TataruTheodoraVerification

C00231174

IT Carlow
Software Development Year 4
Software Engineering
2020-2021

TABLE OF CONTENTS

Task 1

Junit Testing: https://drive.google.com/drive/folders/17a_FLvcCqagQ0RP5lyVXxThHWZrqwC_y?usp=sharing

Black box excel sheet:
https://docs.google.com/spreadsheets/d/1mIz9NA1eCGtw43GyCfAr5_GqI08lqHYJPDrPLGBcCJU/edit?usp=sharing

Constrains	
Period	
startHour >= 0	
startHour <= 24	
endHour >= 0	
endHour <= 24	
startHour < endHour	
Rate	
normalRate >= 0	
reducedRate >= 0	
normalRate >= reducedRate	
normalPeriods	
reducedPeriods	
normalPeriods do not overlap normalPeriods	
reducedPeriods do not overlap reducedPeriods	
reducedPeriods not overlap normalPeriods	
CarParkKind == Staff || Student || Management || Visitor	
	Test 1 - Rate
	normalRate = 3
	reducedRate = 1
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**normalRate >= reducedRate**
Expected Output	Valid rate object
	Test 2 - Rate
	normalRate = 1
	reducedRate = 1
	normalPeriods = (8-12) (17-20)

	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**normalRate == reducedRate**
Expected Output	Valid rate object
	Test 3 - Rate
	normalRate = 0
	reducedRate = 0
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**normalRate == 0**
Expected Output	Valid rate object
	Test 4 - Rate
	normalRate = 1
	reducedRate = 0
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**reducedRate == 0**
Expected Output	Valid rate object
	Test 5 - Rate
	normalRate = -1
	reducedRate = 1
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**normalRate < 0**
Expected Output	IllegalArgumentException
	Test 6 - Rate
	normalRate = 1
	reducedRate = -1
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT

Partitioned Test	**reducedRate < 0**
Expected Output	IllegalArgumentException
	Test 7 - Rate
	normalRate = 1
	reducedRate = 3
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**normalRate < reducedRate**
Expected Output	IllegalArgumentException
	Test 8 - Rate
	normalRate = 4
	reducedRate = 3
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**normalPeriods do not overlap the reducedPeriods**
Expected Output	Valid rate object
	Test 9 - Rate
	normalRate = 4
	reducedRate = 3
	normalPeriods = (8-15) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**normalPeriods overlap the reducedPeriods**
Expected Output	IllegalArgumentException
	Test 10 - Rate
	normalRate = 4
	reducedRate = 3
	normalPeriods = ()
	reducedPeriods = (8-12)
	kind = STUDENT
Partitioned Test	**normalPeriods == ()**
Expected Output	Valid rate object

	Test 11 - Rate
	normalRate = 4
	reducedRate = 3
	normalPeriods = (17-20)
	reducedPeriods = ()
	kind = STUDENT
Partitioned Test	**reducedPeriods== ()**
Expected Output	Valid rate object
	Test 12 - Rate
	normalRate = 4
	reducedRate = 3
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STAFF
Partitioned Test	**kind == STAFF**
Expected Output	Valid rate object
	Test 13 - Rate
	normalRate = 4
	reducedRate = 3
	normalPeriods = null
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**normalPeriods = null**
Expected Output	IllegalArgumentException
	Test 14 - Rate
	normalRate = 4
	reducedRate = 3
	normalPeriods = (8-12)
	reducedPeriods = null
	kind = STUDENT
Partitioned Test	**reducedPeriods = null**
Expected Output	IllegalArgumentException
	Test 15 - Rate
	normalRate = null

	reducedRate = 3
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**normalRate = null**
Expected Output	IllegalArgumentException
	Test 16 - Rate
	normalRate = 1
	reducedRate = null
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**reducedRate = null**
Expected Output	IllegalArgumentException
	Test 17 - Rate
	normalRate = 9
	reducedRate = 3.5
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**reducedRate = has decimal point**
Expected Output	Valid rate object
	Test 18 - Rate
	normalRate = 9.5
	reducedRate = 6
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**normalRate = has decimal point**
Expected Output	Valid rate object
	Test 19 - Rate
	normalRate = 9.5
	reducedRate = 6
	normalPeriods = (8-12) (11-15)

	reducedPeriods = (15-17)
	kind = STUDENT
Partitioned Test	**normalPeriods overlap**
Expected Output	IllegalArgumentException
	Test 20 - Rate
	normalRate = 9.5
	reducedRate = 6
	normalPeriods = (8-12)
	reducedPeriods = (12-17)(15-20)
	kind = STUDENT
Partitioned Test	**reducedPeriods overlap**
Expected Output	IllegalArgumentException
	Test 21 - Rate
	normalRate = 9.5
	reducedRate = 6
	normalPeriods = (12-8)
	reducedPeriods = (17-19)
	kind = STUDENT
Partitioned Test	**normalPeriods invalid (startRate > endRate)**
Expected Output	IllegalArgumentException
	Test 22 - Rate
	normalRate = 9.5
	reducedRate = 6
	normalPeriods = (8-12)
	reducedPeriods = (19-12)
	kind = STUDENT
Partitioned Test	**reducedPeriods invalid (startRate > endRate)**
Expected Output	IllegalArgumentException

	Test 23 - Rate
	normalRate = 9.5
	reducedRate = 6
	normalPeriods = (8-12)

	reducedPeriods = (19-19)
	kind = STUDENT
Partitioned Test	**reducedPeriods invalid (startRate == endRate)**
Expected Output	IllegalArgumentException
	Test 24 - Rate
	normalRate = 9.5
	reducedRate = 6
	normalPeriods = (12-12)
	reducedPeriods = (17-19)
	kind = STUDENT
Partitioned Test	**normalPeriods invalid (startRate == endRate)**
Expected Output	IllegalArgumentException

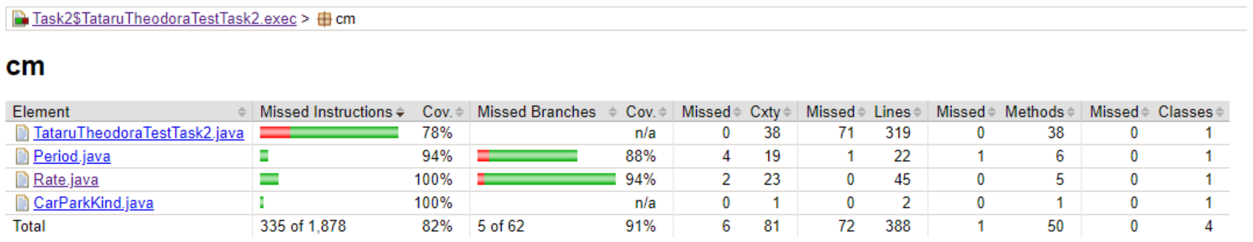
Precondition	**Rate object for calculate()**
	normalRate = 9
	reducedRate = 3
	normalPeriods = (8-12)
	reducedPeriods = (12-17)
	kind = VISITOR
	Test 1 - Calculate
	startHour = 11
	endHour = 12
Partitioned Test	**startHour < endHour**
Expected Output	9
	Test 2 - Calculate
	startHour = 0
	endHour = 20
Partitioned Test	**startHour == 0**
Expected Output	51
	Test 3 - Calculate
	startHour = 0
	endHour = 0
Partitioned Test	**endHour == 0**

Expected Output	IllegalArgumentException
	Test 4 - Calculate
	startHour = 17
	endHour = 17
Partitioned Test	**startHour == endHour**
Expected Output	IllegalArgumentException
	Test 5 - Calculate
	startHour = 19
	endHour = 17
Partitioned Test	**startHour > endHour**
Expected Output	IllegalArgumentException
	Test 6 - Calculate
	startHour = -2
	endHour = 13
Partitioned Test	**startHour < 0**
Expected Output	IllegalArgumentException
	Test 7 - Calculate
	startHour = 25
	endHour = 13
Partitioned Test	**startHour > 24**
Expected Output	IllegalArgumentException
	Test 8 - Calculate
	startHour = 13
	endHour = -1
Partitioned Test	**endHour < 0**
Expected Output	IllegalArgumentException
	Test 9 - Calculate
	startHour = 13
	endHour = 26
Partitioned Test	**endHour > 24**
Expected Output	IllegalArgumentException
	Test 10 - Calculate
	startHour = 3
	endHour = 24
Partitioned Test	**endHour == 24**

Expected Output	51
	Test 11 - Calculate
	startHour = 19
	endHour = 21
Partitioned Test	**startHour && endHour outside paying periods**
Expected Output	0
	Test 12 - Calculate
	startHour = 0
	endHour = 24
Partitioned Test	**startHour && endHour cover all day**
Expected Output	51

Task 2

Test coverage without any modifications



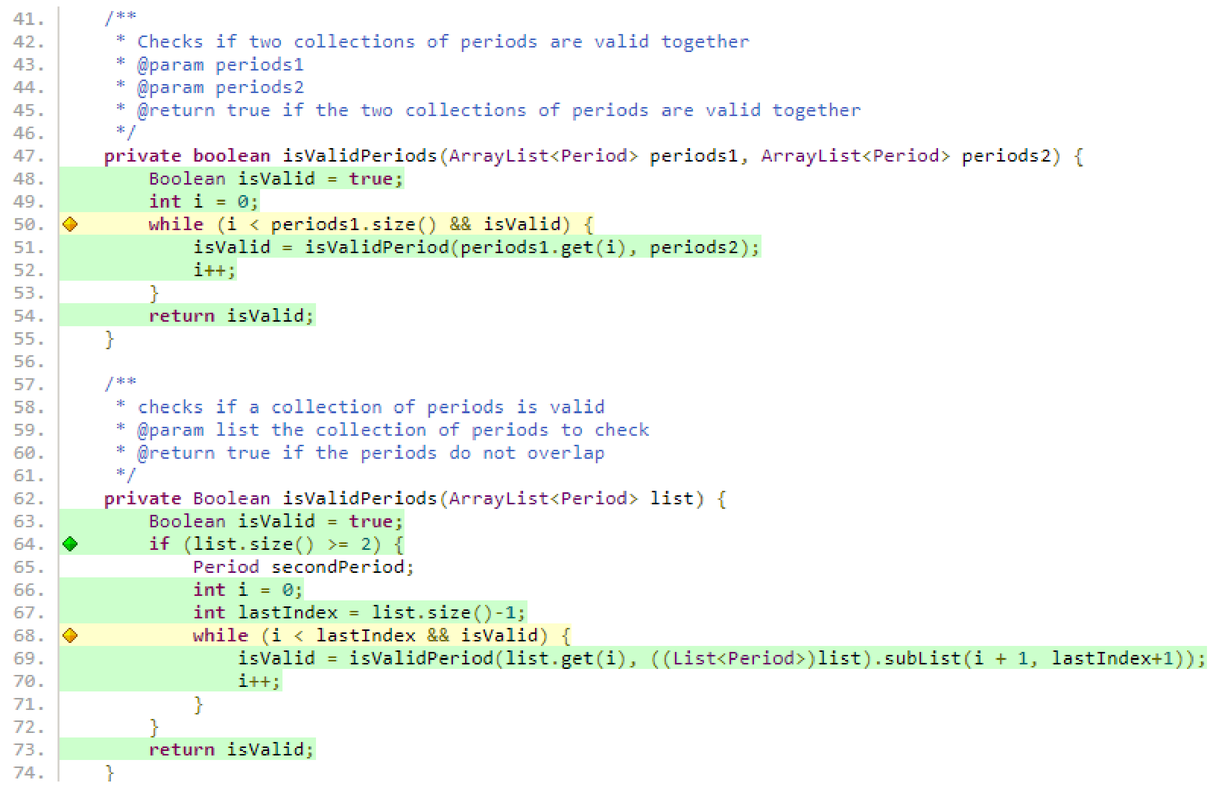
Task2$TataruTheodoraTestTask2.exec > cm

cm

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
TataruTheodoraTestTask2.java	🟥🟩	78%		n/a	0	38	71	319	0	38	0	1
Period.java	🟩	94%	🟥🟩	88%	4	19	1	22	1	6	0	1
Rate.java	🟩	100%	🟥🟩	94%	2	23	0	45	0	5	0	1
CarParkKind.java	🟩	100%		n/a	0	1	0	2	0	1	0	1
Total	335 of 1,878	82%	5 of 62	91%	6	81	72	388	1	50	0	4

Uncovered branches

Marked with **yellow** are the branches that were not fully covered.



```
    /**
     * Checks if two collections of periods are valid together
     * @param periods1
     * @param periods2
     * @return true if the two collections of periods are valid together
     */
    private boolean isValidPeriods(ArrayList<Period> periods1, ArrayList<Period> periods2) {
        Boolean isValid = true;
        int i = 0;
        while (i < periods1.size() && isValid) {
            isValid = isValidPeriod(periods1.get(i), periods2);
            i++;
        }
        return isValid;
    }

    /**
     * checks if a collection of periods is valid
     * @param list the collection of periods to check
     * @return true if the periods do not overlap
     */
    private Boolean isValidPeriods(ArrayList<Period> list) {
        Boolean isValid = true;
        if (list.size() >= 2) {
            Period secondPeriod;
            int i = 0;
            int lastIndex = list.size()-1;
            while (i < lastIndex && isValid) {
                isValid = isValidPeriod(list.get(i), ((List<Period>)list).subList(i + 1, lastIndex+1));
                i++;
            }
        }
        return isValid;
    }
```

White Box Testing

Changes or Additions

	Test 25 - Rate
	normalRate = 9.5
	reducedRate = 6
	normalPeriods = (9,12)(11-15)(15-18)
	reducedPeriods = (18-20)
	kind = STUDENT
Partitioned Test	**normalPeriods does overlap normalPeriods**
Expected Output	IllegalArgumentException
	Test 26 - Rate
	normalRate = 4
	reducedRate = 3
	normalPeriods = (8-15)(15-20)
	reducedPeriods = (19-21)(21-22)
	kind = STUDENT
Partitioned Test	**normalPeriods overlap the reducedPeriods**
Expected Output	IllegalArgumentException

Tests changed

	Test 2 - Calculate
	startHour = 0
	endHour = 20
Partitioned Test	**startHour == 0**
Expected Output	51

	Test 10 - Calculate
	startHour = 3
	endHour = 24
Partitioned Test	**endHour == 24**

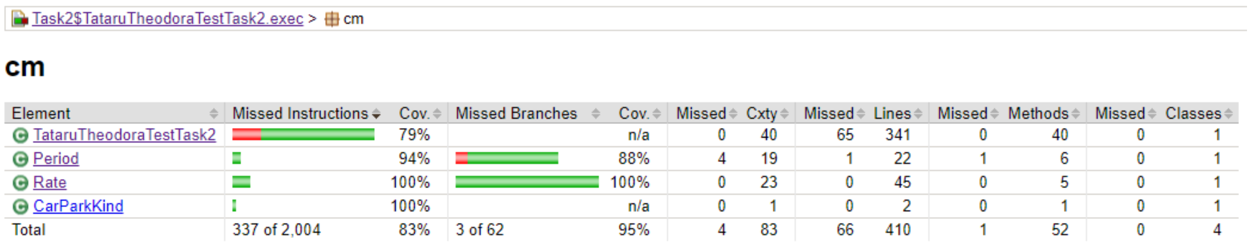
Expected Output	51

	Test 12 - Calculate
	startHour = 0
	endHour = 24
Partitioned Test	**startHour && endHour cover all day**
Expected Output	51

The tests were correct, but when I calculated the output, I wrote the wrong Expected Output, stating the output should be 60, while the correct Expected output should be 51. I checked the specifications, re-do the calculations and found that the mistake was done by me, when stating the expected output.

Test coverage after adding two more JUnit test

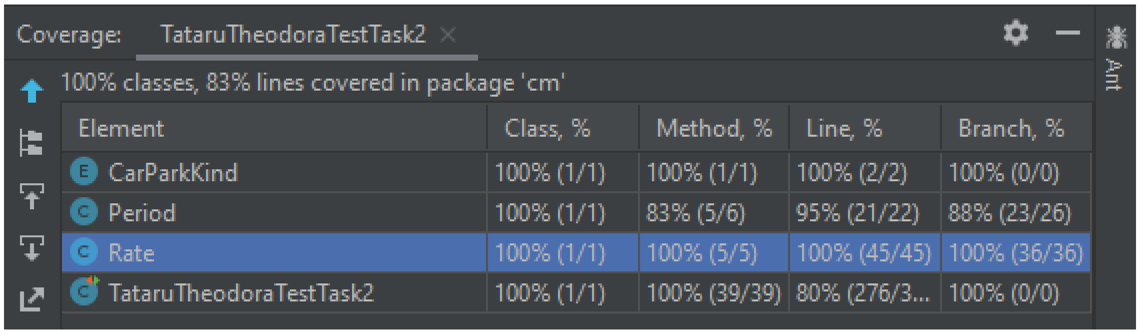
With the 2 new tests added above the following coverage for the Rate class was achieved:



Task2$TataruTheodoraTestTask2.exec > cm

cm

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
TataruTheodoraTestTask2		79%		n/a	0	40	65	341	0	40	0	1
Period		94%		88%	4	19	1	22	1	6	0	1
Rate		100%		100%	0	23	0	45	0	5	0	1
CarParkKind		100%		n/a	0	1	0	2	0	1	0	1
Total	337 of 2,004	83%	3 of 62	95%	4	83	66	410	1	52	0	4



Coverage: TataruTheodoraTestTask2

100% classes, 83% lines covered in package 'cm'

Element	Class, %	Method, %	Line, %	Branch, %
CarParkKind	100% (1/1)	100% (1/1)	100% (2/2)	100% (0/0)
Period	100% (1/1)	83% (5/6)	95% (21/22)	88% (23/26)
Rate	100% (1/1)	100% (5/5)	100% (45/45)	100% (36/36)
TataruTheodoraTestTask2	100% (1/1)	100% (39/39)	80% (276/3...	100% (0/0)

Bugs found in the code

The specification states that:

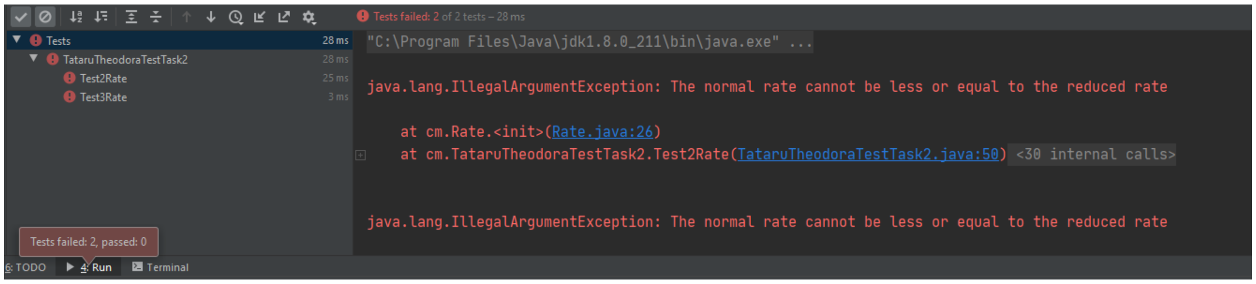
"Some additional constraints are:
- the normalRate and reducedRate are greater or equal to 0
- the normalRate has to be greater or equal to than the reducedRate
- the reducedPeriods and normalPeriods must be valid, i.e.
  - a collection of period must not overlap
  - the two collections must not overlap themselves"

According to the specification, the tests 2 and 3 should have passed.

Tests:

	Test 2 - Rate
	normalRate = 1
	reducedRate = 1
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**normalRate == reducedRate**
Expected Output	Valid rate object

	Test 3 - Rate
	normalRate = 0
	reducedRate = 0
	normalPeriods = (8-12) (17-20)
	reducedPeriods = (12-17)
	kind = STUDENT
Partitioned Test	**normalRate == 0**
Expected Output	Valid rate object



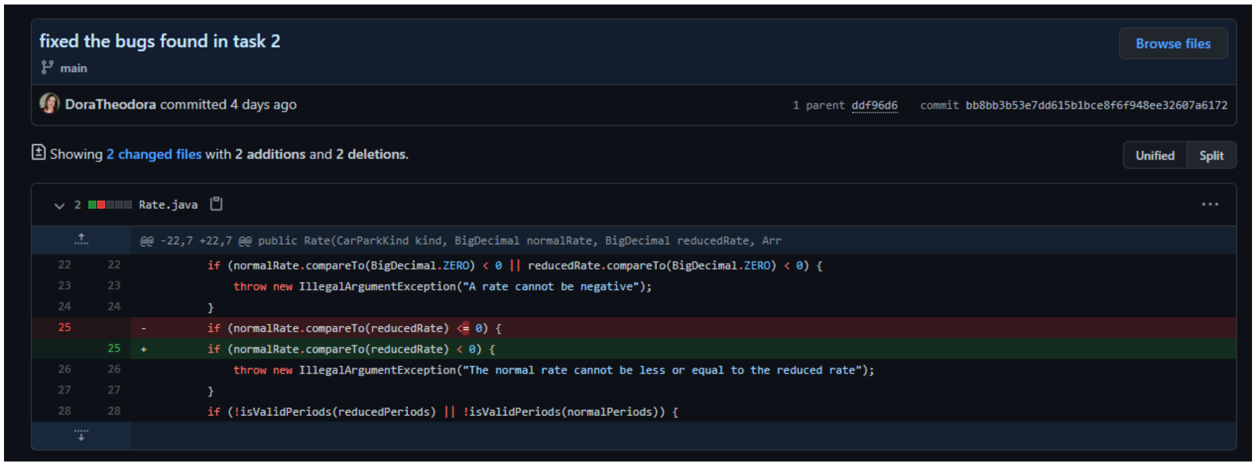
```
Tests failed: 2 of 2 tests – 28 ms
Tests                    28 ms
  TataruTheodoraTestTask2  28 ms
    Test2Rate              25 ms
    Test3Rate               3 ms

"C:\Program Files\Java\jdk1.8.0_211\bin\java.exe" ...

java.lang.IllegalArgumentException: The normal rate cannot be less or equal to the reduced rate

    at cm.Rate.<init>(Rate.java:26)
    at cm.TataruTheodoraTestTask2.Test2Rate(TataruTheodoraTestTask2.java:50) <30 internal calls>


java.lang.IllegalArgumentException: The normal rate cannot be less or equal to the reduced rate

Tests failed: 2, passed: 0
```

JUnit Tests

Link: https://drive.google.com/drive/folders/17a_FLvcCqagQ0RP5lyVXxThHWZrqwC_y?usp=sharing

Task 3

Bugs fixed



fixed the bugs found in task 2

main

DoraTheodora committed 4 days ago

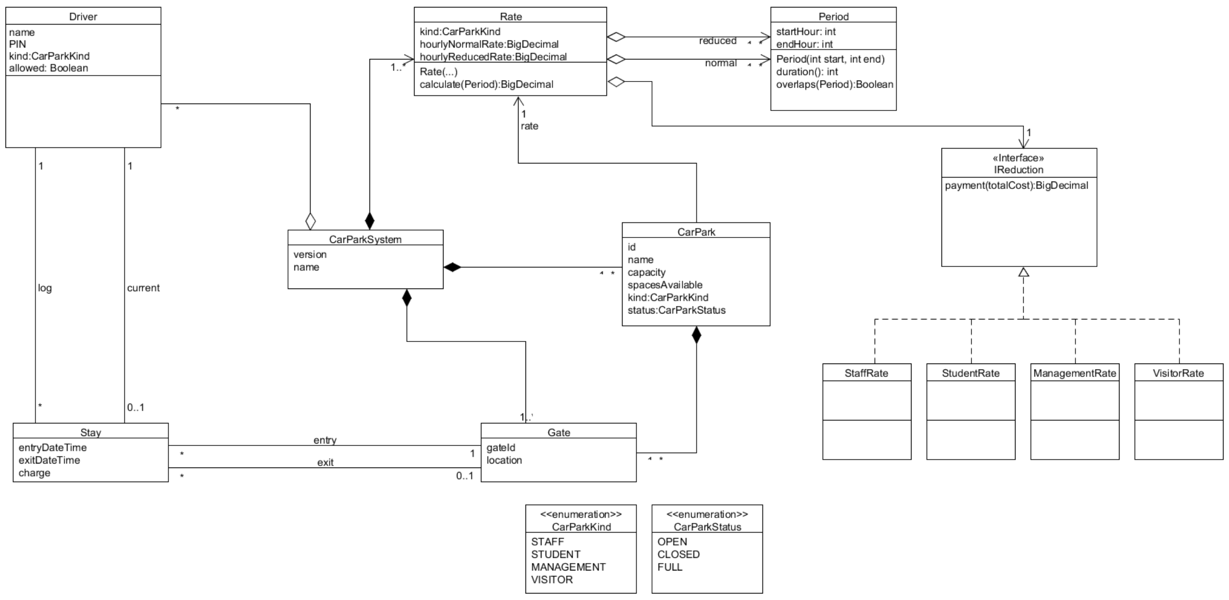
1 parent ddf96d6 commit bb8bb3b53e7dd615b1bce8f6f948ee32607a6172

Showing 2 changed files with 2 additions and 2 deletions.

Rate.java

```
@@ -22,7 +22,7 @@ public Rate(CarParkKind kind, BigDecimal normalRate, BigDecimal reducedRate, Arr
          if (normalRate.compareTo(BigDecimal.ZERO) < 0 || reducedRate.compareTo(BigDecimal.ZERO) < 0) {
              throw new IllegalArgumentException("A rate cannot be negative");
          }
-         if (normalRate.compareTo(reducedRate) <= 0) {
+         if (normalRate.compareTo(reducedRate) < 0) {
              throw new IllegalArgumentException("The normal rate cannot be less or equal to the reduced rate");
          }
          if (!isValidPeriods(reducedPeriods) || !isValidPeriods(normalPeriods)) {
```

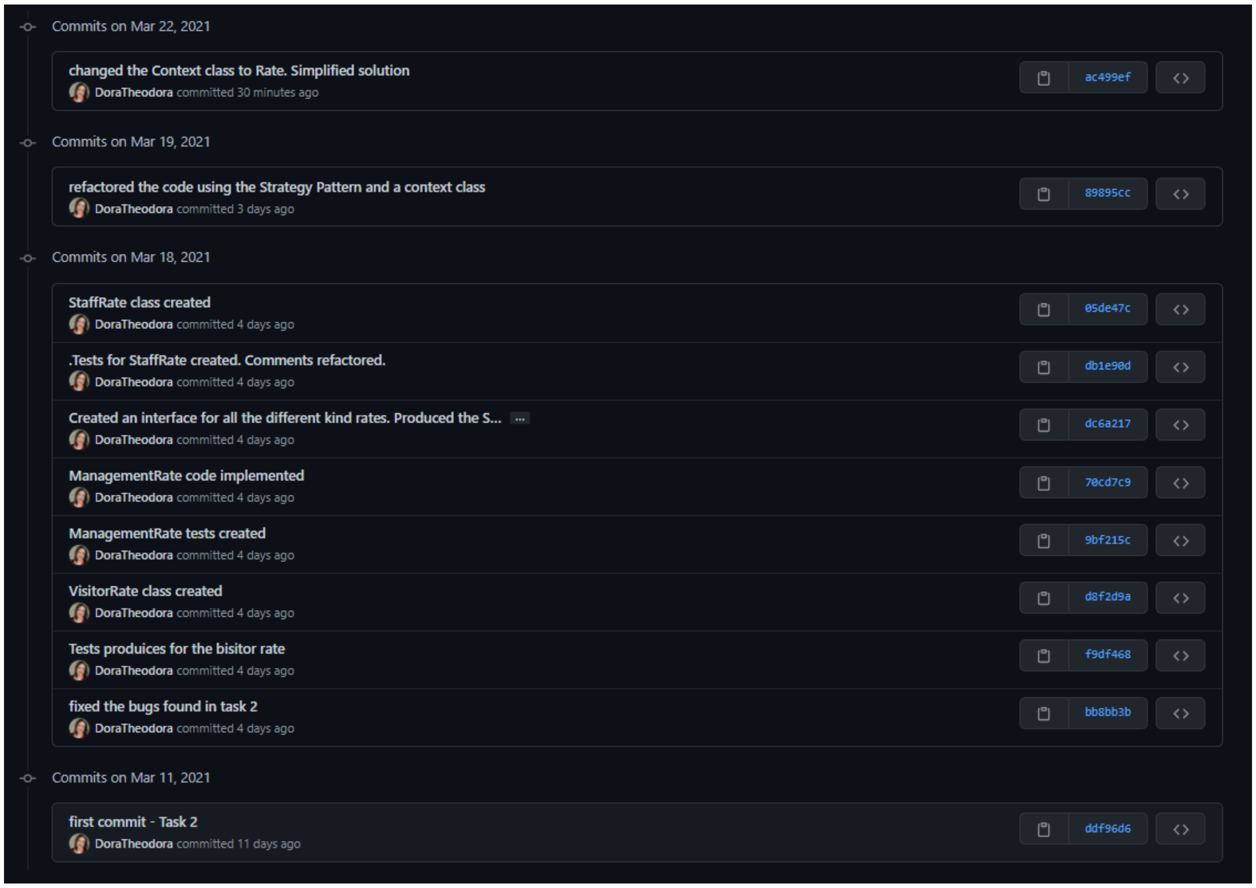
UML - Implementing Strategy Pattern



The Strategy Pattern was chosen for this implementation, as it permits selecting an algorithm at runtime. As each different KIND of rate implements a different algorithm, the strategy pattern facilitates selecting the algorithm at run time and reduces the code complexity, with the help of the IReduction interface, from which all the different kind of rate classes inherit from (ManagementRate, VisitorRate, StaffRate, and StudentRate). This way, if new kinds of rate classes will be created in the future, the new rate classes can be implemented without changing the code architecture.

GitHub: DoraTheodora/JUnit_Car_Park (github.com)

As requested by the specification document, the development of the code followed the Test Driven Development. This was achieved by developing the tests first, followed by the development of the code required to pass the tests.



Commits on Mar 22, 2021

changed the Context class to Rate. Simplified solution
DoraTheodora committed 30 minutes ago — ac499ef

Commits on Mar 19, 2021

refactored the code using the Strategy Pattern and a context class
DoraTheodora committed 3 days ago — 89895cc

Commits on Mar 18, 2021

StaffRate class created
DoraTheodora committed 4 days ago — 05de47c

.Tests for StaffRate created. Comments refactored.
DoraTheodora committed 4 days ago — db1e90d

Created an interface for all the different kind rates. Produced the S...
DoraTheodora committed 4 days ago — dc6a217

ManagementRate code implemented
DoraTheodora committed 4 days ago — 70cd7c9

ManagementRate tests created
DoraTheodora committed 4 days ago — 9bf215c

VisitorRate class created
DoraTheodora committed 4 days ago — d8f2d9a

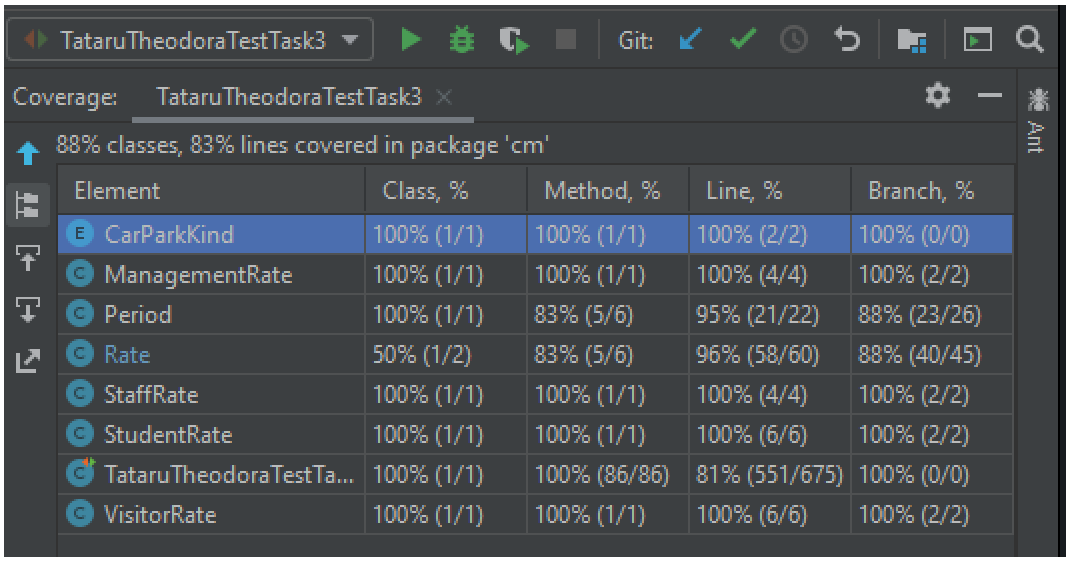
Tests produices for the bisitor rate
DoraTheodora committed 4 days ago — f9df468

fixed the bugs found in task 2
DoraTheodora committed 4 days ago — bb8bb3b

Commits on Mar 11, 2021

first commit - Task 2
DoraTheodora committed 11 days ago — ddf96d6

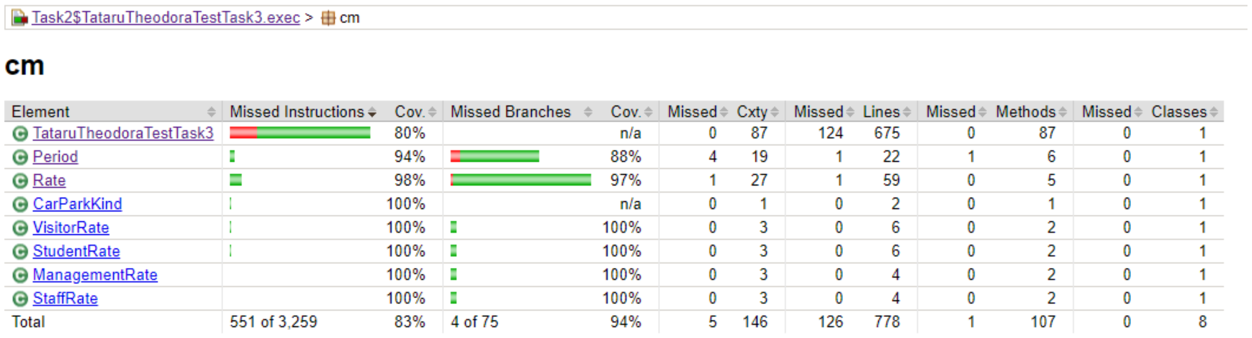
Test coverage after full implementation



88% classes, 83% lines covered in package 'cm'

Element	Class, %	Method, %	Line, %	Branch, %
CarParkKind	100% (1/1)	100% (1/1)	100% (2/2)	100% (0/0)
ManagementRate	100% (1/1)	100% (1/1)	100% (4/4)	100% (2/2)
Period	100% (1/1)	83% (5/6)	95% (21/22)	88% (23/26)
Rate	50% (1/2)	83% (5/6)	96% (58/60)	88% (40/45)
StaffRate	100% (1/1)	100% (1/1)	100% (4/4)	100% (2/2)
StudentRate	100% (1/1)	100% (1/1)	100% (6/6)	100% (2/2)
TataruTheodoraTestTa...	100% (1/1)	100% (86/86)	81% (551/675)	100% (0/0)
VisitorRate	100% (1/1)	100% (1/1)	100% (6/6)	100% (2/2)

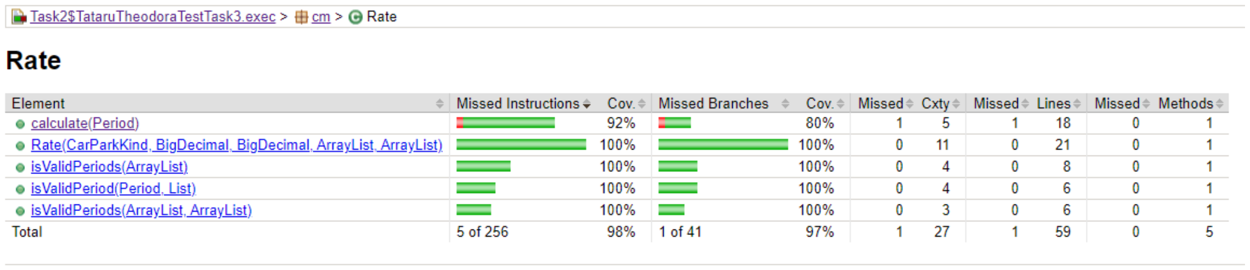
Test coverage breakdown



Task2$TataruTheodoraTestTask3.exec > cm

cm

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
TataruTheodoraTestTask3		80%		n/a	0	87	124	675	0	87	0	1
Period		94%		88%	4	19	1	22	1	6	0	1
Rate		98%		97%	1	27	1	59	0	5	0	1
CarParkKind		100%		n/a	0	1	0	2	0	1	0	1
VisitorRate		100%		100%	0	3	0	6	0	2	0	1
StudentRate		100%		100%	0	3	0	6	0	2	0	1
ManagementRate		100%		100%	0	3	0	4	0	2	0	1
StaffRate		100%		100%	0	3	0	4	0	2	0	1
Total	551 of 3,259	83%	4 of 75	94%	5	146	126	778	1	107	0	8



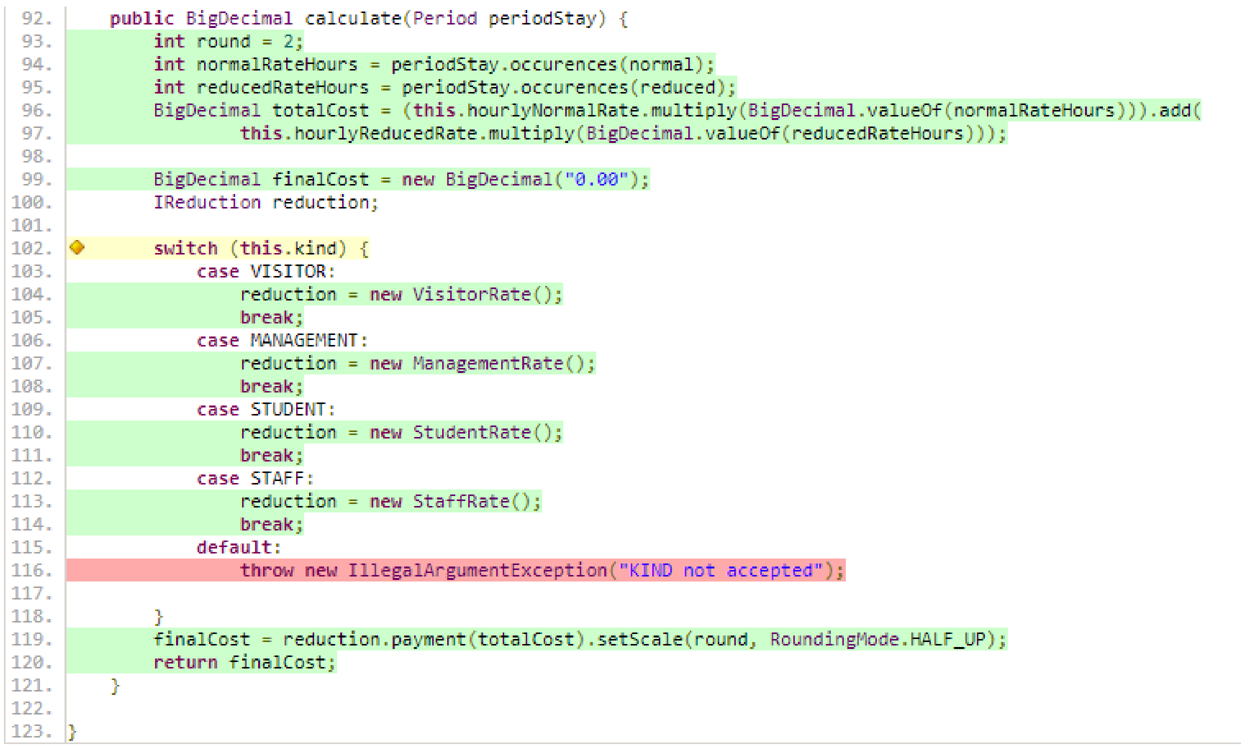
Task2$TataruTheodoraTestTask3.exec > cm > Rate

Rate

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
calculate(Period)		92%		80%	1	5	1	18	0	1
Rate(CarParkKind, BigDecimal, BigDecimal, ArrayList, ArrayList)		100%		100%	0	11	0	21	0	1
isValidPeriods(ArrayList)		100%		100%	0	4	0	8	0	1
isValidPeriod(Period, List)		100%		100%	0	4	0	6	0	1
isValidPeriods(ArrayList, ArrayList)		100%		100%	0	3	0	6	0	1
Total	5 of 256	98%	1 of 41	97%	1	27	1	59	0	5

As seen marked with **yellow**, the **switch** statement is not covered by the unit tests, but they cover all the SWITCH cases; therefore, it is considered that the tests produced for the code are covering all the branches.

With red is marked, the branch that is not covered by the unit tests. This branch could never be tested as KIND is an enumerator, therefore, a non-existing kind cannot be used to create a Rate object.



```java
    public BigDecimal calculate(Period periodStay) {
        int round = 2;
        int normalRateHours = periodStay.occurences(normal);
        int reducedRateHours = periodStay.occurences(reduced);
        BigDecimal totalCost = (this.hourlyNormalRate.multiply(BigDecimal.valueOf(normalRateHours))).add(
                this.hourlyReducedRate.multiply(BigDecimal.valueOf(reducedRateHours)));

        BigDecimal finalCost = new BigDecimal("0.00");
        IReduction reduction;

        switch (this.kind) {
            case VISITOR:
                reduction = new VisitorRate();
                break;
            case MANAGEMENT:
                reduction = new ManagementRate();
                break;
            case STUDENT:
                reduction = new StudentRate();
                break;
            case STAFF:
                reduction = new StaffRate();
                break;
            default:
                throw new IllegalArgumentException("KIND not accepted");

        }
        finalCost = reduction.payment(totalCost).setScale(round, RoundingMode.HALF_UP);
        return finalCost;
    }

}
```

Test Driven Development Process

Developing a program using the Test Driven Development (TDD) for the first time was interesting, different, challenging at the beginning, and provided me a different perspective over software development.

Reading the specification and creating unit tests before development was challenging initially, but over the whole process, it helped tremendously in developing the code.

I found TASK 1 the hardest, as to compile the unit tests according to the specification, I had to implement dummy classes. Creating dummy classes is a straightforward process, but being accustomed to implementing complex code at first, I found the task very challenging. After extended research, I was able to ease and convince myself that is a very healthy way of producing code.

Creating the unit tests first was the best way of verifying the specification provided, and when the implementation of the code was finally produced, the specification was clear.

Usually, until now, I would implement the code according to its specification and spend hours, if not days, designing the code and working on development, constantly altering the code as I was spotting mistakes. Developing the unit tests first, all these obstacles were not present, or they were smaller, and the development time was reduced considerably.

TASK 2 was very useful to see how the code produced did not meet the specification, as according to the specification document. Two bugs were found in the implementation, with the unit tests' help developed before the implementation.

For TASK 3, creating the tests and running them initially without all the code implementation was very intimidating, as most tests failed. But as I mentioned before, the implementation afterward was fast, and with the help of unit tests, I was able to find my mistakes easily. Writing the tests at this stage was straightforward, as most parts of the specification were well understood. Writing the tests was performed in small repetitive steps, taking one kind of rate at a time. The following process was to implement the corresponding functionally to match the tests and rerun the tests to check the outcome. All this process required discipline as I was very tempted to write additional code that was not covered by the existing unit tests.

Also, the TDD changed my perspective over the code architecture, allowing me to question the code design and pushed me to research different patterns and strategies to refactor the code to obtain the desired functionality. This step was as well challenging, intimating, and different as this was the first time I actually implemented a pattern for a "real-world" example. After extensive research, I chose the Strategy Pattern. I believe that choosing an algorithm at run-time would benefit the program, as the rate is calculated differently depending on the kind. Implementing the Strategy Pattern, the architecture of the code consists now of highly cohesive and loosely coupled components.

When refactoring the code to implement the Strategy Pattern, the unit tests were extremely helpful to ensure that all the implemented code changes did not alter its functionality. With every slight change performed over the implementation, the tests were re-run. Of course, during this process, many times, the tests failed, allowing me to spot the bugs and mistakes in the implementation.

Overall, TDD exposed me to a new way of writing code, a way that I believe benefits the development. The practice of using TDD enabled me to be more productive and understand the specification faster and better. After writing the tests, the implementation of code was also quicker, along with finding bugs in the code and fixing them. Refactoring the code to change its architecture is a risky process, but using TDD, I was able to find and solve all the problems raised by this process.