Oblig 1
IKT

We can see that each.getDaysRented is called 6 times.
We can make a variable x = each.getDaysRented and use this instead, to avoid calling the method that many times.

```
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();
        int daysRented = each.getDaysRented();
        MovieType movieType = each.getMovie().getMovieType();

        // determine amount for each line
        thisAmount = movieType.getAmountOwed(daysRented);
        // add frequent renter points
        frequentRenterPoints ++;
        // add bonus for a two day new release rental
        if ((movieType.getClass() == MovieType.NEW_RELEASE.class) &&
                daysRented > 1) frequentRenterPoints ++;

        //show figures for this rental
        result += "\t" + each.getMovie().getTitle()+ "\t" + thisAmount + "\n";
        totalAmount += thisAmount;
    }
    //add footer lines
    result += "Amount owed is " + totalAmount + "\n";
    result += "You earned " + frequentRenterPoints +
            " frequent renter points";
    return result;
}
```

```
abstract class MovieType {

    abstract double getAmountOwed(int days);

    class REGULAR extends MovieType {

        double getAmountOwed(int days) {
            double amount = 0.0;
            amount += 2;
            if (days > 2) {
                amount += (days - 2) * 1.5;
            }
            return amount;
        }
    }

    class NEW_RELEASE extends MovieType {

        double getAmountOwed(int days) {
            double amount = 0.0;
            amount += days * 3;
            return amount;
        }
    }

    class CHILDRENS extends MovieType {

        double getAmountOwed(int days) {
            double amount = 0.0;
            amount += 2;
            if (days > 3) {
                amount += (days - 3) * 1.5;
            }
            return amount;
        }
    }

}
```

We see that price codes are defined at final fields in Movie.java and used in the switch in Customer.java. We can move these to an abstract class, called MovieType.java, which will return the correct amount owed based on the (abstract) class. (Replace Conditional With Polymorphism)

We see now that the variable ‘pricecode’ is redundant, since we now use MovieType to differenciate them. So in Movie’s constructor, we now have an abstract ‘MovieType’ which can be polymorphed into Regular, New_release or childrens.

This means that we can define different getPrice() methods in

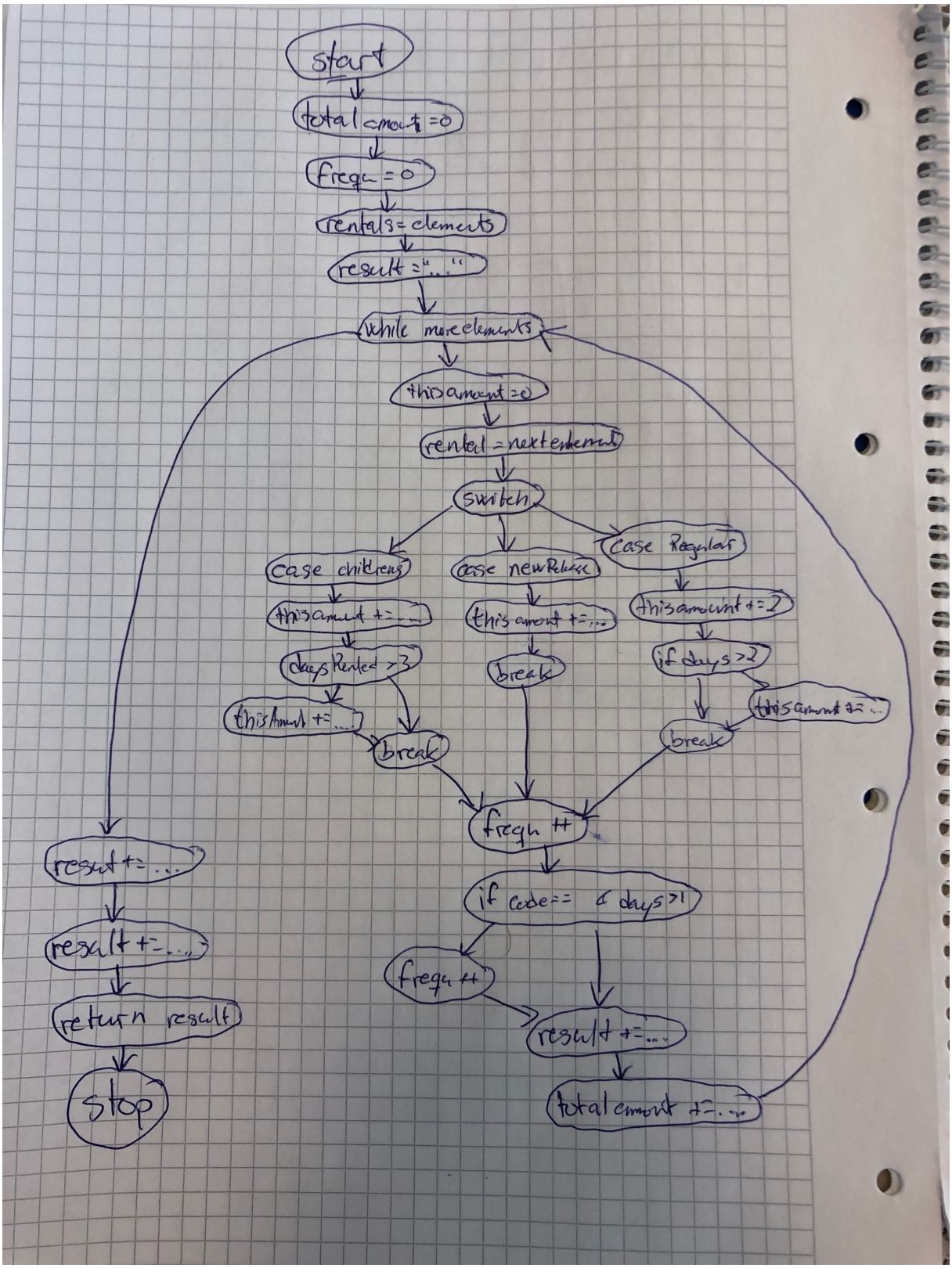
each of the abstract classes, and can move the method away from the switch-case from Customer.java (to MovieType).

SLOC & McCabe's Cyclomatic Complexity

SLOC before: 35 lines of code
SLOC after: 22 lines of code

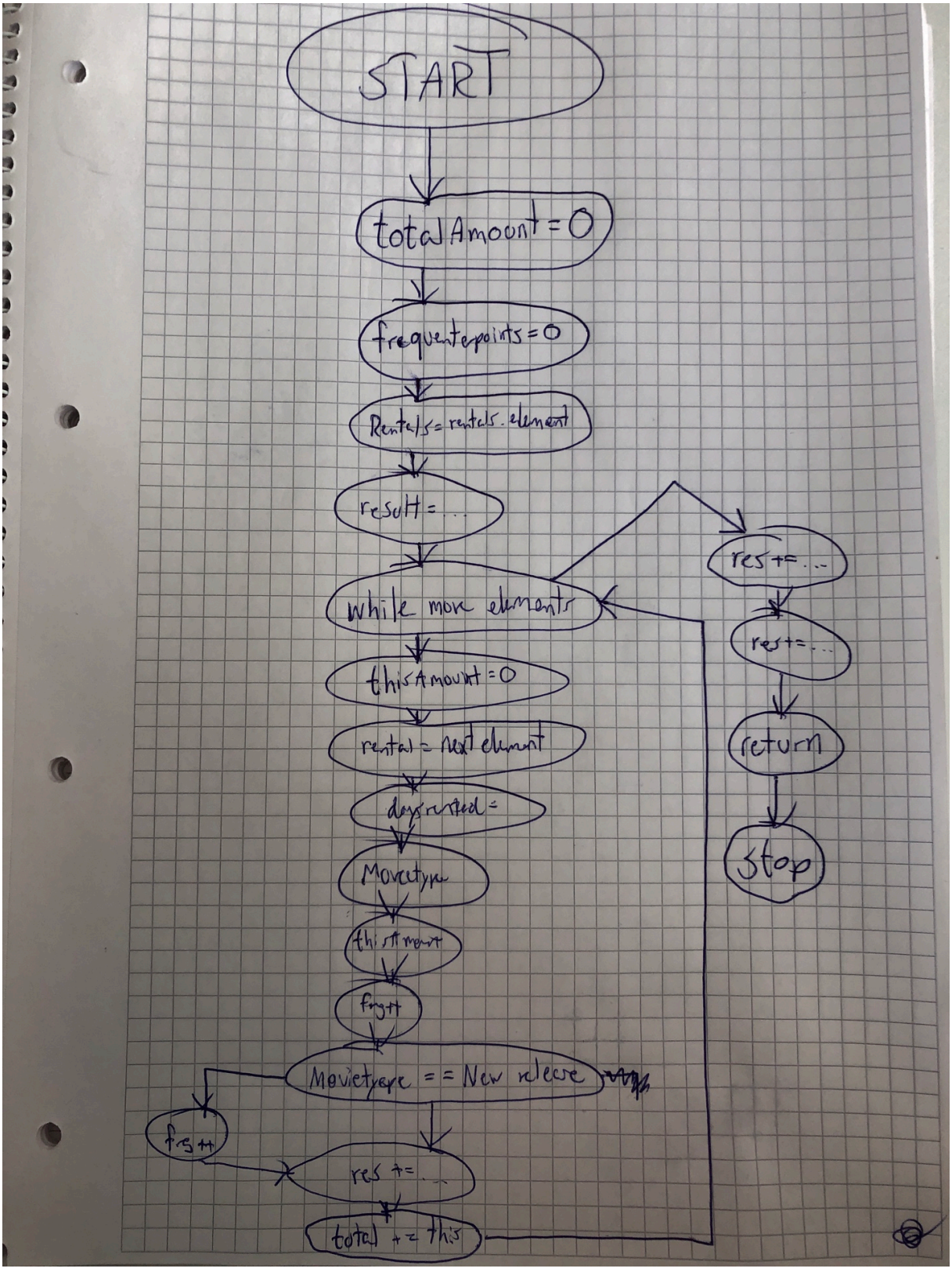
M = E – N + 2P

Cyclomatic complexity before:


start
totalAmount = 0
frequ = 0
rentals = elements
result = "..."
while moreElements
thisAmount = 0
rental = nextElement
switch
case Regular
case childrens
case newRelease
thisAmount += 2
if days > 2
thisAmount += ...
break
thisAmount += ...
daysRented > 3
thisAmount += ...
break
break
frequ ++
if code == 1 days > 1
frequ ++
result += ...
totalAmount += ...
result += ...
result += ...
return result
stop

M = 36 - 31 + 2*1 = 7

Cyclomatic complexity after:



We have 21 edges and 20 nodes and 1 connected component.

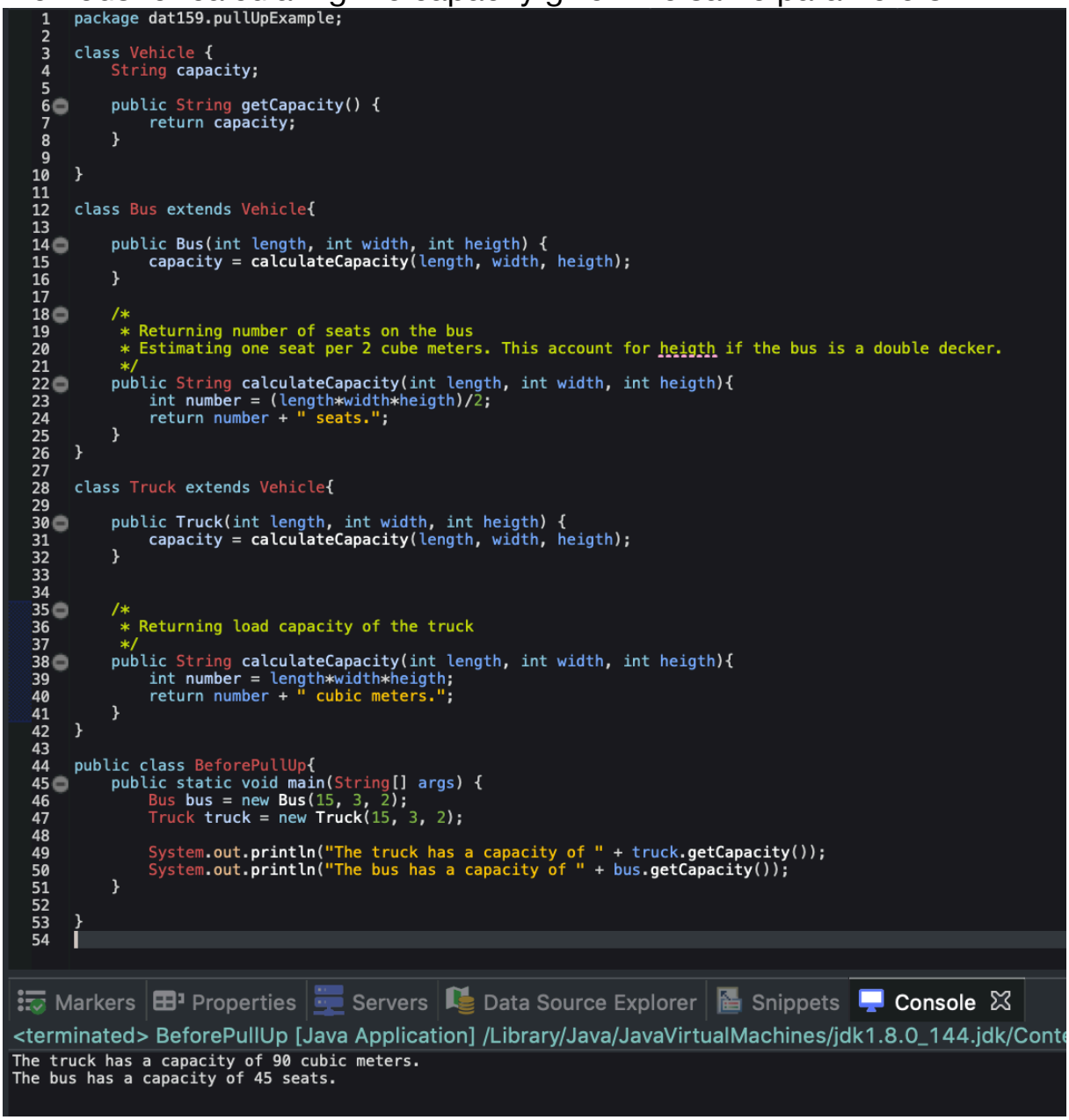
That gives 21-20+2 = 3.

The refactoring that impacted the SLOC the most was moving the switch-statement to MovieType.java as a method (move method). This made the statement method do a lot fewer comparisons.

Small, self-contained Java example

We decided to look at the capacity of two different vehicles bus and truck. They are both dependent on the size of the vehicle, can both be presented as integers or strings - but they represent two completely different sets of information. A bus' capacity refers to its seat-capacity - whereas a truck's capacity refers to its measurements.

The following example shows the two classes having their separate methods for calculating the capacity given the same parameters:



```
1  package dat159.pullUpExample;
2
3  class Vehicle {
4      String capacity;
5
6      public String getCapacity() {
7          return capacity;
8      }
9
10 }
11
12 class Bus extends Vehicle{
13
14     public Bus(int length, int width, int heigth) {
15         capacity = calculateCapacity(length, width, heigth);
16     }
17
18     /*
19      * Returning number of seats on the bus
20      * Estimating one seat per 2 cube meters. This account for heigth if the bus is a double decker.
21      */
22     public String calculateCapacity(int length, int width, int heigth){
23         int number = (length*width*heigth)/2;
24         return number + " seats.";
25     }
26 }
27
28 class Truck extends Vehicle{
29
30     public Truck(int length, int width, int heigth) {
31         capacity = calculateCapacity(length, width, heigth);
32     }
33
34
35     /*
36      * Returning load capacity of the truck
37      */
38     public String calculateCapacity(int length, int width, int heigth){
39         int number = length*width*heigth;
40         return number + " cubic meters.";
41     }
42 }
43
44 public class BeforePullUp{
45     public static void main(String[] args) {
46         Bus bus = new Bus(15, 3, 2);
47         Truck truck = new Truck(15, 3, 2);
48
49         System.out.println("The truck has a capacity of " + truck.getCapacity());
50         System.out.println("The bus has a capacity of " + bus.getCapacity());
51     }
52
53 }
54
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

```
<terminated> BeforePullUp [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Conte
The truck has a capacity of 90 cubic meters.
The bus has a capacity of 45 seats.
```

Here is the same case after applying the pull-up principle of the method calculateCapacity():

```java
package dat159.pullUpExample;

class Vehicle2 {
    String capacity;

    public String getCapacity() {
        return capacity;
    }

    /*
     * Returning capacity as String
     */
    public String calculateCapacity(int length, int width, int heigth){
        int number = (length*width*heigth)/2;
        return number + " seats.";
    }

}

class Bus2 extends Vehicle2{

    public Bus2(int length, int width, int heigth) {
        capacity = calculateCapacity(length, width, heigth);
    }
}

class Truck2 extends Vehicle2{

    public Truck2(int length, int width, int heigth) {
        capacity = calculateCapacity(length, width, heigth);
    }
}

public class AfterPullUp {
    public static void main(String[] args) {
        Bus2 bus = new Bus2(15, 3, 2);
        Truck2 truck = new Truck2(15, 3, 2);

        System.out.println("The truck has a capacity of " + truck.getCapacity());
        System.out.println("The bus has a capacity of " + bus.getCapacity());
    }

}
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

```
<terminated> AfterPullUp [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents
The truck has a capacity of 45 seats.
The bus has a capacity of 45 seats.
```

In this case, the two classes should have their separate methods and the pull-up principle should not be applied.