THEORICAL WORK 2 SOFTWARE ENGENEERING II

GROUP: A03

Carlos Pulido Hernández
Alberto Quintana Algaba

Content

ET.02.03 Second Problem Description

In its new public transport plan of the JCCM, we have been asked to implement an application that determines the ticket price of the transport, depending on the health and age conditions of a person, and depending on the state of the pandemic. In this sense, the AI of Castilla-La Mancha will be monitored (we can assume it to be "constant" during the execution of the project, which will require an initialization of the environment), with a reduction of seats as follows: if it is less than 100 there will be no space restrictions (level 0), if it is between 100 and 200 (level 1) the capacity of the means of transport is reduced to 80%, if it is between 201 and 300 (level 2), it is reduced to 60%, if it is between 301 and 500 (level 3) the capacity is reduced to 40%, and if it is higher than 501 (level 4) to 30%. In order to avoid unnecessary movements, an increase in the ticket price will be established. For this purpose, the following rules will be used to calculate the ticket price:

• Regardless of AI status, a person who is ill, has recent contact within the last 10 days with infected persons, or has suspected symptoms of COVID will not be allowed to travel. A person with a COVID passport and not ill may travel if space is available, regardless of their occupation type.

• If there are no space restrictions (level 0), any person, regardless of age, may travel, and will receive a 60% discount if under 23 years of age, and an 80% discount if over 65 years of age.

• In Level 1, those under 23 years of age will have a 30% discount, and those over 65 will have a 50% discount. No transportation priorities are established, but places are reduced, so that a price will only be given (consider implement and throw some kind of exception) if there are places available.

• In Level 2, those under 23 years of age will have no discount, and those over 65 years of age will have an increase of 20%. At this level, of the possible reduced capacity, 60% of the places are reserved for professionals with essential professions.

• At Level 3, those under 23 years of age will have a 20% surcharge, and those over 65 years of age will have a 50% increase. At this capacity level, 80% of the available reduced space is reserved for essential professionals.

• At Level 4, those under 23 years of age will have a 50% surcharge, and those over 65 years of age will not be allowed to travel. A 90% space is reserved for people with essential professions.

It is requested:

1) Write, at least the pseudocode of the identified method.

2) Identify the variables that must be considered to test the method.

3) Identify the test values for each one of the variables previously identified, specifying the technique used to obtain each of those values).

4) Calculate the maximum possible number of test cases that could be generated from the test values.

5) Define some test suites using each use.

6) Define test suits to achieve pairwise coverage by using the proposed algorithm in Lectures. You can check the results by means of the software PICT1.

7) For code snippets that include decisions, propose a set of test cases to achieve coverage of decisions.

8) For code snippets that include decisions, propose test case sets to achieve MC/DC coverage.

9) Comment on the results of the number of test cases obtained in section 4, 5, and 6, as well as the execution of the oracles: what could be said about the coverage achieved?

1) Write, at least the pseudocode of the identified method.

```java
public double calculateTicketCost(Person person) throws
NoSeatsAvailableException {
		double price = 0;

		if(this.level == 0) {
			if(person.getAge() < 23) {
				price = 0.4 * this.ticketPrice;
			}else if(person.getAge() > 65) {
				price = 0.2 * this.ticketPrice;
			}else {
				price = this.ticketPrice;
			}
		}else if(this.level == 1){
			if(person.getAge() < 23) {
				price = 0.7 * this.ticketPrice;
			}else if (person.getAge() > 65) {
				price = 0.5 * this.ticketPrice;
			}else {
				price = this.ticketPrice;
			}
		}else if(this.level == 2) {
			if (person.getAge() > 65) {
				price = 1.2 * this.ticketPrice;
			}else {
				price = this.ticketPrice;
			}
		}else if(this.level == 3) {
			if(person.getAge() < 23) {
				price = 1.2 * this.ticketPrice;
			}else if (person.getAge() > 65) {
				price = 1.5 * this.ticketPrice;
			}else {
				price = this.ticketPrice;
			}
		}else{
			if(person.getAge() < 23) {
				price = 1.5 * this.ticketPrice;
			}else if (person.getAge() > 65) {
				throw new NoSeatsAvailableException("You are not
allowed to travel");
			}else {
				price = this.ticketPrice;
			}
		}
```

```
            return price;
    }
```

2) Identify the variables that must be considered to test the method.

There are variables that correspond to different classes:

- ***Transport***
  - level: According to the level we will have one specific discount.
  - normalSeats: Total number of seats after reduction.
  - essentialProfessionsSeats: Seats that are reserved for professionals.
- ***Person***
  - age: Determine the discount applying for the group age.
  - essentialProfession: Determine if it is an essential profession or not.
  - COVIDpassport: Represents the COVID passport for travelling
  - Ill: Represents if the person is ill or has symptoms

However, our method to test is to calculate the price of the ticket Transport. So, we will need the variables that determine the ticket cost:

- ***Transport***
  - level: According to the level we will have one specific discount.
- ***Person***
  - age: Determine the discount applying for the group age.

The rest of the variables are used for initialization of the environment (check health conditions and assign a seat to the person).

3) Identify the test values for each one of the variables previously identified, specifying the technique used to obtain each of those values).

Integer = [-2147483648, 2147483647]

level = (-Inf, -2147483648), [-2147483648, 0), [0], [1], [2], [3], [4], (4, 2147483647], (2147483647, +Inf)

age = (-Inf, -2147483648), [-2147483648, 23), [23, 65), [65, 2147483647], (2147483647, +Inf)

VARIABLES	ERROR GUESSING	BOUNDARY VALUES	
level	-2247483648,0,2247483648	-2147483648,0,1,2,3,4 2147483647	-2147483649, -2147483648, -2147483647,

			-1,0,1,2,3,4,5 2147483646, 2147483647, 2147483648,
age	-2247483648,0,30,69 2247483648	-2147483648,23,65, 2147483647	-2147483649, -2147483648, -2147483647, 22,23,24,64,65,66, 2147483646, 2147483647, 2147483648,

4) Calculate the maximum possible number of test cases that could be generated from the test values.

In this case, as we are dealing with integers that can get any value from -Inf to +Inf, we can have infinite number of test cases.

If we consider the test values previously generated:

level: -2247483648,-2147483649,-2147483648,-2147483647,-1,0,1,2,3,4,5,2147483646, 2147483647,2147483648,2247483648. ***(There are 15 possible values).***

age:-2247483648,-2147483649,-2147483648,- 2147483647,22,23,24,64,65,66,2147483646, 2147483647,2147483648,0,30,69,2247483648. ***(There are 17 possible values).***

Maximum number of combinations: 15*17 = 255 test cases. This test cases will represent most of the scenarios. So, we have reduced the number of cases significantly.

For calculating the ticket cost there are two preconditions:

- The person must be healthy. COVIDpassport variable *true* and ill variable *false.*

- There musts be seats to assign to the person. normalSeats > 0 if not a person with an essential profession and essentialProfessionsSeats > 0 if a person with an essential profession.

5) Define some test suites using each use.

{level,age}

Test suite 1: {0, 23}

Test suite 2: {1, 66}

Test suite 3: {2, 65}

Test suite 4: {3, 64}

Test suite 5: {4, 24}

These are some test suites, however, to reach each use coverage, 17 test cases are needed (each value of a variable must be taken at least once, so the number of test cases must be at least the maximum values a variable can take).

6) Define test suits to achieve pairwise coverage by using the proposed algorithm in Lectures. You can check the results by means of the software PICT1.

Firstly, we have to consider the values each variable can take. Once this is done, we must achieve pairwise coverage. Due to the number of variables and the number of values they can take, we will make use software PICT1. In the file test-results there are a total of 255 test cases. The solution is in the file named "pairwise-result".

7) For code snippets that include decisions, propose a set of test cases to achieve coverage of decisions.

```
if(person.getAge() < 23) {
      price = 0.4 * this.ticketPrice;
}else if(person.getAge() > 65) {
      price = 0.2 * this.ticketPrice;
}else {
      price = this.ticketPrice;
}
```

age	DECISION	EXPECTED
23	False, False, True	ticketPrice
65	False, False, True	ticketPrice
66	False, True, False	ticketPrice * 0.2
22	True, False, False	ticketPrice * 0.4

Test suite 1: {0, 65}
Test suite 2: {0, 66}
Test suite 3: {0, 22}

```
if(person.getAge() < 23) {
      price = 0.7 * this.ticketPrice;
}else if (person.getAge() > 65) {
      price = 0.5 * this.ticketPrice;
}else {
      price = this.ticketPrice;
}
```

age	DECISION	EXPECTED
23	False, False, True	ticketPrice
65	False, False, True	ticketPrice
66	False, True, False	ticketPrice * 0.5
22	True, False, False	ticketPrice * 0.7

Test suite 4: {1, 65}
Test suite 5: {1, 66}
Test suite 6: {1, 22}

```
if (person.getAge() > 65) {
	price = 1.2 * this.ticketPrice;
}else {
	price = this.ticketPrice;
}
```

age	DECISION	EXPECTED
23	False, True	ticketPrice
65	False, True	ticketPrice
66	True, False	ticketPrice * 1.2
22	False, True	ticketPrice

Test suite 7: {2, 65}
Test suite 8: {2, 66}

```
if(person.getAge() < 23) {
	price = 1.2 * this.ticketPrice;
}else if (person.getAge() > 65) {
	price = 1.5 * this.ticketPrice;
}else {
	price = this.ticketPrice;
}
```

age	DECISION	EXPECTED
23	False, False, True	ticketPrice
65	False, False, True	ticketPrice
66	False, True, False	ticketPrice * 1.5
22	True, False, False	ticketPrice * 1.2

Test suite 9: {3, 65}
Test suite 10: {3, 66}
Test suite 11: {3, 22}

```
if(person.getAge() < 23) {
	price = 1.5 * this.ticketPrice;
}else if (person.getAge() > 65) {
	throw new NoSeatsAvailableException("You are not allowed to travel");
}else {
	price = this.ticketPrice;
}
```

age	DECISION	EXPECTED
23	False, False, True	ticketPrice
65	False, False, True	ticketPrice
66	False, True, False	NoSeatsAvailableException
22	True, False, False	ticketPrice * 1.5

Test suite 12: {4, 65}
Test suite 13: {4, 66}
Test suite 14: {4, 22}

8) For code snippets that include decisions, propose test case sets to achieve MC/DC coverage.

```
if(this.level == 0) {
                    if(person.getAge() < 23) {
                          price = 0.4 * this.ticketPrice;
                    }else if(person.getAge() > 65) {
                          price = 0.2 * this.ticketPrice;
                    }else {
                          price = this.ticketPrice;
                          }
```

this.level == 0	Person.getAge() < 23	DECISION	DOMINANT CONDITION
True	True	True	A,B
True	False	False	B
False	True	False	A
False	False	False	A

Test suite 1: {0, 22}
Test suite 2: {0, 23}
Test suite 3: {1, 22}

this.level == 0	Person.getAge() > 65	DECISION	DOMINANT CONDITION
True	True	True	A,B
True	False	False	B
False	True	False	A
False	False	False	A

Test suite 4: {0, 66}
Test suite 5: {0, 65}
Test suite 6: {1, 66}

```
else if(this.level == 1){
                        if(person.getAge() < 23) {
                                price = 0.7 * this.ticketPrice;
                        }else if (person.getAge() > 65) {
                                price = 0.5 * this.ticketPrice;
                        }else {
                                price = this.ticketPrice;
                        }
```

this.level == 1	**Person.getAge() < 23**	**DECISION**	**DOMINANT CONDITION**
True	True	True	A,B
True	False	False	B
False	True	False	A
False	False	False	A

Test suite 2: {1, 22}
Test suite 7: {1, 23}
Test suite 8: {2, 22}

this.level == 1	**Person.getAge() > 65**	**DECISION**	**DOMINANT CONDITION**
True	True	True	A,B
True	False	False	B
False	True	False	A
False	False	False	A

Test suite 6: {1, 66}
Test suite 9: {1, 65}
Test suite 10: {2, 66}

```
}else if(this.level == 2) {
                        if (person.getAge() > 65) {
                                price = 1.2 * this.ticketPrice;
                        }else {
                                price = this.ticketPrice;
                        }
```

this.level == 2	**Person.getAge() > 65**	**DECISION**	**DOMINANT CONDITION**
True	True	True	A,B
True	False	False	B
False	True	False	A
False	False	False	A

Test suite 10: {2, 66}
Test suite 11: {2, 65}
Test suite 12: {3, 66}

```
else if(this.level == 3) {
                        if(person.getAge() < 23) {
                                price = 1.2 * this.ticketPrice;
                        }else if (person.getAge() > 65) {
                                price = 1.5 * this.ticketPrice;
                        }else {
                                price = this.ticketPrice;
                        }
```

this.level == 3	**Person.getAge() < 23**	**DECISION**	**DOMINANT CONDITION**
True	True	True	A,B
True	False	False	B
False	True	False	A
False	False	False	A

Test suite 13: {3, 22}
Test suite 14: {3, 23}
Test suite 15: {4, 22}

this.level == 3	**Person.getAge() > 65**	**DECISION**	**DOMINANT CONDITION**
True	True	True	A,B
True	False	False	B
False	True	False	A
False	False	False	A

Test suite 12: {3, 66}
Test suite 16: {3, 65}
Test suite 17: {4, 66}

```
else{
                        if(person.getAge() < 23) {
                                price = 1.5 * this.ticketPrice;
                        }else if (person.getAge() > 65) {
                                throw new NoSeatsAvailableException("You are not
allowed to travel");
                        }else {
                                price = this.ticketPrice;
                        }
                }
```

else	**Person.getAge() < 23**	**DECISION**	**DOMINANT CONDITION**
True	True	True	A,B
True	False	False	B
False	True	False	A
False	False	False	A

Test suite 15: {4, 22}

`Test suite 18:` {4, 23}
`Test suite 1:` {0, 22}

else	**Person.getAge() > 65**	**DECISION**	**DOMINANT CONDITION**
True	True	True	A,B
True	False	False	B
False	True	False	A
False	False	False	A

`Test suite 17:` {4, 66}
`Test suite 19:` {4, 65}
`Test suite 4:` {0, 66}

Some of the test suites can be reused for different if structures. In those cases, the number of the test suite will be repeated.

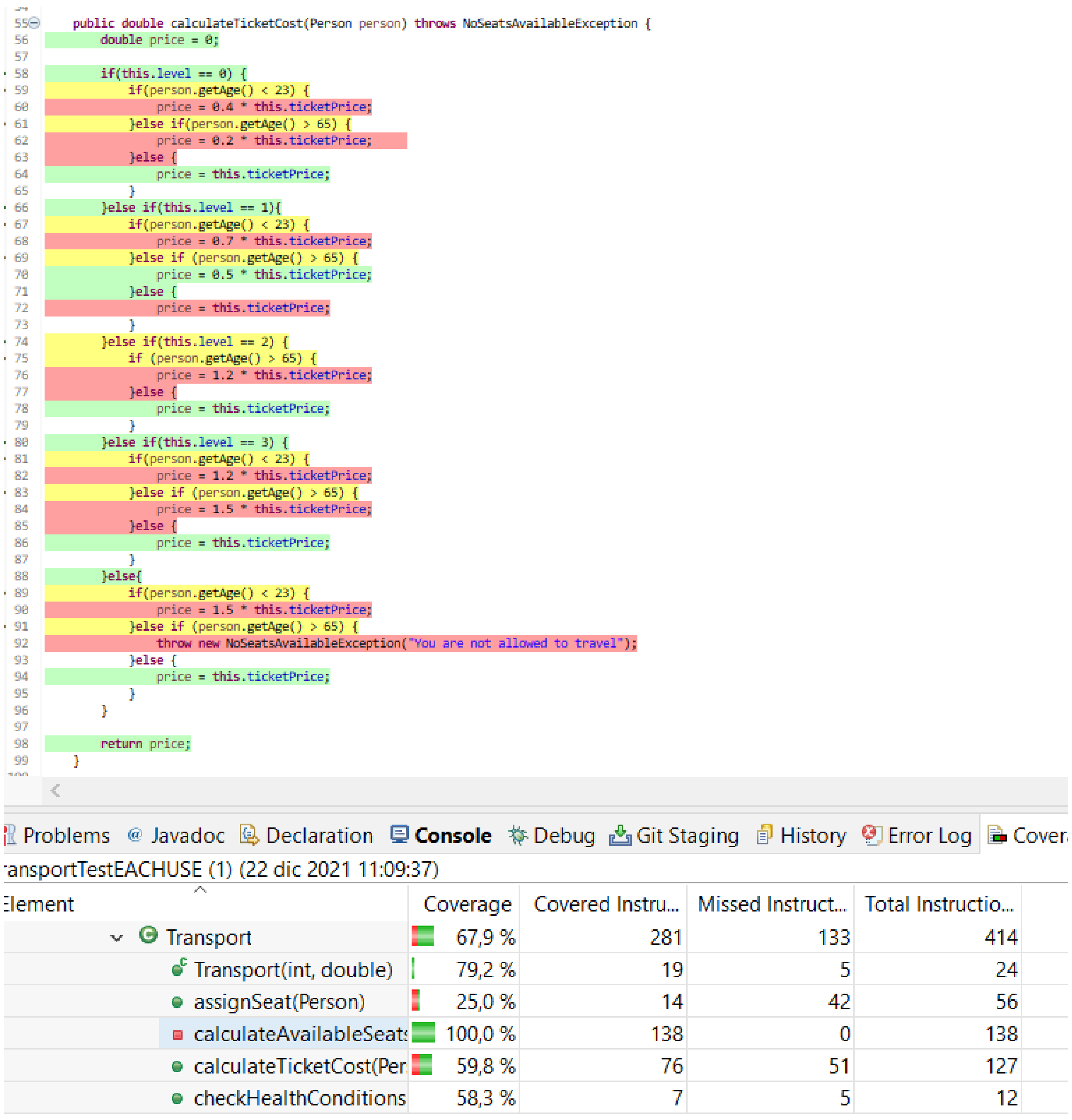
9) Comment on the results of the number of test cases obtained in section 4, 5, and 6, as well as the execution of the oracles: what could be said about the coverage achieved?

The number of test cases in the case of pairwise is the higher one. However, this is the expected one as we have several variables that are integers and can take several values (we got the values with techniques as error guessing and boundary values). In addition, decision coverage is the one with least test cases (12).

Tests done:

- Each use coverage (17 test cases)
- Pairwise (10 test cases, just a set is implemented)
- Decision coverage (12 test cases)
- MC/DC coverage (19 test cases)

EACH USE



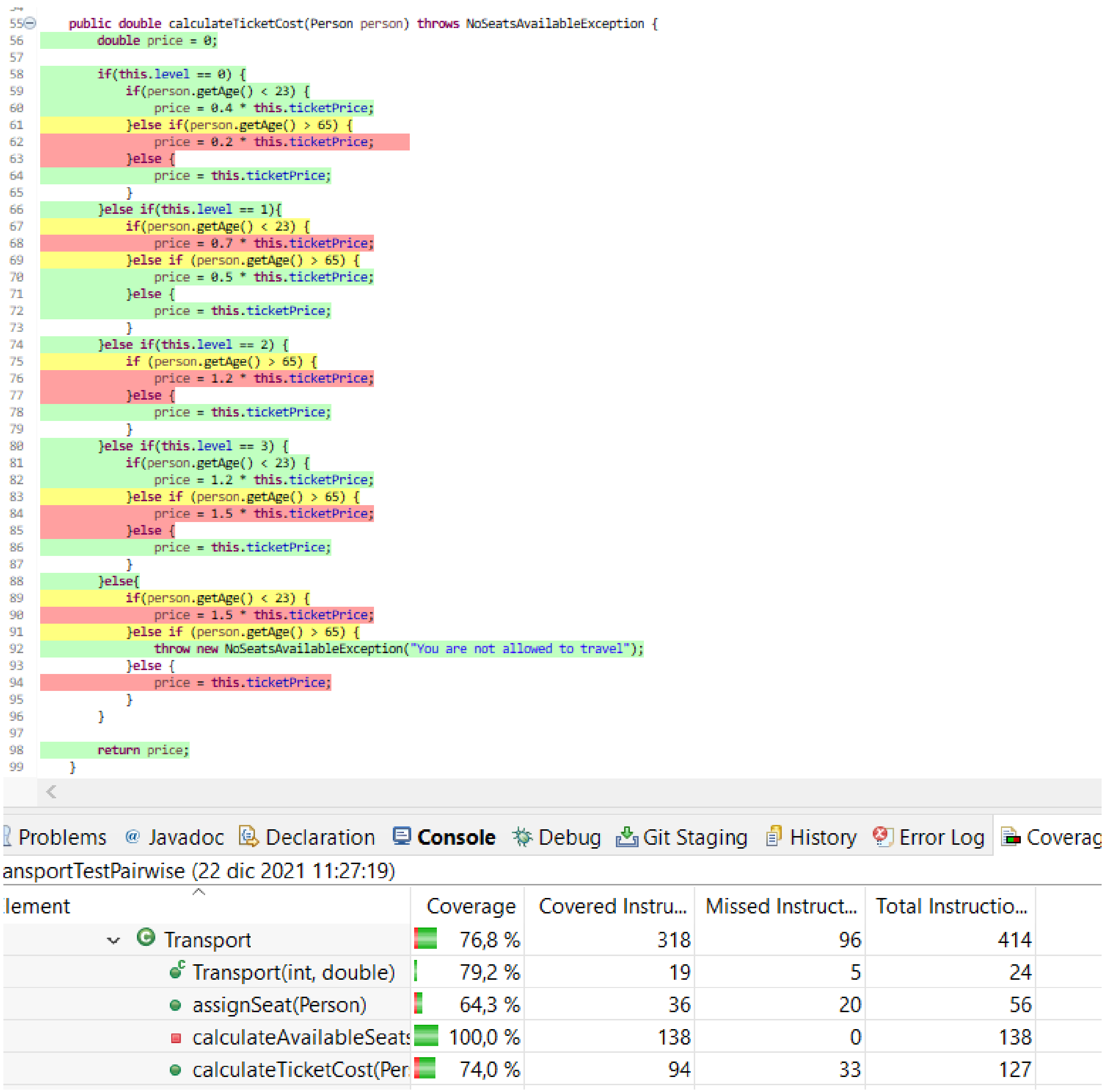
```java
    public double calculateTicketCost(Person person) throws NoSeatsAvailableException {
        double price = 0;

        if(this.level == 0) {
            if(person.getAge() < 23) {
                price = 0.4 * this.ticketPrice;
            }else if(person.getAge() > 65) {
                price = 0.2 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else if(this.level == 1){
            if(person.getAge() < 23) {
                price = 0.7 * this.ticketPrice;
            }else if (person.getAge() > 65) {
                price = 0.5 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else if(this.level == 2) {
            if (person.getAge() > 65) {
                price = 1.2 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else if(this.level == 3) {
            if(person.getAge() < 23) {
                price = 1.2 * this.ticketPrice;
            }else if (person.getAge() > 65) {
                price = 1.5 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else{
            if(person.getAge() < 23) {
                price = 1.5 * this.ticketPrice;
            }else if (person.getAge() > 65) {
                throw new NoSeatsAvailableException("You are not allowed to travel");
            }else {
                price = this.ticketPrice;
            }
        }

        return price;
    }
```

ansportTestEACHUSE (1) (22 dic 2021 11:09:37)

lement	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
Transport	67,9 %	281	133	414
Transport(int, double)	79,2 %	19	5	24
assignSeat(Person)	25,0 %	14	42	56
calculateAvailableSeat	100,0 %	138	0	138
calculateTicketCost(Per	59,8 %	76	51	127
checkHealthConditions	58,3 %	7	5	12

The coverage is almost 60% of the method ***"calculateTicketCost".*** This will depend on the way we choose the values (Values must be used at least once to reach each use coverage). So, to reach a high coverage for the method you must carefully select the test cases to be implemented or repeating some values. The coverage is not as high in this case because most of the values have problems (level is negative or out of int limits and most of the time doesn't even enter to our test method).

PAIRWISE



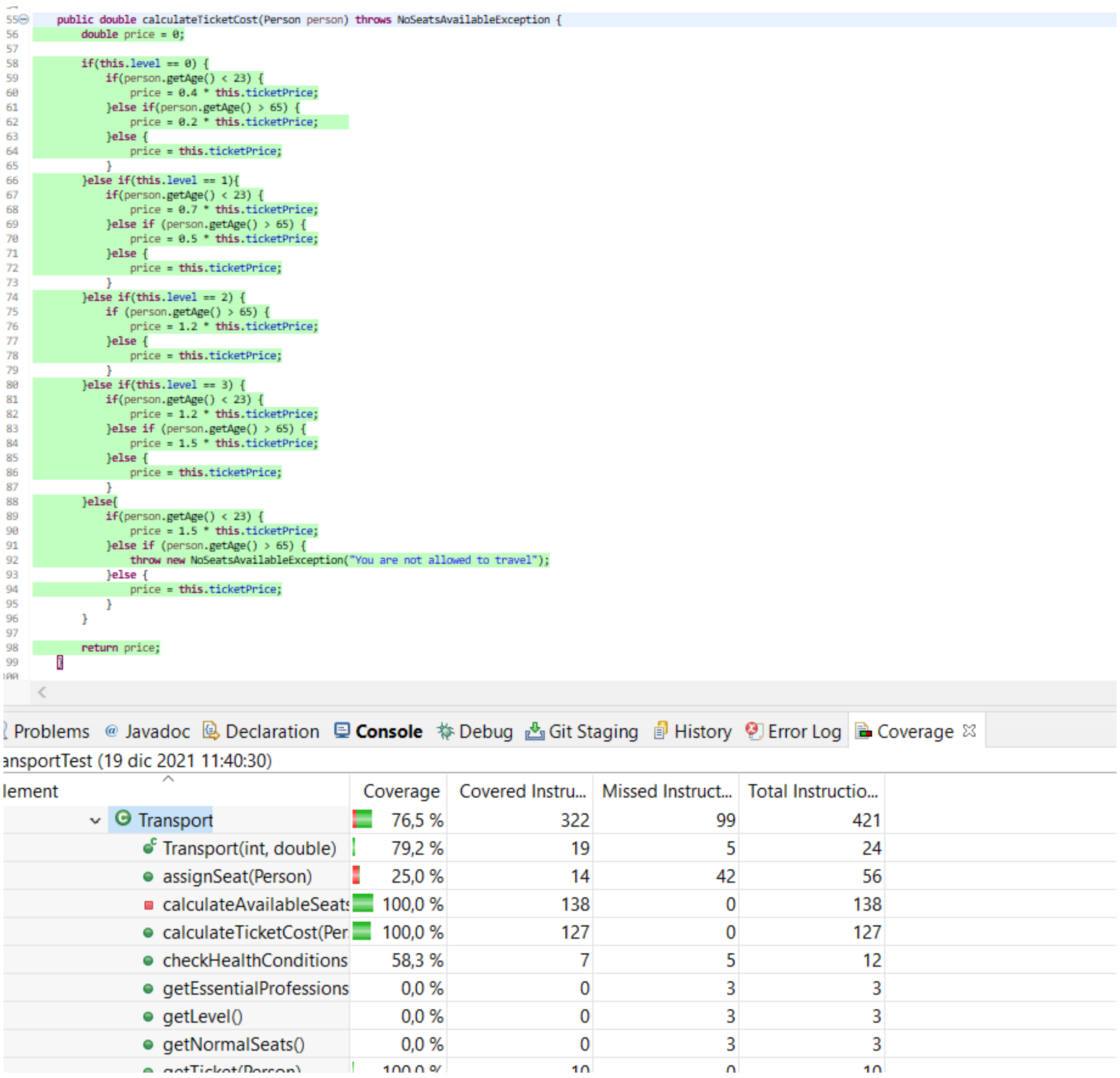
```java
    public double calculateTicketCost(Person person) throws NoSeatsAvailableException {
        double price = 0;

        if(this.level == 0) {
            if(person.getAge() < 23) {
                price = 0.4 * this.ticketPrice;
            }else if(person.getAge() > 65) {
                price = 0.2 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else if(this.level == 1){
            if(person.getAge() < 23) {
                price = 0.7 * this.ticketPrice;
            }else if (person.getAge() > 65) {
                price = 0.5 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else if(this.level == 2) {
            if (person.getAge() > 65) {
                price = 1.2 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else if(this.level == 3) {
            if(person.getAge() < 23) {
                price = 1.2 * this.ticketPrice;
            }else if (person.getAge() > 65) {
                price = 1.5 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else{
            if(person.getAge() < 23) {
                price = 1.5 * this.ticketPrice;
            }else if (person.getAge() > 65) {
                throw new NoSeatsAvailableException("You are not allowed to travel");
            }else {
                price = this.ticketPrice;
            }
        }

        return price;
    }
```

ansportTestPairwise (22 dic 2021 11:27:19)

lement	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
Transport	76,8 %	318	96	414
Transport(int, double)	79,2 %	19	5	24
assignSeat(Person)	64,3 %	36	20	56
calculateAvailableSeats	100,0 %	138	0	138
calculateTicketCost(Per	74,0 %	94	33	127

For this set of Pairwise test cases we got a coverage of 74.0%. As we have not implemented all of the test cases for achieving pairwise (just a set, specifically 10), this percentage is not as high. However, it should reach 100% if all of them were implemented because in pairwise we have all the possible combinations.

DECISION COVERAGE



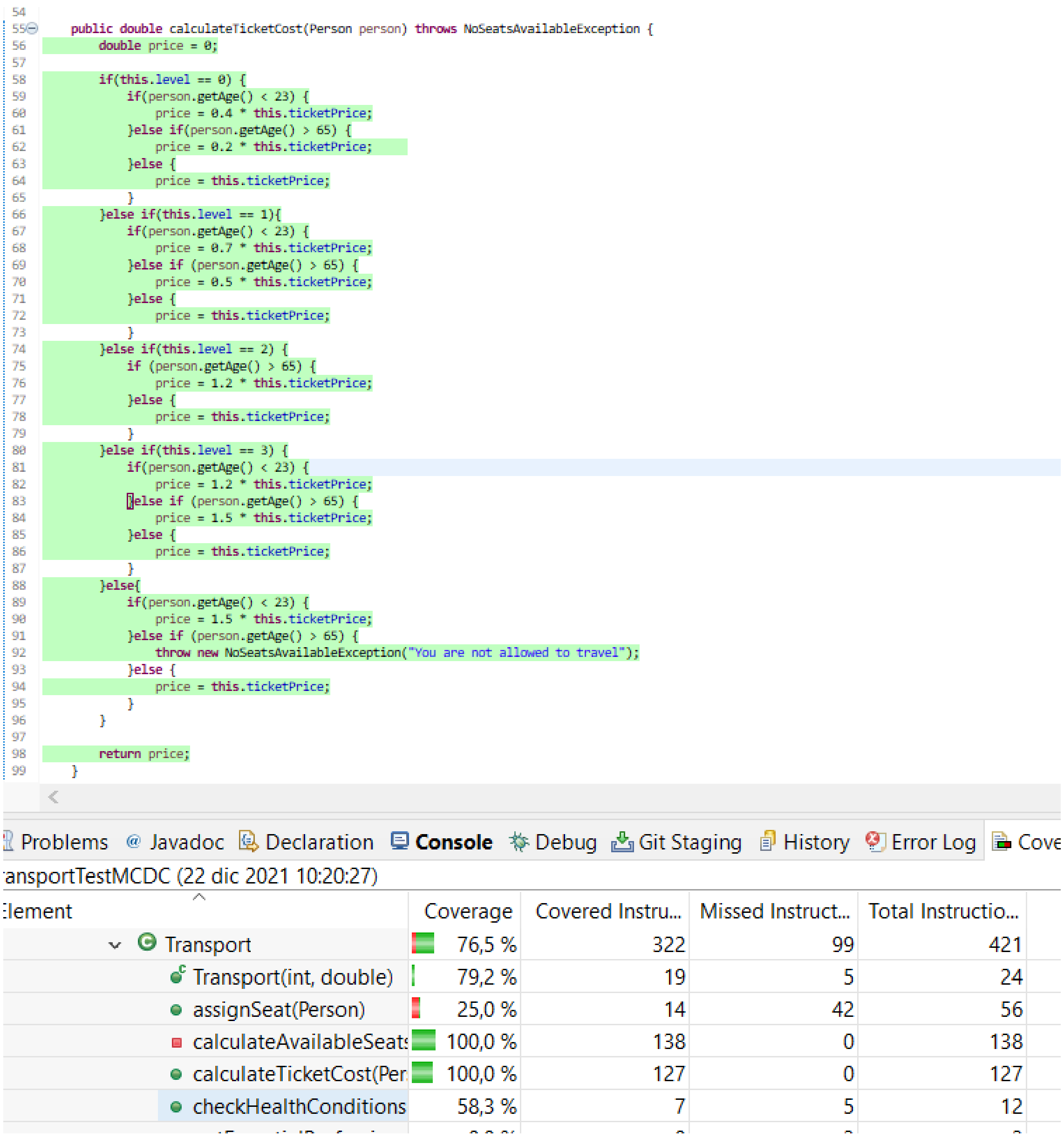
```
    public double calculateTicketCost(Person person) throws NoSeatsAvailableException {
        double price = 0;

        if(this.level == 0) {
            if(person.getAge() < 23) {
                price = 0.4 * this.ticketPrice;
            }else if(person.getAge() > 65) {
                price = 0.2 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else if(this.level == 1){
            if(person.getAge() < 23) {
                price = 0.7 * this.ticketPrice;
            }else if (person.getAge() > 65) {
                price = 0.5 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else if(this.level == 2) {
            if (person.getAge() > 65) {
                price = 1.2 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else if(this.level == 3) {
            if(person.getAge() < 23) {
                price = 1.2 * this.ticketPrice;
            }else if (person.getAge() > 65) {
                price = 1.5 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else{
            if(person.getAge() < 23) {
                price = 1.5 * this.ticketPrice;
            }else if (person.getAge() > 65) {
                throw new NoSeatsAvailableException("You are not allowed to travel");
            }else {
                price = this.ticketPrice;
            }
        }

        return price;
    }
```

ansportTest (19 dic 2021 11:40:30)

lement	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
Transport	76,5 %	322	99	421
Transport(int, double)	79,2 %	19	5	24
assignSeat(Person)	25,0 %	14	42	56
calculateAvailableSeats	100,0 %	138	0	138
calculateTicketCost(Per	100,0 %	127	0	127
checkHealthConditions	58,3 %	7	5	12
getEssentialProfessions	0,0 %	0	3	3
getLevel()	0,0 %	0	3	3
getNormalSeats()	0,0 %	0	3	3
getTicket(Person)	100,0 %	10	0	10

We have done all the test cases related to it. So, all decisions must be covered. For this reason, the coverage of the method ***"calculateTicketCost"*** is 100%.

MC/DC



```java
    public double calculateTicketCost(Person person) throws NoSeatsAvailableException {
        double price = 0;

        if(this.level == 0) {
            if(person.getAge() < 23) {
                price = 0.4 * this.ticketPrice;
            }else if(person.getAge() > 65) {
                price = 0.2 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else if(this.level == 1){
            if(person.getAge() < 23) {
                price = 0.7 * this.ticketPrice;
            }else if (person.getAge() > 65) {
                price = 0.5 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else if(this.level == 2) {
            if (person.getAge() > 65) {
                price = 1.2 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else if(this.level == 3) {
            if(person.getAge() < 23) {
                price = 1.2 * this.ticketPrice;
            }else if (person.getAge() > 65) {
                price = 1.5 * this.ticketPrice;
            }else {
                price = this.ticketPrice;
            }
        }else{
            if(person.getAge() < 23) {
                price = 1.5 * this.ticketPrice;
            }else if (person.getAge() > 65) {
                throw new NoSeatsAvailableException("You are not allowed to travel");
            }else {
                price = this.ticketPrice;
            }
        }

        return price;
    }
```

ansportTestMCDC (22 dic 2021 10:20:27)

lement	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
Transport	🟩 76,5 %	322	99	421
Transport(int, double)	79,2 %	19	5	24
assignSeat(Person)	🟥 25,0 %	14	42	56
calculateAvailableSeat	🟩 100,0 %	138	0	138
calculateTicketCost(Per	🟩 100,0 %	127	0	127
checkHealthConditions	58,3 %	7	5	12

We have done all the test cases related to it. The coverage of the method **"calculateTicketCost**" is 100%. The difference with decision coverage is that a higher number of test cases is needed to reach it. This is because in decision coverage we just focus on the decisions.