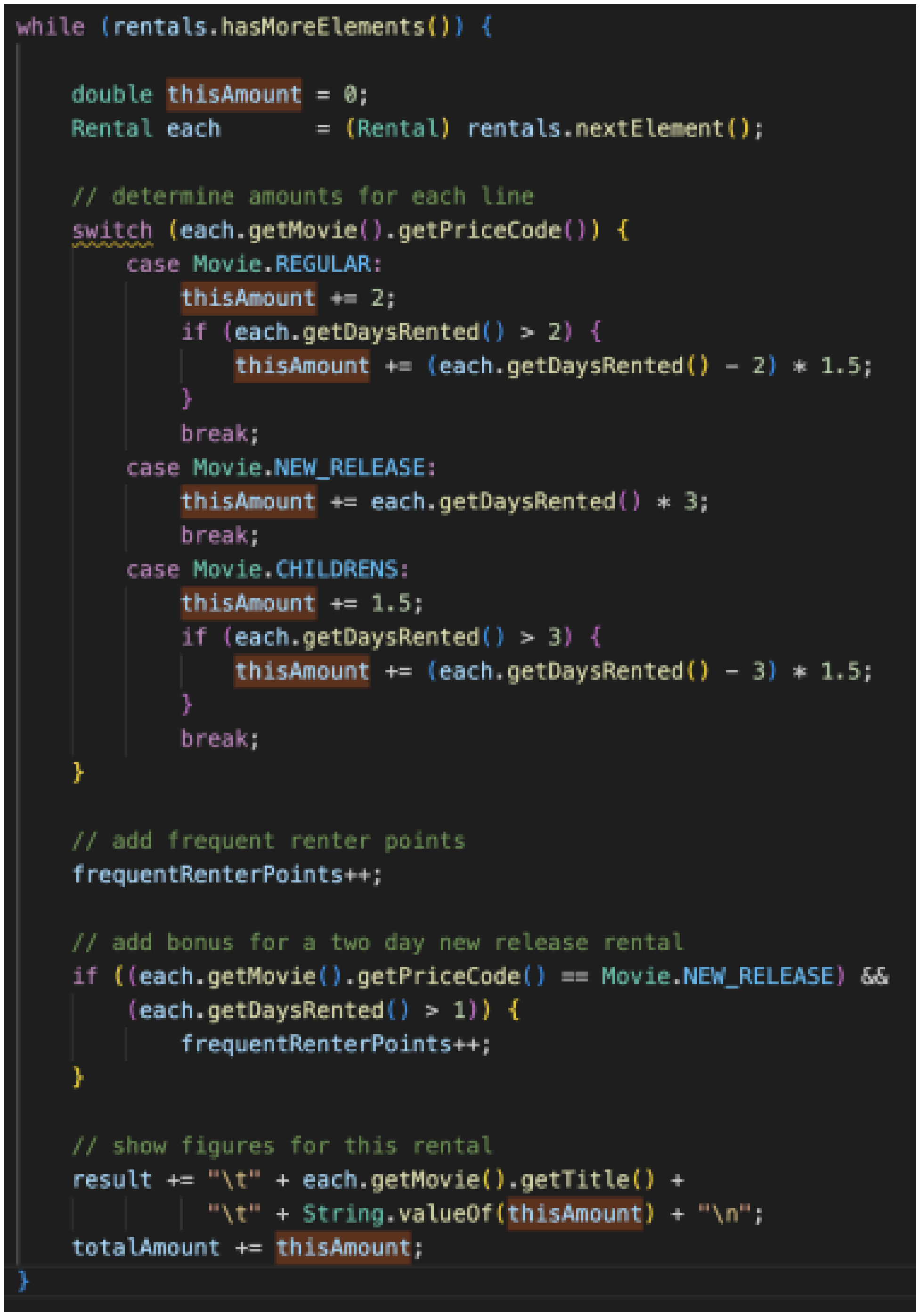


```java
// determine amounts for each line
switch (each.getMovie().getPriceCode()) {
    case Movie.REGULAR:
        thisAmount += 2;
        if (each.getDaysRented() > 2) {
            thisAmount += (each.getDaysRented() - 2) * 1.5;
        }
        break;
    case Movie.NEW_RELEASE:
        thisAmount += each.getDaysRented() * 3;
        break;
    case Movie.CHILDRENS:
        thisAmount += 1.5;
        if (each.getDaysRented() > 3) {
            thisAmount += (each.getDaysRented() - 3) * 1.5;
        }
        break;
}
```

- This piece of code is a code smell because it contains a lack of comments for logic.

12)

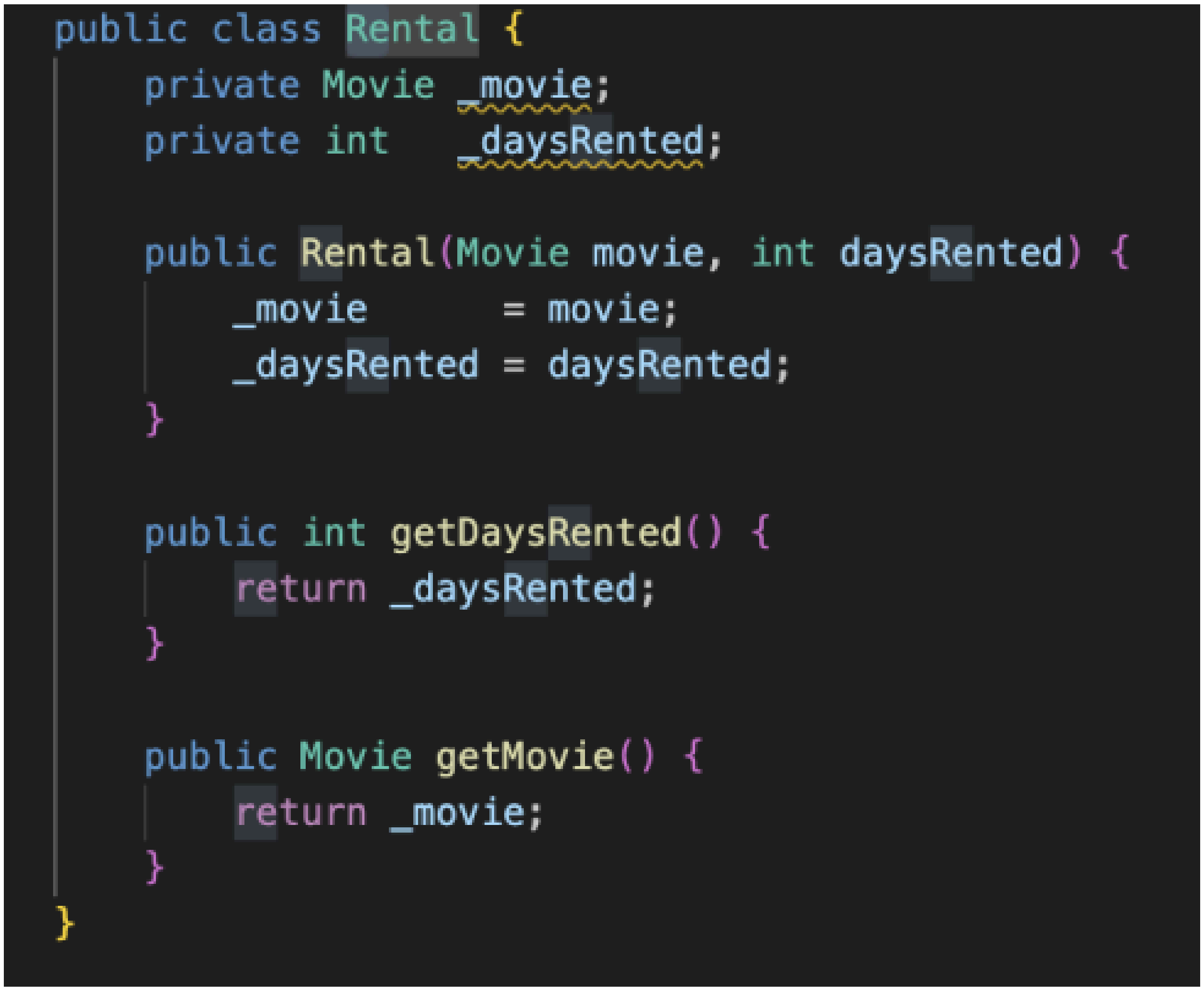
- Screenshot on next page



```java
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;
    }

    // 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;
}
```

- thisAmount is a code smell because it is only used within the while loop, then it is discarded. It is an example of a temporary field.

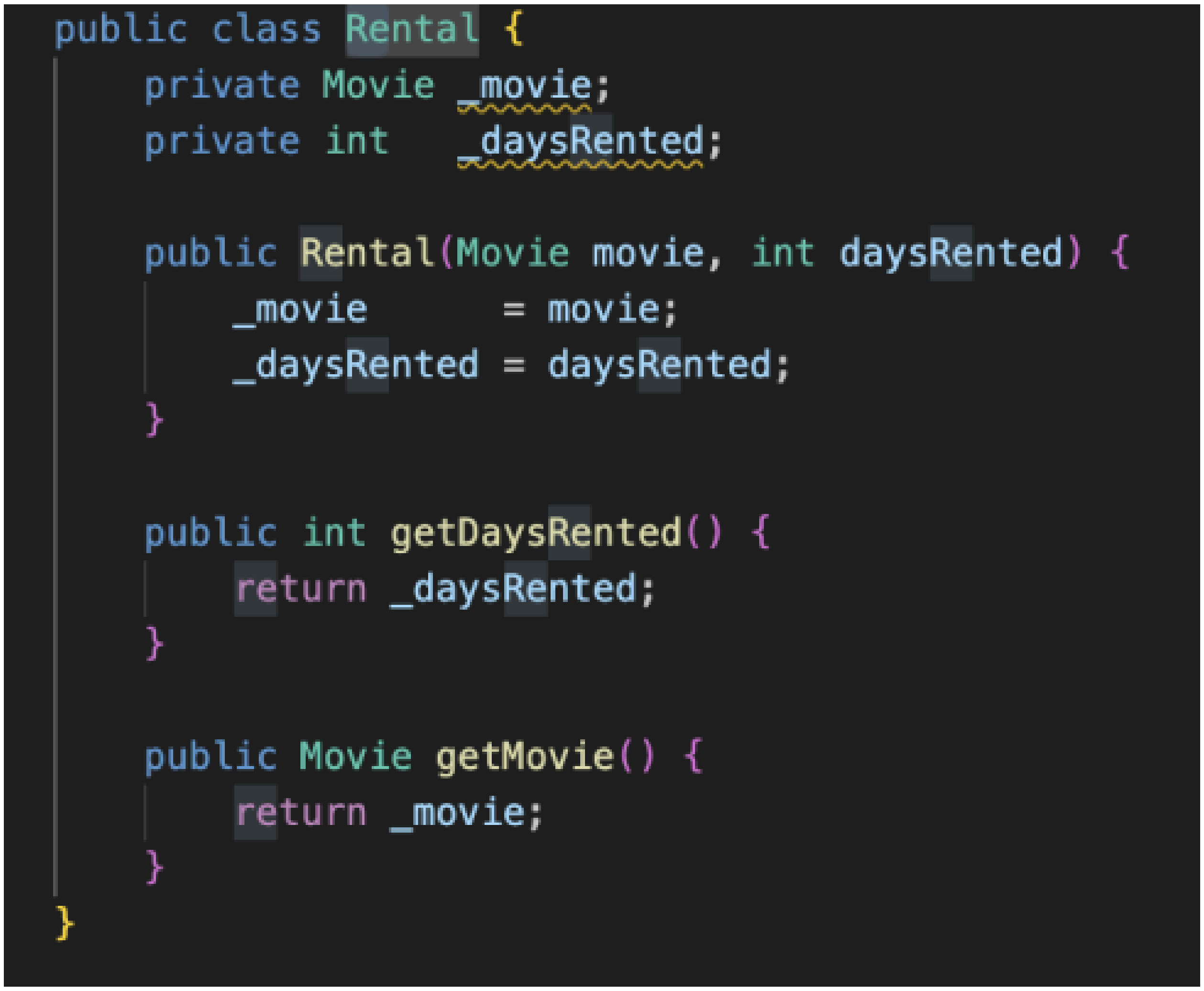
13)



```
public 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;
    }
}
```

- This piece of code is a code smell because the Rental class is a lazy class that does not do much except holding data.

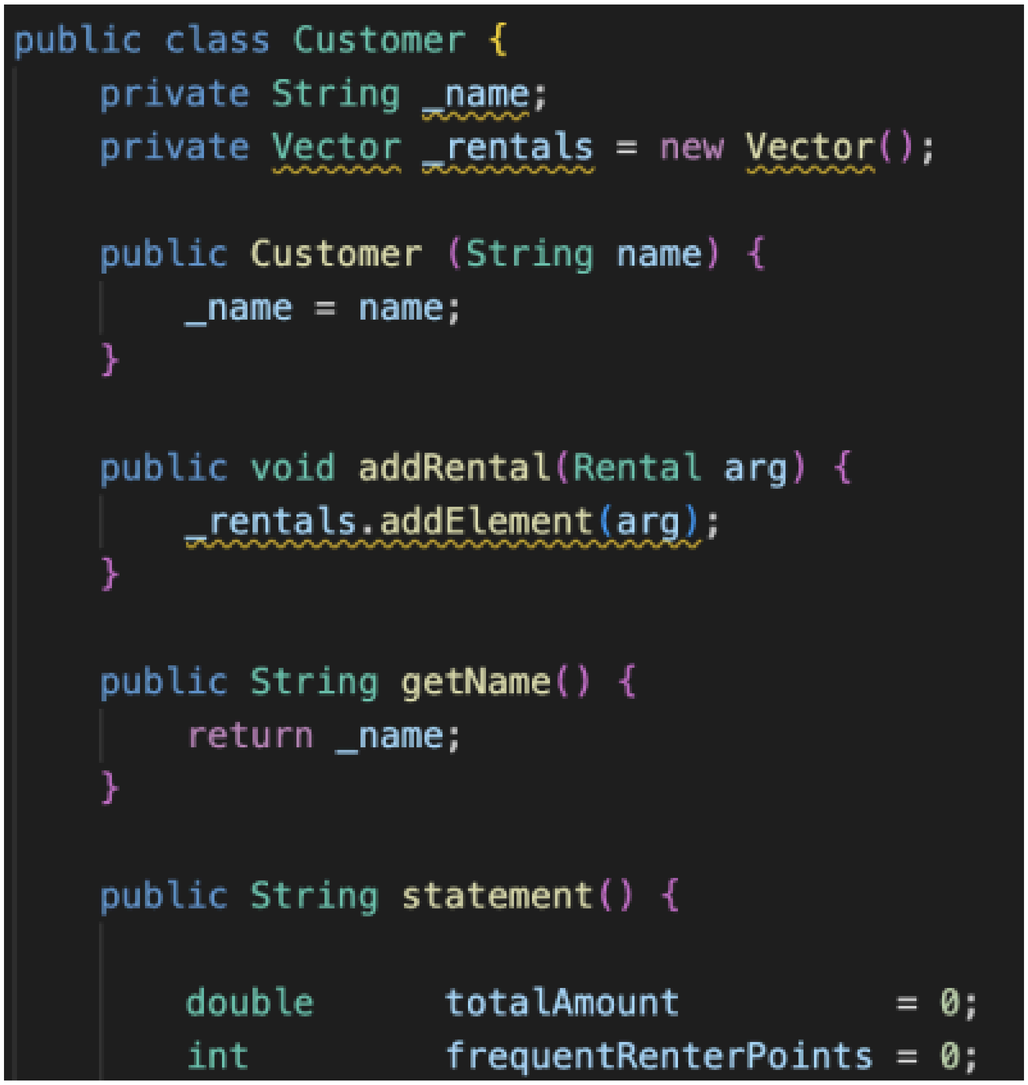
14)



```java
public 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;
    }
}
```

- Rental class is an example of a middle man because the class is dedicating half of the responsibilities to the Movie class.

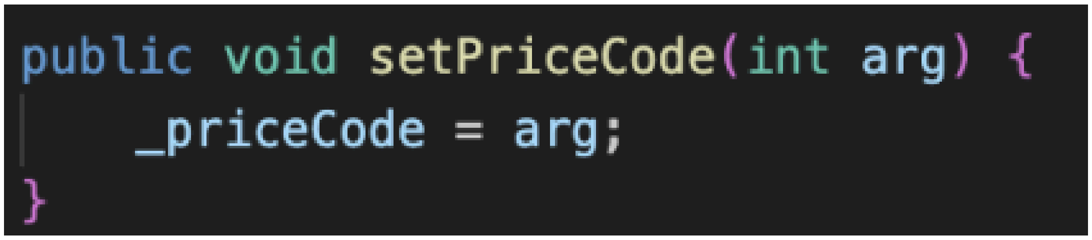
15)



```java
public 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;
    }

    public String statement() {

        double      totalAmount          = 0;
        int         frequentRenterPoints = 0;
```

- This piece of code is a code smell because it uses inconsistent naming conventions. Some variables have an _ before the name, and the variables inside the statement method do not.

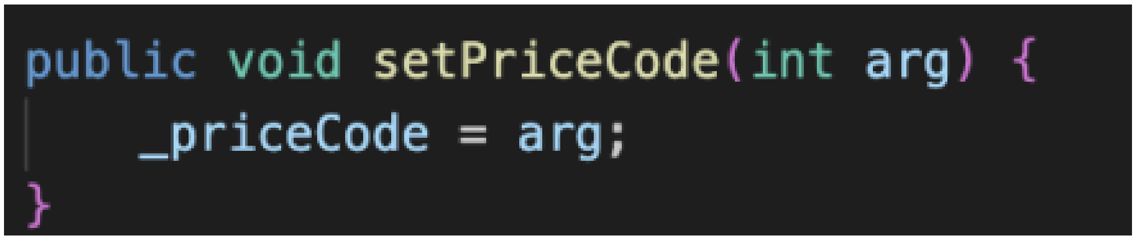
16)



```java
public void setPriceCode(int arg) {
    _priceCode = arg;
}
```

- This piece of code is a code smell because the setPriceCode() method written in the Rental class is never called in the Customer class.

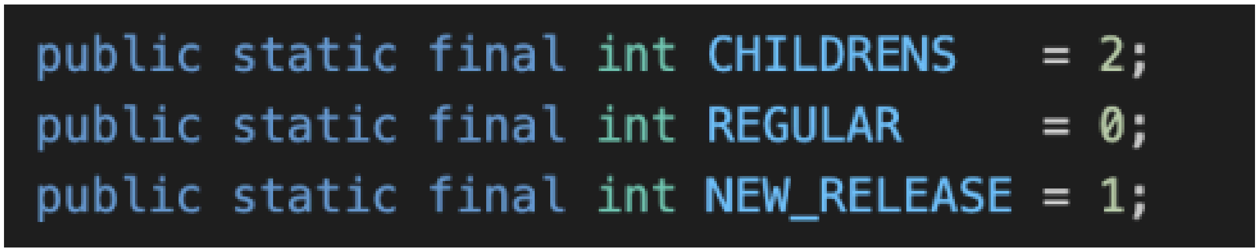
17)



```java
public void setPriceCode(int arg) {
    _priceCode = arg;
}
```

- Changing price logic affects the Customer and Rental class, making it a code smell. It is an example of a divergent change where one type of change requires changing other methods.

18)

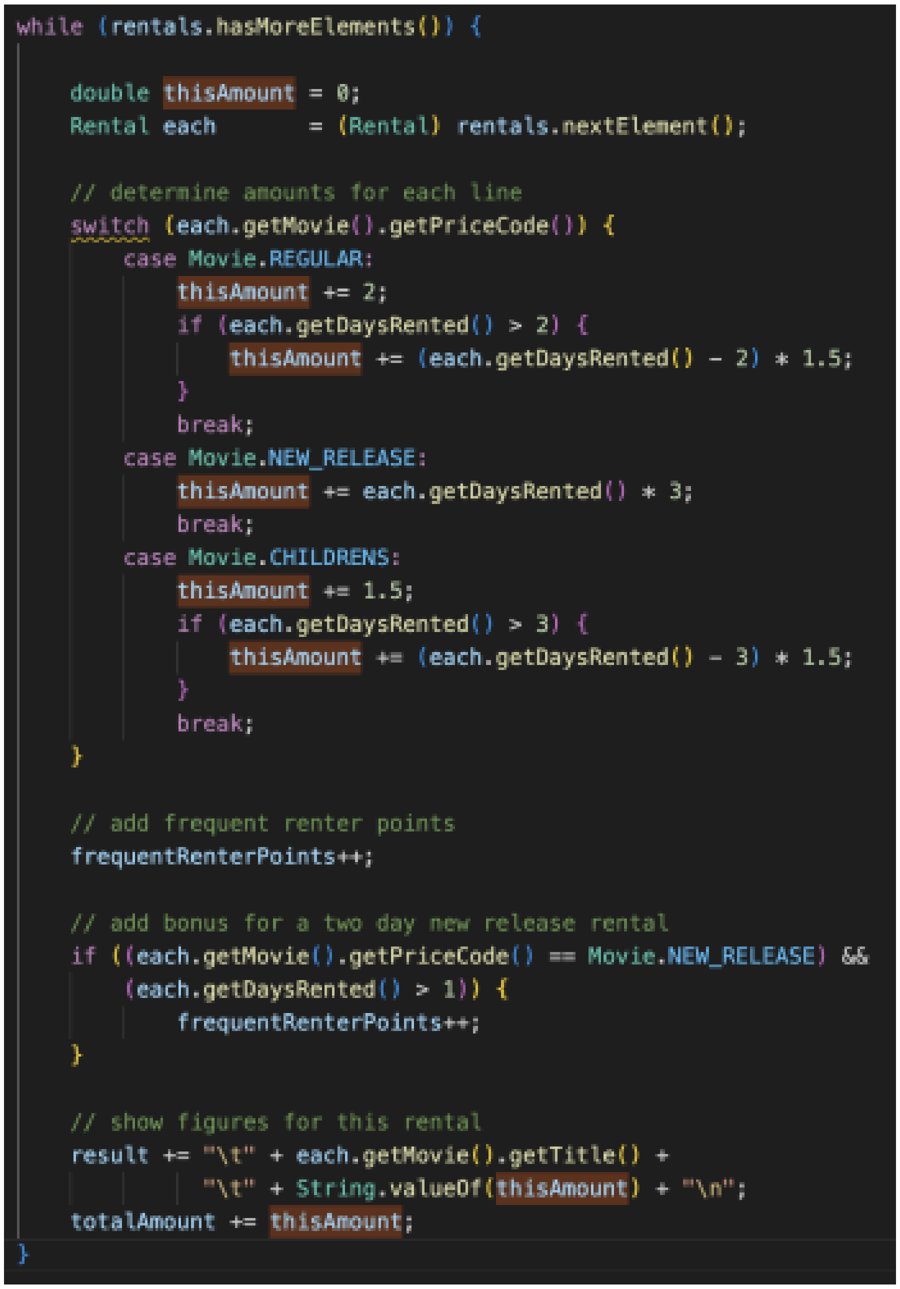


```java
public static final int CHILDRENS   = 2;
public static final int REGULAR     = 0;
public static final int NEW_RELEASE = 1;
```

- Movie class uses static fields which is a code smell. It is better to use enum values since it reduces errors.

19)

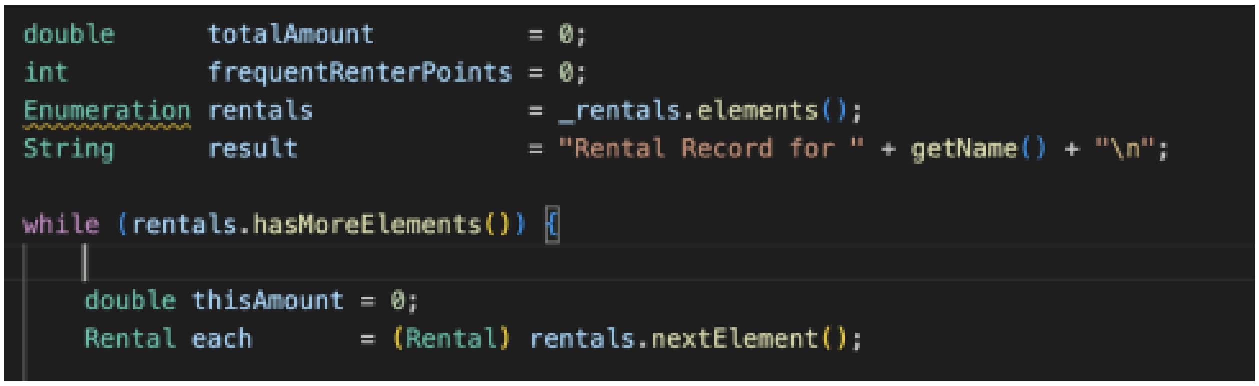
- Screenshot on next page



```java
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;
    }

    // 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;
}
```

- This piece of code contains a lot of magic hard-coded numbers such as 2, 1.5, and 3 which can be replaced with named constants for better clarity and understanding. This is an example of a code smell.

20)



```
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();
```

- This piece of code is a code smell because the totalAmount, frequentRenterPoints, and thisAmount variables are <u>primitive data types</u>, instead of being encapsulated by objects in a class.