DAT 159 Oblig Assignment: Refactoring

Marc Bentele, George Korosty, Laura Mann

September 16th, 2018

Question 1 a)

Extract Method: getAmount()

We first extracted a method called getAmount, which is taken mainly from the switch statement inside the while loop. We created a method to return a double (the amount), and copied the switch statement inside the while loop. Since the **each** variable is referenced inside this switch statement and again outside of it, we passed a reference to each to the new method. The other variable that was being referenced from the original method was **rentals**, so we also created an argument for Enumeration rentals in our new method, and passed the variable to it. Instead of declaring thisAmount in the original method as being 0, we moved it down below the **each** declaration, and made its value the return value from our new method. We also changed the name of thisAmount in our new method to amount, so the two methods wouldn't have the same variable name.

We also condensed the if statements in the switch statement to one line, by removing the line adding a value to amount, and including that in the inline statement.

```
private double getAmount(Rental each, Enumeration rentals) {
    double amount = 0;
    int daysRented = each.getDaysRented();

    // determine amount for each line
    switch (each.getMovie().getPriceCode()) {
        case Movie.REGULAR:
            amount += (daysRented > 2) ? 2 + (daysRented - 2) * 1.5 : 2;
            break;
        case Movie.NEW_RELEASE:
            amount += daysRented * 3;
            break;
        case Movie.CHILDRENS:
            amount += (daysRented > 3) ? 1.5 + (daysRented - 3) * 1.5 : 1.5;
            break;
    }
    return amount;
}
```

Extract Variable: priceCode

To make the if statement in the method simpler, We created a variable called priceCode to replace the line each.getMovie().getPriceCode(), then replaced the expression with the variable. This allowed us to put the whole if statement in one line.

```
int priceCode = each.getMovie().getPriceCode();

addFrequentRenterPoints();

// add bonus for a two day new release rental
if (priceCode == Movie.NEW_RELEASE && each.getDaysRented() > 1) addFrequentRenterPoints();
```

Extract Method: addFrequentRenterPoints()

Since the frequent renter points are added to twice in the main method, we created a new method called addFrequentRenterPoints, which just adds 1 to the frequentRenterPoints variable. The variable was declared locally in the method, so we moved it to the class, and made it a private variable. Then we replaced all the expressions adding 1 to frequentRenterPoints with a call to the method addFrequentRenterPoints().

```
private void addFrequentRenterPoints() {
    frequentRenterPoints ++ ;
}
```

Extract Method: getRentalFigures() and getResult()

The main method, a string is declared called result, and different text and variables are added to this throughout the method. To make things simpler, and so that we could add all text to the string from external methods, we declare the string result as empty.

To make the method cleaner, we extracted two methods, getRentalFigures() and getResult(). The getRentalFigures method is called from inside the while loop, and references variables that are local to that loop. The new method takes these variables as parameters, and adds them to the result string, returning the new string back to the existing method.

```
private String getRentalFigures(String title, double amount) {
    return "\t" + title + "\t" + String.valueOf(amount) + "\n";
}
```

The second method we created, getResult, adds 3 strings together, making the final result. All of the method calls that were made previously in the result string statements become parameters in the method, and are passed in once, at the end of the existing method. This method also takes a parameter “result”, which is the variable we returned in getRentalFigures().

```
private String getResult(String name, String result, double total, int points) {
    return "Rental Record for " + name + "\n" + result + "Amount owed is " + String.valueOf(total) + "\n"
+ "You earned " + String.valueOf(points) + " frequent renter points";
}
```

Finally, remove all alterations to the string result, and make two calls: we set result to the result of getRentalFigures() in the while loop, and we return the result of the getResult method.

```
return getResult(getName(), result, totalAmount, frequentRenterPoints);
```

Extract Methods: regular_amount, new_release_amount, childrens_amount

To simplify the switch statement in getAmount, we extracted the expressions from each case into their own methods. Each method returns the amount specific to the type of movie.

```
public double regular_amount(int daysRented) {
    return (daysRented > 2) ? 2 + (daysRented - 2) * 1.5 : 2;
}

public double new_release_amount(int daysRented) {
    return daysRented * 3;
}

public double childrens_amount(int daysRented) {
    return (daysRented > 3) ? 1.5 + (daysRented - 3) * 1.5 : 1.5;
}
```

Switch statement calling the new methods:

```
private double getAmount(Rental each, Enumeration rentals) {
    double amount = 0;
    int daysRented = each.getDaysRented();

    // determine amount for each line
    switch (each.getMovie().getPriceCode()) {
        case Movie.REGULAR:
            amount += (daysRented > 2) ? 2 + (daysRented - 2) * 1.5 : 2;
            break;
        case Movie.NEW_RELEASE:
            amount += daysRented * 3;
            break;
        case Movie.CHILDRENS:
            amount += (daysRented > 3) ? 1.5 + (daysRented - 3) * 1.5 : 1.5;
            break;
    }
    return amount;
}
```

Replace Type Code with Polymorphism

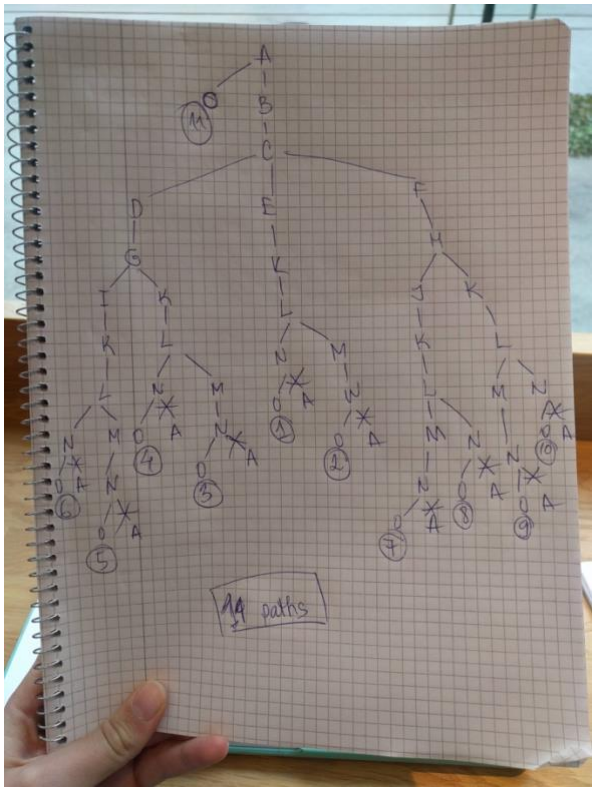
To eliminate the switch statement altogether, we created subclasses of Movie for each of the cases: Regular, New_Release, and Childrens. We made the Movie class abstract, and created a method called amount(), which takes the daysRented and returns the amount of the movie. We then moved the methods we just extracted in getAmount() to these new classes, and renamed them all to be the same: amount(), to override the method in Movie. Now, we can remove the switch statement completely, and just return the amount by calling Movie.amout(), which will look at which type of movie it is, and calculate and return the appropriate amount.

```
private double getAmount(Rental each) {
    return each.getMovie().amount(each.getDaysRented());
}
```

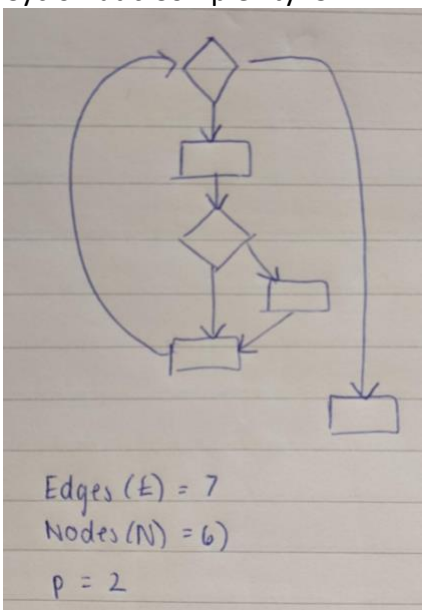
Move Method
We performed move method refactoring in the process of replacing the switch statement with polymorphism, as we moved the extracted methods we had created for returning the correct amount for each movie into the appropriate class: Regular, New_Release, and Childrens.

Question 1 b)

Before Refactoring
SLOC on statement method before refactoring: 34
Cyclomatic Complexity: 11

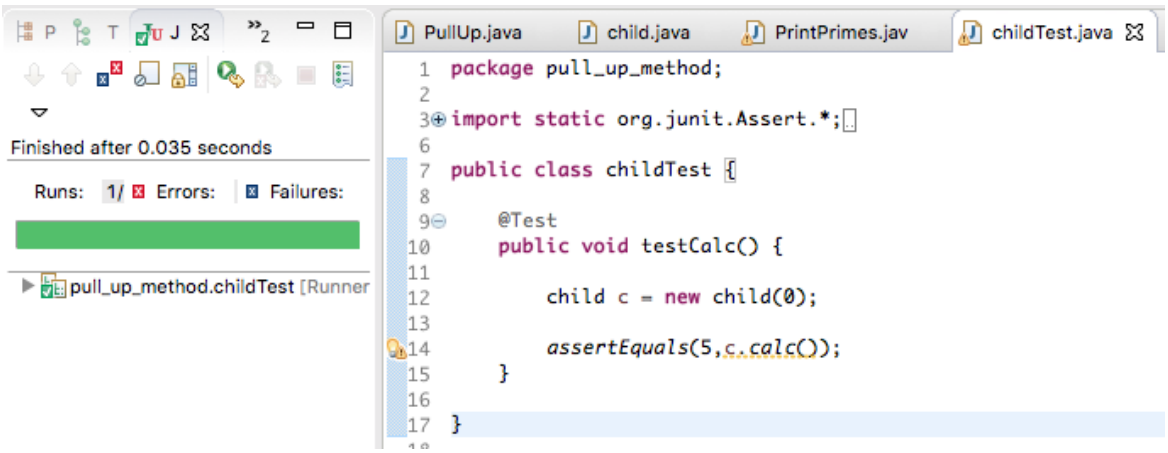


After Refactoring
SLOC: 13
Cyclomatic Complexity: 3



Question 2: Pull Up Method

To demonstrate how the pull up method can alter the results of the code, we created a simple project that just adds 2 numbers together and returns the result. One of these numbers is defined in the class, and one is defined in the method, **calc().** We created a superclass called PullUp, and a subclass called child, where the calc() method is located. This example defines a number (3) in the calc class and then adds 2 to it, which should return 5. We created a unit test, which passed fine in the subclass.



Next, we pulled up the method, calc, to the main class. To show how this could alter the code, we defined another number in the main class, so that the method would still run, but it would be using a different number.

```
public class PullUp {

    protected static int num =10;

    public static void main(String[] args) {
        int number = child.calc();
        System.out.println(number);
    }

    public static int calc() {
        return num+2;
    }
```

When this is run, we get 12 instead of 5:

But when we perform a unit test on the method in the main class, it fails because it's not giving the right result.