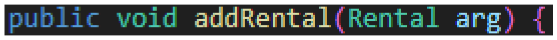
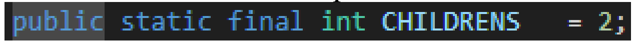
Homework 1 – Movie Rental Code Smells

1. 

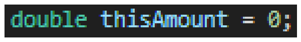
```
public void addRental(Rental arg) {
```

   a. Smell Type: Bad / Non-Descriptive Variable Name
   b. Explanation: Using the variable name ‘arg’ is an example of a non-descriptive variable name since it only indicates the variable holds a generic argument and provides no further information. This is a code smell since developers and users will have a hard time understanding what the purpose of ‘arg’ is, making it harder to use and modify the method and increasing the possibility of a misunderstanding of the variable’s meaning leading to code bugs. Therefore, this variable name reduces software qualities such as readability and understandability of the code.

2. 

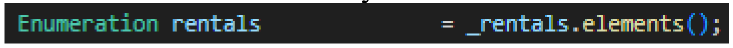
```
public static final int CHILDRENS   = 2;
```

   a. Smell Type: Bad / Non-Descriptive Constant Name
   b. Explanation: The constant name ‘CHILDRENS’ is an example of a non-descriptive constant name since although it provides some description, the description is still vague about the variable’s purpose and can be improved. This is a code smell since developers and users will have a hard time understanding what the exact meaning and purpose of ‘CHILDRENS’ is, making it harder to use and modify the class and increasing the possibility of a misunderstanding of the constant’s meaning leading to code bugs. Therefore, this constant name reduces software qualities such as readability and understandability of the code.

3. 

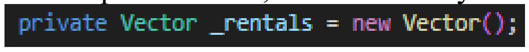
```
double thisAmount = 0;
```

   a. Smell Type: Bad / Non-Descriptive Variable Name
   b. Explanation: The variable name ‘thisAmount’ is an example of a non-descriptive variable name since it only indicates the variable holds an amount which is very vague as the amount could be a weight, cost, volume, etc. This is a code smell since developers and users will have a hard time understanding what the purpose of ‘thisAmount’ is, making it harder to use and modify the method and increasing the possibility of a misunderstanding of the variable’s meaning leading to code bugs. Therefore, this variable name reduces software qualities such as readability and understandability of the code.

4. 

```
Enumeration rentals           = _rentals.elements();
```

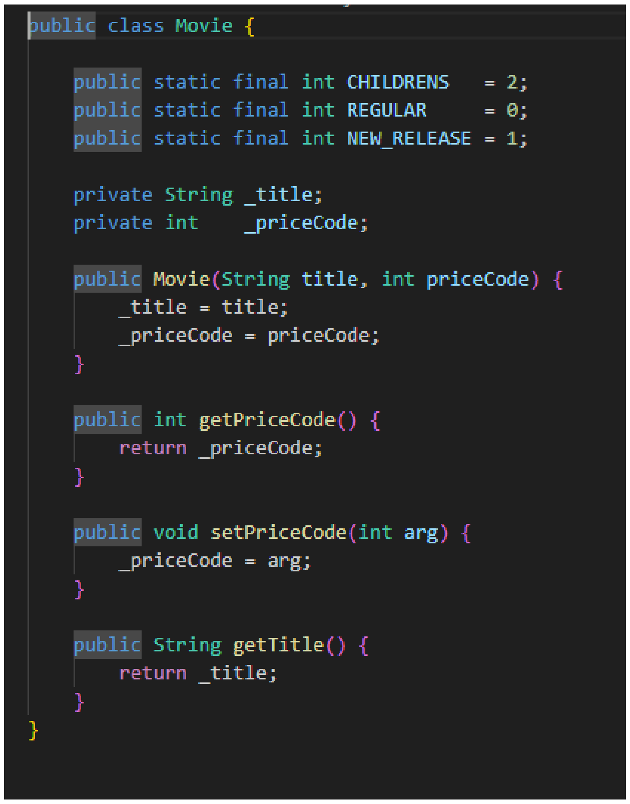
   a. Smell Type: Legacy Data Structure Usage
   b. Explanation: Enumeration is considered a legacy data structure in Java and has been replaced by use of Iterator or other iteration techniques. This is a code smell since legacy data structures will be less commonly used by modern developers and developers will be less familiar with them. Additionally, legacy data structures are often no longer maintained, perform generally worse than modern counterparts and may even be discontinued which could cause unexpected bugs for the code base or require the development team to rapidly change many sections of code in the case of a discontinuation. Therefore, the use of legacy data structures reduces software quality in terms of robustness, reliability, performance, maintainability and understandability.

5. 

```
private Vector _rentals = new Vector();
```

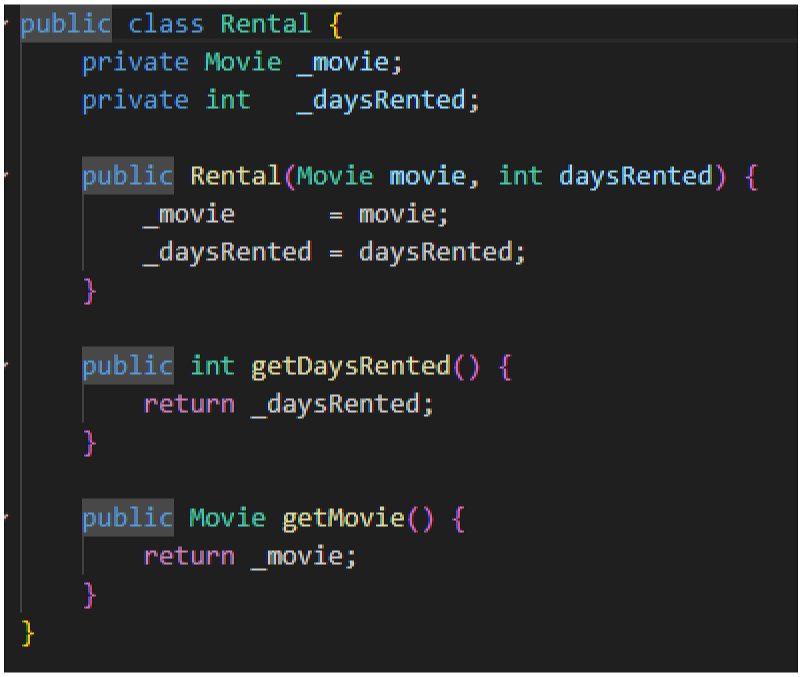
   a. Smell Type: Legacy Data Structure Usage

b. Explanation: Vector is considered a legacy data structure in Java and has been replaced by use of the List class. This is a code smell since legacy data structures will be less commonly used by modern developers and they will be less familiar with them. Additionally, legacy data structures are often no longer maintained, perform generally worse than modern counterparts and may even be discontinued which could cause sudden bugs for the code base or require the development team to rapidly change many sections of code in the case of a discontinuation. Therefore, the use of legacy data structures reduces software quality in terms of robustness, reliability, performance, maintainability and understandability.

6. 

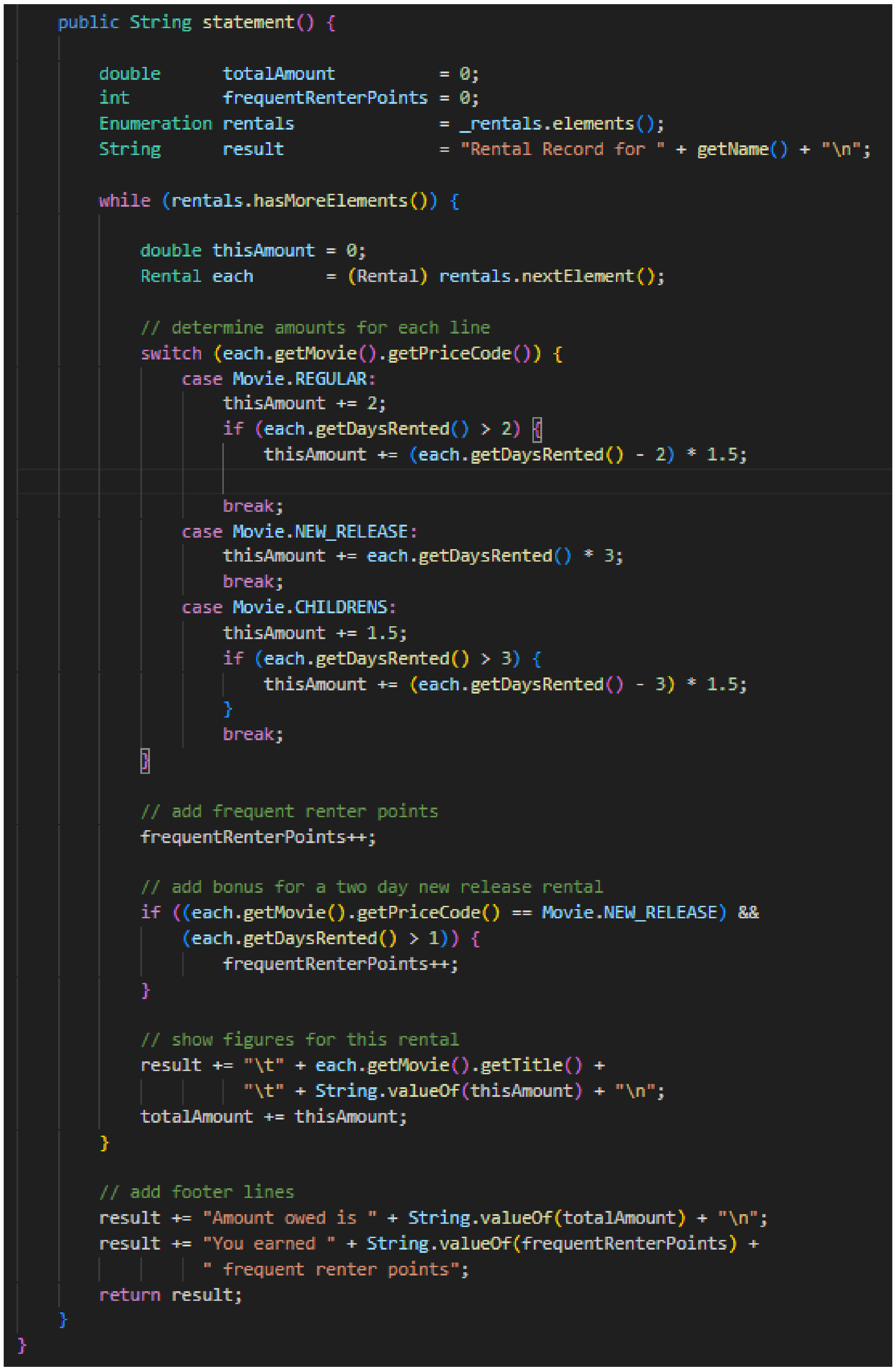
```java
public class Movie {

    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;
    }
}
```

a. Smell Type: Data Class
b. Explanation: The Movie class is a data class because it contains data along with only getter and setter methods. This is a code smell since it violates the encapsulation principle of OOP where classes should contain data and behavior methods that operate on them. This could naturally lead to many other classes having methods that operate on this data. However, if the data class is changed in a significant way, many of these dependent class methods may also need to be changed and it may be unclear to the developers where these dependent class methods are. Therefore, this data class reduces the software qualities of maintainability and encapsulation.

7. 

```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;
    }
}
```

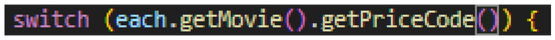
   a. Smell Type: Data Class
   b. Explanation: The Rental class is a data class because it contains data along with only getter and setter methods. This violates the encapsulation principle of OOP where classes should contain data and behavior methods that operate on them. This could naturally lead to many other classes having methods that operate on this data. However, if the data class is changed in a significant way, many of these dependent class methods may also need to be changed and it may be unclear to the developers where these dependent class methods are. Therefore, this data class reduces the software qualities of maintainability and encapsulation.



```java
    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;
            }

            // 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;
    }
}
```

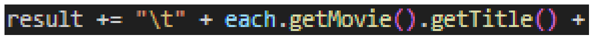
8.
   a. Smell Type: Long Method
   b. Explanation: The statement method is a ‘long method’ because it handles multiple responsibilities that could be separated into different methods such as statement calculations, updating bonus points, and displaying output figures. This is a code smell because in the future developers that wish to reuse sections of this code will be tempted to copy and paste it, which increases the risk of needing shotgun

surgery in the future along with other maintainability issues. Additionally, the code is messy and clearly difficult to read and understand, which impacts users and developers who are trying to utilize or debug the code. In terms of software qualities this reduces the code's maintainability and understandability while violating the principle of separation of concerns.

9. 

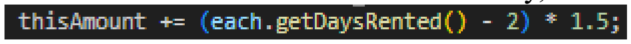
```
switch (each.getMovie().getPriceCode()) {
```

   a. Smell Type: Message Chain
   b. Explanation: This is an example of a message chain because the Movie object returned by getMovie() calls another method called getPriceCode() resulting in a two-method long message chain. This is a code smell because it makes the code less maintainable since a change to either the Rental class or the Movie class now may possibly affect the entire message chain. Additionally, each method must check for null / None objects correctly or else errors may occur, which is an additional potential issue for maintainability. Therefore, in terms of software qualities, these issues reduce the code's reliability, robustness, and maintainability.

10. 

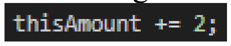
```
result += "\t" + each.getMovie().getTitle() +
```

   a. Smell Type: Message Chain
   b. Explanation: This is an example of a message chain because the Movie object returned by getMovie() calls another method called getTitle() resulting in a two-method long message chain. This is a code smell because it makes the code less maintainable since a change to either the Rental class or the Movie class now may possibly affect the entire message chain. Additionally, each method must check for null / None objects correctly or else errors may occur, which is an additional potential issue for maintainability. Therefore, in terms of software qualities, these issues reduce the code's reliability, robustness, and maintainability.

11. 

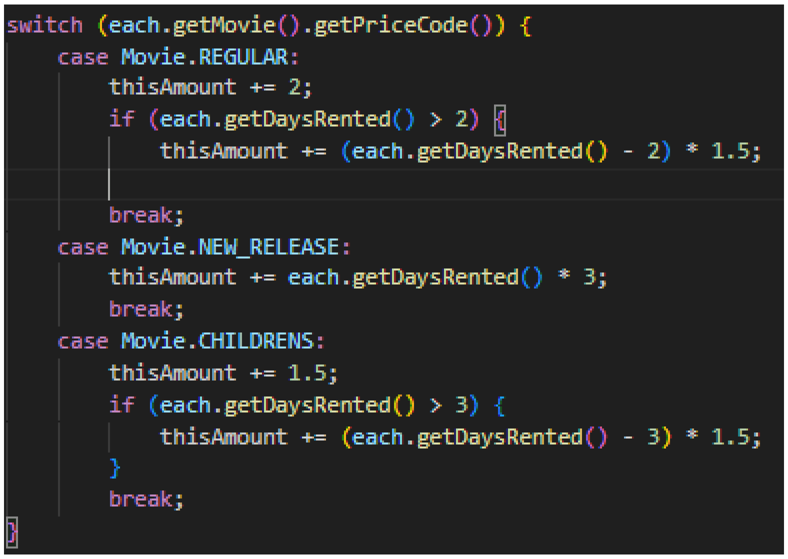
```
thisAmount += (each.getDaysRented() - 2) * 1.5;
```

   a. Smell Type: Magic Numbers / Hard Code
   b. Explanation: This is an example of a magic number since the numbers '2' and '1.5' are hard coded as numbers rather than variables. This is a code smell as it reduces the software qualities of understandability and maintainability because future users and developers will have to spend more time to understand the meaning of these numbers and are more likely to misuse the code or introduce bugs since it is harder to understand the meaning of the numbers.

12. 

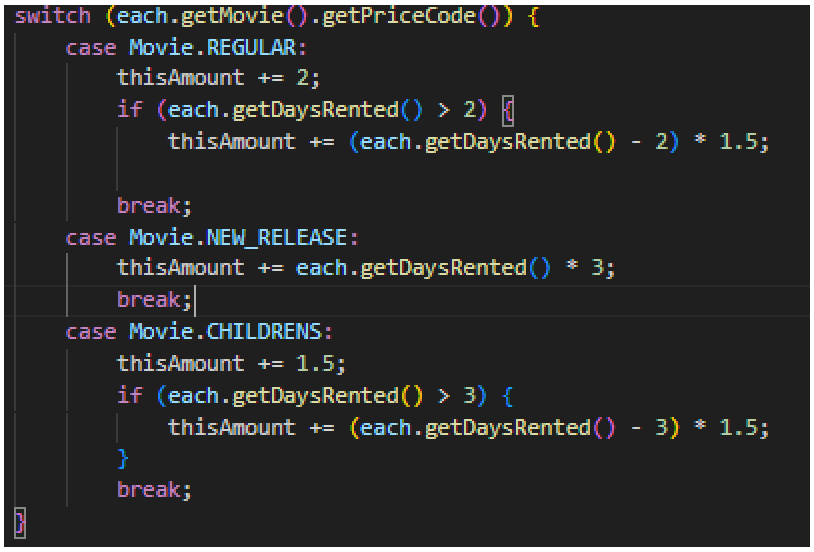
```
thisAmount += 2;
```

   a. Smell Type: Magic Numbers / Hard Code
   b. Explanation: This is an example of a magic number smell because the number '2' is hard coded as a number rather than a variable. This is a code smell since it reduces the software qualities of understandability and maintainability because future users and developers will have to spend more time to understand the meaning of these numbers and are more likely to misuse the code or introduce bugs since it is harder to understand the meaning of the numbers.

13. 

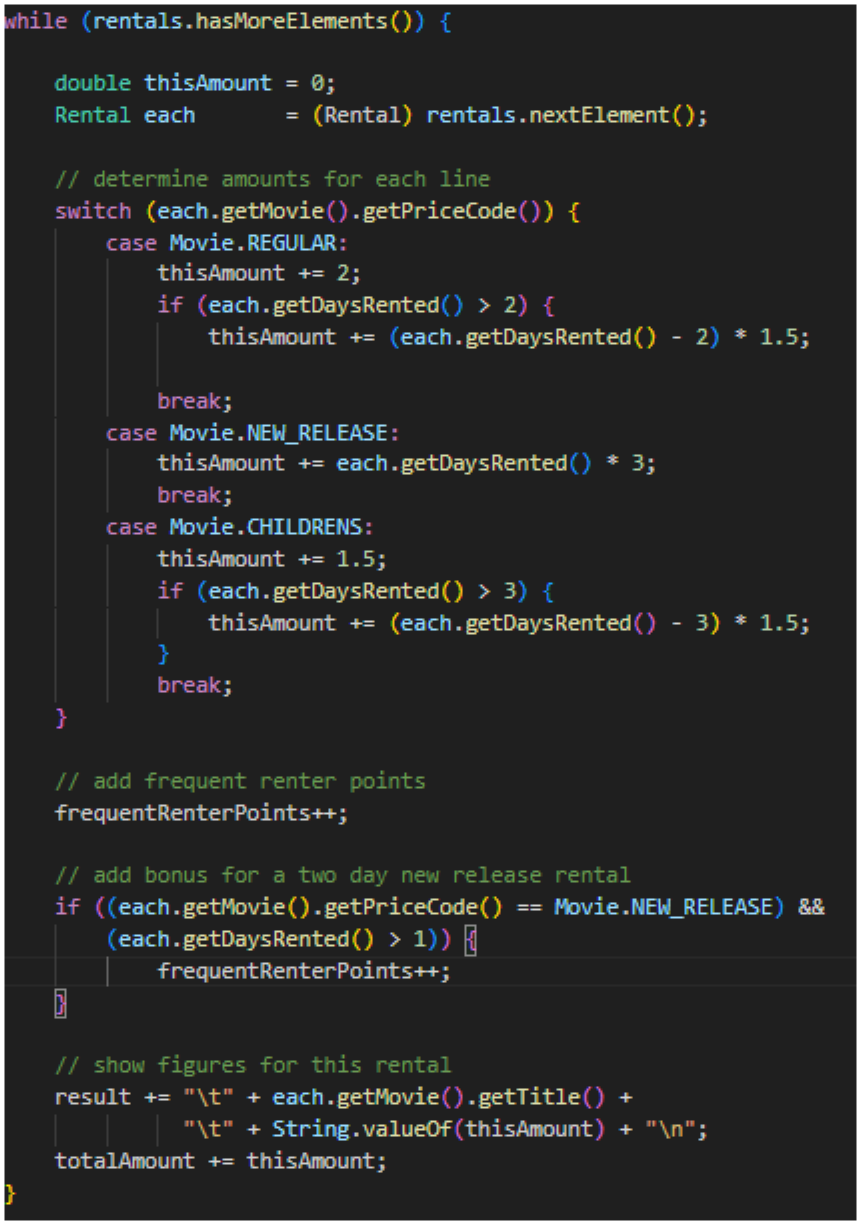
```
switch (each.getMovie().getPriceCode()) {
    case Movie.REGULAR:
        thisAmount += 2;
        if (each.getDaysRented() > 2) {
            thisAmount += (each.getDaysRented() - 2) * 1.5;

        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;
}
```

a. Smell Type: Switch Statement
b. Explanation: The above code shows a switch statement with a case for each Movie type. This is a code smell because if the number of Movie types is later changed, all switch statements like this will have to be modified accordingly. This reduces the software quality of maintainability since it may be difficult to locate all of these switch statements throughout the codebase and the developer may not even realize they need to update these switch statements when changing the types of the Movie class.

14. 

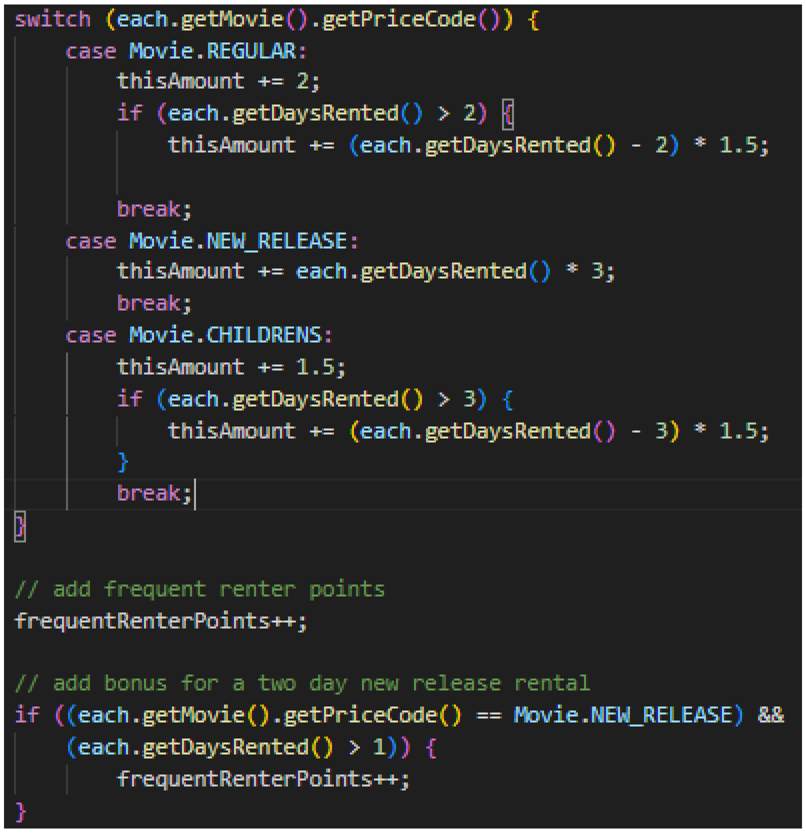
```
switch (each.getMovie().getPriceCode()) {
    case Movie.REGULAR:
        thisAmount += 2;
        if (each.getDaysRented() > 2) {
            thisAmount += (each.getDaysRented() - 2) * 1.5;

        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;
}
```

a. Smell Type: Switch Statement Lacking Default Case
b. Explanation: In the above code, the switch statement has a case for each Movie type but does not have a case to handle other data. This is a code smell because later new cases may be introduced, and the switch statement will not handle them. This can result in silent errors, incorrect behavior, and hard-to-trace bugs. Therefore, this code smell results in reduction in the software qualities of robustness, correctness, and reliability.



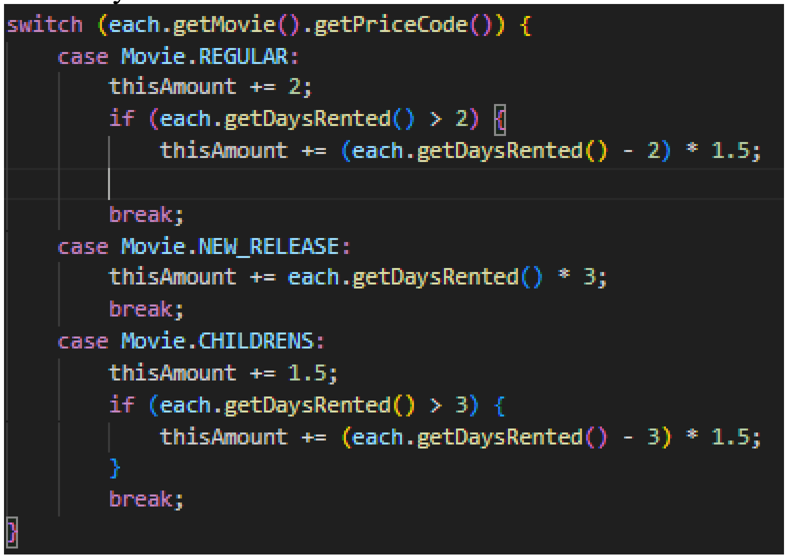
```
while (rentals.hasMoreElements()) {

    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;
}
```

15.
  a. Smell Type: Feature Envy
  b. Explanation: This is an example of feature envy since each.getDaysRented() is called multiple times indicating that the Customer classes statement method requires a great deal of information from the Rental class. This is a code smell because the greater distance between the data and the methods that operate on it, the more likely you will need to use message chains or other problematic code structures. Therefore, the code here reduces the software qualities of maintainability, reusability and encapsulation of the software system.

16. 

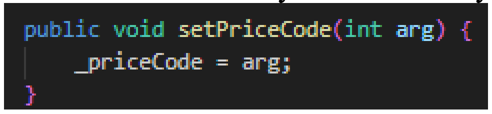
```
switch (each.getMovie().getPriceCode()) {
    case Movie.REGULAR:
        thisAmount += 2;
        if (each.getDaysRented() > 2) {
            thisAmount += (each.getDaysRented() - 2) * 1.5;
        
        break;
    case Movie.NEW_RELEASE:
        thisAmount += each.getDaysRented() * 3;
     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

a. Smell Type: Feature Envy
b. Explanation: This is another example of feature envy since each.getMovie().getPriceCode is called multiple times indicating that the Customer classes statement method requires a great deal of information from the Movie class. This is a code smell because the greater distance between the data and the methods that operate on it, the more likely you will need to use message chains or other problematic code structures. Therefore, the code here reduces the software qualities of maintainability, reusability and encapsulation of the software system.

17. 

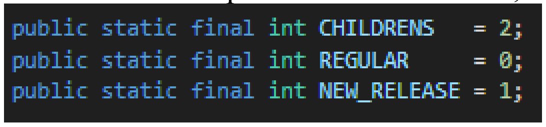
```
switch (each.getMovie().getPriceCode()) {
    case Movie.REGULAR:
        thisAmount += 2;
        if (each.getDaysRented() > 2) {
            thisAmount += (each.getDaysRented() - 2) * 1.5;
        
        break;
    case Movie.NEW_RELEASE:
        thisAmount += each.getDaysRented() * 3;
     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

a. Smell Type: Similar or Duplicated Code / Logic
b. Explanation: This is an example of similar/duplicated logic and code since each ‘thisAmount’ calculation uses a similar code structure. This is a code smell because if a developer wishes to change the way ‘thisAmount’ is calculated in a fundamental way then they would need to change each individual ‘thisAmount’ calculation. Therefore, having this duplicated logic reduces the software quality of maintainability and reusability.

18. 

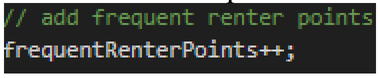
```
public void setPriceCode(int arg) {
    _priceCode = arg;
}
```

a. Smell Type: Lack of Argument Validation
b. Explanation: This is an example of lack of argument validation because arg is not checked to be a valid price code. This is a code smell because the price code could easily be set to an invalid int which could lead to an invalid switch statement case or other bug in the future. Therefore, this lack of argument validation reduces the software qualities of correctness, robustness, and reliability.

19. 

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

a. Smell Type: Primitive Obsession
b. Explanation: This is an example of primitive obsession because these constants are defined as ints when they could be instead defined as an enum type. This is a code smell because these primitive variable names are less descriptive than the enum analogue such as ‘priceCode.CHILDREN’ and they also lack type checking which would prevent a possible future line of incorrect code such as ‘priceCode = 3’ from being executed. Therefore, this case of primitive obsession reduces the software qualities of correctness, robustness, reliability, and understandability.

20. 

```
// add frequent renter points
frequentRenterPoints++;
```

a. Smell Type: Missing State Persistence / Missing Object State
b. Explanation: This is an example of missing state persistence because the frequent renter points should be stored persistently in the Customer class, but they are not. This is a code smell because in the future if a developer wishes to add features that utilize the frequent renter points, they will not have the ability to do so since the values are not stored persistently yet. They will also likely spend time being confused about how the frequent renter points are designed. Therefore, this case of missing state persistence reduces the software qualities of maintainability and understandability.