Universidad Carlos III

Software Development 2023-24

Assignment 2

Course 2023-24

Guided exercise 2

GROUP:**88** TEAM: **T05**

Members:

Izan Sánchez Álvaro, 100495774

Diego San Román Posada, 100495878

Bárbara Sánchez Moratalla, 100495857

TABLE OF CONTENTS

0. INTRODUCTION

This document serves as an exhaustive explanation of the tests employed to check the functionalities of the GE2 functions.
The document is divided in sections, depending on which functions it makes reference, and each section will have different subcategories since tests are structured to cover different aspects of the system, including equivalence classes and boundaries, grammars and graphs, and control flow analysis.
All of them are aimed to give a better understanding on why we have chosen those tests and explain the approach of the methods conducted in each of the three functions needed to be implemented.

1. First function: roomReservation

The first function is the one used to perform the hotel reservations. This function has 7 inputs, including:

- credit_card: a 16 digit integer that is correct according to the Luhn's algorithm
- name_surname: a string made up of at least two stringst whose length is between 10 characters and 50 characters
- id_card: a 3 digit integer
- phone_number: a 9 digit integer
- room_type: a string chosen from the following options: "single", "double" or "premium"
- arrival_date: a string of the format "YYYY-MM-DD" where YYYY is the year (a number between the current year and 9999), MM the month (a number between 01 and 12) and DD the day (a number between 01 and 31)
- num_days: an integer between 1 and 10

To check all the inputs we need to implement analysis of *equivalence classes* (EC) and *boundary values* (BV). For this, we analyse the equivalence classes and boundary values for each one of the 7 inputs. Then, we do the same for the output of the function, a valid 32 character string representing a localizer.

Glossary:
ECV: Equivalence Class Valid
ECNV: Equivalence Class Not Valid
BVV: Boundary Value Valid
BVNV: Boundary Value Not Valid

1.1 Equivalence classes and Boundary Values

1.1.1 credit_card

Rule	Valid Class	Invalid Classes
Datatype	ECV1. Integer	ECNV1. Datatype different than integer
Size: 16 digits integer	ECV2. 16 digits integer BVV1. 16	ECNV2. More than 16 digits integer ECNV3. Less than 16 digits integer BVNV1. 17 digits integer BVNV2. 15 digits integer
Content: valid credit card number according to Luhn's algorithm	ECV3. Valid credit card number according to the Luhn's algorithm	ECNV4. Invalid credit card number according to the Luhn's algorithm

1.1.2 name_surname

Rule	Valid Class	Invalid Classes
Datatype	ECV4. String	ECNV5. Dataype different than string
10 <= len(name_surname) <= 50	ECV5. 10 <= name_surname <= 50 BVV2. 10 BVV3. 11 BVV4. 50 BVV5. 49	ECNV6. name_surname < 10 (9) ECNV7. name_surname > 50 (51) BVNV3. 9 BVNV4. 51
Content: at least 2 strings separated by a space	EVC6. At least 2 strings separated by a space BVV6. 2 strings separated by a space BVV7. 3 strings separated by a space	ECNV8. Strings separated by a separator different than a space BVNV5. 1 string

1.1.3 id_card

Rule	Valid Class	Invalid Classes
Datatype	ECV7. Integer	ECNV9. Datatype different than integer
Size: 3 digits	ECV8. 3 digits integer BVV8. 3	ECNV10. More than 3 digits integer ECNV11. Less than 3 digits integer BVNV6. 4 digits integer BVNV7. 2 digits integer

1.1.4 phone_number

Rule	Valid Class	Invalid Classes
Datatype	ECV9. Integer	ECNV12. Datatype different than integer
Size: 9 digits	ECV10. 9 digits integer BVV9. 9	ECNV13. More than 9 digits integer ECNV14. Less than 9 digits integer BVNV8. 10 digits integer BVNV9. 8 digits integer

1.1.5 room_type

Rule	Valid Class	Invalid Classes
Datatype	ECV11. String	ECNV15. Datatype different than string
Allowed values	ECV12. "single" ECV13. "double" ECV14. "suite"	ECNV16. Any other value i.e. “OTHER”.

1.1.6 arrival_date

Rule	Valid Class	Invalid Classes
Datatype	ECV15. String	ECNV17. Datatype diferent than string
Size: 10 characters	ECV16. 10 character string BVV10. 10	ECNV18. More than 10 character string ECNV19. Less than 10 character string BVNV10. 11digits integer BVNV11. 9 digits integer
Content: String of the format "DD/MM/YYYY"	ECV17. "DD/MM/YYYY"	ECNV20. String with 10 characters but with an incorrect format
Size of DD (day): 01 <= DD <= 31	ECV18. 01 <= DD <= 31 BVV11. 01 BVV12. 02 BVV13. 31 BVV14. 30	ECNV21. DD < 01 (00) ECNV22. DD > 31 (32) BVNV12. 00 BVNV13. 32
Size of MM (month): 01 <= MM <= 12	ECV19. 01 <= MM <= 12 BVV15. 01 BVV16. 02 BVV17. 12 BVV18. 11	ECNV23. MM < 01 (00) ECNV24. MM > 12 (13) BVNV14. 00 BVNV15. 13
Size of YYYY(year): current year (2024) <= YYYY <= 9999	ECV20. current year (2024) <= YYYY <= 9999 BVV19. current year (2024) BVV20. current year + 1 (2025) BVV21. 9999 BVV22. 9998	ECNV25. YYYY < current year (2023) ECNV26. YYYY > 9999 (10000) BVNV16. 2023 BVNV17. 10000

1.1.7 num_days

Rule	Valid Class	Invalid Classes
Datatype	ECV21. Integer	ECNV27. Datatype different than integer
1<= num_days <= 10	ECV22. 1 <= num_days <= 10 BVV23. 1 BVV24. 2 BVV25. 10 BVV26. 9	ECNV28. num_days < 1 (0) ECNV29. num_days > 10 (11) BVNV18. 0 BVNV19. 11

1.1.8 Outputs

Rule	Valid Class	Invalid Classes
Outputs	ECV23. Valid localizer	ECNV30. Exception: Invalid credit card format ECNV31. Exception: Invalid credit card number ECNV32. Exception: Invalid name and surname length ECNV33. Exception: Invalid name and surname format ECNV34. Exception: Invalid ID card ECNV35. Exception: Invalid phone number ECNV36. Exception: Invalid room type ECNV37. Exception: Invalid arrival date format ECNV38. Exception: Invalid day in arrival date ECNV39. Exception: Invalid month in arrival date ECNV40. Exception: Invalid year in arrival date ECNV41. Exception: Invalid number of days ECNV42. Exception: name_surname already has a reservation

1.2 Tests Explanation

In total, there are 32 tests for this first function. These tests are divided based on whether they're valid or invalid. Then, for the invalid tests, they're divided depending on why they failed, like if the person already had a reservation or if there's an incorrect input.

Valid Test Case (TC1Valid)

- Purpose: This test case verifies a valid scenario where all information is correct, and the person does not have an existing reservation.
- Input Explanation: The input parameters include valid credit card information, name and surname, ID card, phone number, room type, arrival date, and number of days.
- Expected Outcome: The function should return a valid localizer string

Test Case involving the client already having a reservation (TC2)

- Purpose: This test case verifies a valid scenario where all information is correct, but the person already has an existing reservation.
- Input Explanation: The input parameters include all the valid information but a name_surname that already appears in the Reservations.json.
- Expected Outcome: A specific exception indicating the person already has a reservation.

Test Cases Involving credit_card (TC3-TC6)

- Purpose: This test cases verifiy an invalid scenario where the credit_card is not correct.
- Input Explanation: The input parameters include a credit card number that is not correct for several reasons.
- Expected Outcome: A specific exception indicating an invalid credit card number for several reasons.

The different reason for the incorrect credit card number are:

- Incorrect datatype (TC3)
- Incorrect length (TC4 and TC5)
- Invalid number according to Luhn’s algorithm (TC6)

Test Cases Involving name_surname (TC7-TC11)

- Purpose: This test cases verifiy an invalid scenario where the name_surname is not correct.
- Input Explanation: The input parameters include a name and surname that is not correct for several reasons.
- Expected Outcome: A specific exception indicating an invalid name and surname for several reasons.

The different reason for the incorrect name and surname are:

- Incorrect datatype (TC7)
- Incorrect separator between name and surname(s) (TC8)
- Incorrect number of elements in the string (TC9)
- Incorrect length (TC10 and TC11)

Test Cases Involving id_card (TC12-TC14)

- Purpose: This test cases verifiy an invalid scenario where the id_card is not correct.
- Input Explanation: The input parameters include a id card that is not correct for several reasons.
- Expected Outcome: A specific exception indicating an invalid id card for several reasons.

The different reason for the incorrect name and surname are:

- Incorrect datatype (TC12)
- Incorrect number of digits (TC13 and TC14)

Test Cases Involving phone_number (TC15-TC17)

- Purpose: This test cases verifiy an invalid scenario where the phone_number is not correct.
- Input Explanation: The input parameters include a phone number that is not correct for several reasons.
- Expected Outcome: A specific exception indicating an invalid phone number for several reasons.

The different reason for the incorrect name and surname are:

- Incorrect datatype (TC15)
- Incorrect number of digits (TC16 and TC17)

Test Cases Involving room_type (TC18-TC19)

- Purpose: This test cases verifiy an invalid scenario where the room_type is not correct.
- Input Explanation: The input parameters include a room type that is not correct for several reasons.
- Expected Outcome: A specific exception indicating an invalid room type.

The different reason for the incorrect name and surname are:

- Incorrect datatype (TC18)
- Room type not in the accepted values (TC19)

Test Cases Involving arrival_date (TC20-TC29)

- Purpose: This test cases verifiy an invalid scenario where the arrival_date is not correct.
- Input Explanation: The input parameters include an arrival date that is not correct for several reasons.
- Expected Outcome: A specific exception indicating an invalid arrival date for several reasons.

The different reason for the incorrect name and surname are:

- Incorrect datatype (TC20)
- Incorrect string length (TC21 and TC22)
- Incorrect format (TC23)
- Day number not in the range of accepted values (TC24-TC25)
- Month number not in the range of accepted values (TC26-TC27)
- Year number not in the range of accepted values (TC28-TC29)

Test Cases Involving num_days (TC30-TC32)

- Purpose: This test cases verifiy an invalid scenario where the num_days is not correct.
- Input Explanation: The input parameters include a number of days that is not correct for several reasons.
- Expected Outcome: A specific exception indicating an invalid number of days for several reasons.

The different reason for the incorrect name and surname are:

- Incorrect datatype (TC30)
- Incorrect number not in range of accepted values (TC31 and TC32)

2. Second function: guestArrival

The inputs of the second function are stored inside the json file named as Arrival.json, and they need to have the following format:

{ "Localizer": "String having 32 hexadecimal characters",
"IdCard": “valid idCard” }

To check that the input has the correct format we need to implement a *syntax analysis* over the input itself. For that, first we create a grammar to understand how the format works, then we create a derivation tree over this grammar and create the unit tests over that derivation tree.

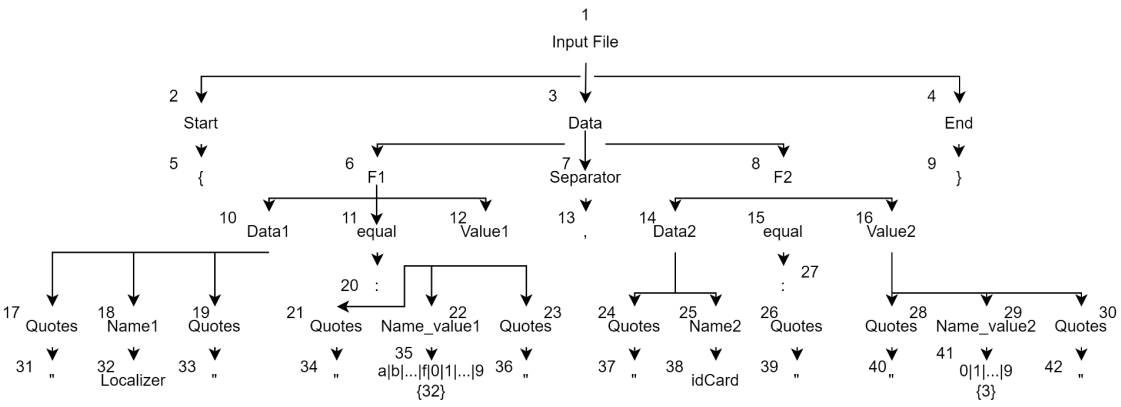
2.1 Grammar for inputs

The grammar needed to represent the valid format for the input is as follows:

{
File ::= StartDataEnd
Start ::= {
End ::= }
Data ::= F1SeparatorF2
Separator ::= ,
F1 ::= Data1equalValue1
equal ::= :
F2 ::= Data2equalValue2
Data1 ::= QuotesName1Quotes
Name1 ::= Localizer
Quotes ::= “
Value1 ::= QuotesName_value1Quotes
Name_value1 ::= a|b|c…|f|0|1…|9 {32 digits}
Data2 ::= Quotes Name2 Quotes
Name2 ::= IdCard
Value2 ::= QuotesName_value2Quotes
Name_value2 ::= 0|1|...|9 {3 digits}
}

The general format is:
StartDataEnd -> { "Localizer": "String having 32 hexadecimal characters",
"IdCard": “valid idCard” }

2.2 Derivation Tree



In this derivation tree we can see all the terminal and non terminal nodes of the grammar just shown above. This derivation tree will help a lot when creating the valid and non valid testes in python since they will be related with nodes of the derivation tree (one or many).

- Terminal nodes: 5, 9, 13, 20, 27, 31, 33, 34, 36, 37, 39, 40, 42
- Non-terminal nodes: The rest of nodes

2.3 Tests explanation

The tests described for this function are created with the aim of checking whether the input itself has the correct format and how the function acts when detecting this error. In total there are 22 tests, 1 valid and 21 invalid tests.

Valid test-case (TC1Valid):

In this check we check whether the function returns the correct roomKey when the input format is well defined, and doesn't return any exception.
The input used is the following:

```
{
   "Localizer": "388170e32dc0ba9b864085b38e0e6442",
    "IdCard": 100
}
```

The roomKey expected and actual is the following:

```
'1c09ee0a41504a55c8b43e1de211f2231edf227742efd70bab0a4bca99399a2a'
```

Tests returning a roomKey:

There are also other tests that, although the format is not correct syntactically, the function will continue returning a roomKey.

- TC6 where there is not a syntax error itself since the data is repeated twice, so the function will continue working.

```
{
  "Localizer": "388170e32dc0ba9b864085b38e0e6442",
  "IdCard": 100,
  "Localizer": "388170e32dc0ba9b864085b38e0e6442",
  "IdCard": 100
}
```

- TC8 where one of the fields is duplicated, it will continue returning the correct roomKey.

```
{
  "Localizer": "388170e32dc0ba9b864085b38e0e6442",
  "Localizer": "388170e32dc0ba9b864085b38e0e6442",
  "IdCard": 100
}
```

Tests that are incorrect syntactically:

The rest of the tests are going to check how the function works over exceptions. All of these tests have the objective of specifying what are the incorrect formats or syntax of the input itself and how these errors should be treated.
We have divided this subsection in types of exceptions returned:

- hotelManagementException("File not found or empty file")

  The vast majority of non-valid tests will have this expected exception. There are cases where the input is empty, so the function cannot continue working and raises this error, or the same happens when the input has an incorrect format as: deleting, modifying or duplicating the separators, the equal signs or the start/end symbols. Some examples are:

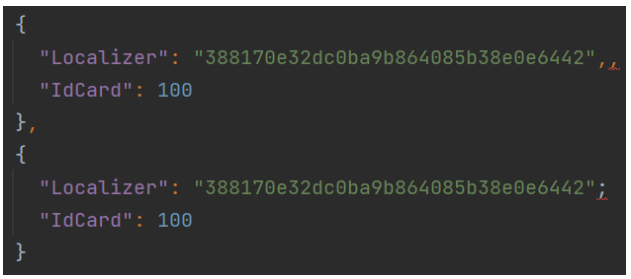
  Duplicated Input file (without a separator)

```
{
   "Localizer": "33baf27f2b20da3de3b24f51d9d25926",
    "IdCard": 236
}
{
   "Localizer": "33baf27f2b20da3de3b24f51d9d25926",
    "IdCard": 236
}
```

  Modifying the start or end symbol (changing "{" for "(")

```
(
   "Localizer":, "33baf27f2b20da3de3b24f51d9d25926",
    "IdCard": 236
},
{
   "Localizer": "33baf27f2b20da3de3b24f51d9d25926",
    "IdCard": 236
)
```

  Modifying or duplicating the separator symbol



Duplicating or deleting Data keys:

- hotelManagementException(“‘Localizer’ or ‘Id’ key missing in JSON”)

  All examples where the localizer or id key are missing (but with quotes “ “: value), duplicated or modified:

  Operations over Localizer

  Operations over Id

```
  {
     "Localizer": "388170e32dc0ba9b864085b38e0e6442",
      "": 100
  },
{
  "Localizer": "388170e32dc0ba9b864085b38e0e6442",
      "IdCardIdCard": 100
},
{
  "Localizer": "388170e32dc0ba9b864085b38e0e6442",
      "IcCerd": 100
}
```


- hotelManagementException("Localizer is not well defined" / "New Idcard is not in reservation")

  Test cases where any operation is performed over the value of the localizer or the id. The function will try to search for that value in the reservations json and will not find it, showing the error message:

  Localizer value


```
  {
     "Localizer": "",
      "IdCard": 236
  },
{
  "Localizer": "33baf27f2b20da3de3b24f51d9d2592633baf27f2b20da3de3b24f51d9d25926",
      "IdCard": 236
},
{
  "Localizer": "33baf27f2b20da3de3b24f51d9d25926",
      "IdCard": 236
}
```


  IdCard value


```
  {
     "Localizer": "388170e32dc0ba9b864085b38e0e6442",
      "IdCard": 0
  },
{
  "Localizer": "388170e32dc0ba9b864085b38e0e6442",
      "IdCard": 236236
},
{
  "Localizer": "388170e32dc0ba9b864085b38e0e6442",
      "IdCard": 239
}
```

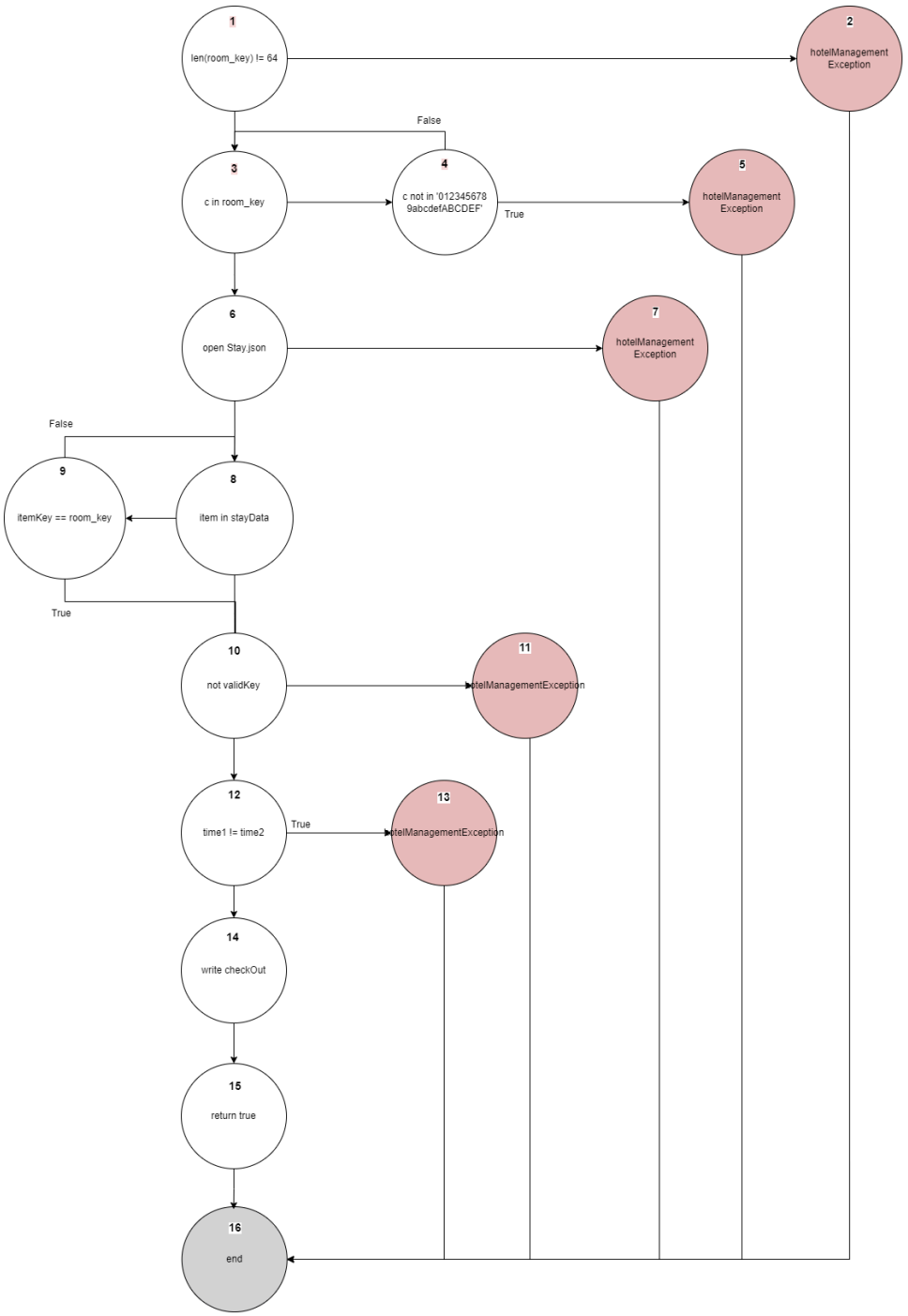
3. Third function: guestCheckout

This last function is responsible for executing the checkout. To do so, it only asks for 1 input, which is the room key in SHA256 hexadecimal string form. For it to be valid, it must satisfy the following conditions:

- The length of the string must be equal to 64.
- The string must be in hexadecimal. That is, it must be composed of any numbers (0-9) and only certain letters (A-F).

3.1. Control Flow Graph

For the analysis of this last function we implemented Structural Testing techniques, which are based on the different paths the function can take when executing itself. To identify all these paths, we created nodes representing different pieces of code in our function. The resulting Control Flow Graph is the following:



According to McCabe complexity, our graph has a total of 23E - 16N + 2 = 9 independent paths that ensure that all statements are executed at least once.However, the total number of paths we can take are 18. Let's go through each of them and explain them individually.

3.2. Paths Explanation

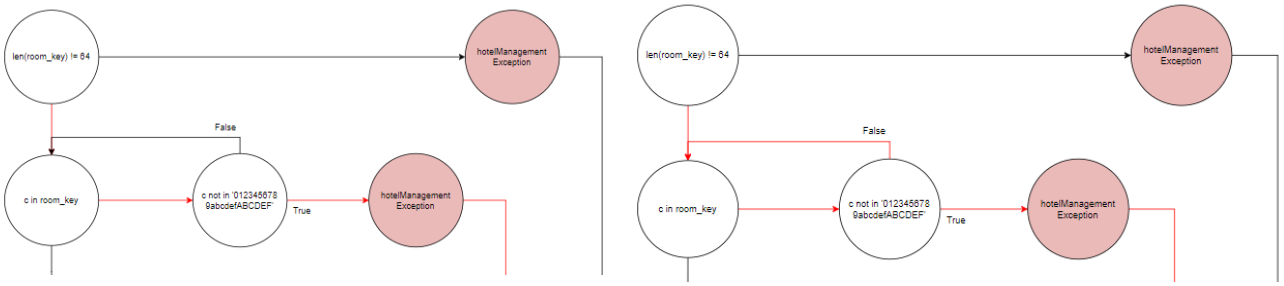
Due to the approach of our function, we must note that some paths cannot be taken at all (specifically those involving how many times we enter the loop, since we will get an hotelManagementException earlier in the code). For that reason, we splitted the test cases into 2 groups: valid test cases and extra test cases.

Valid Test Cases (TC1 - TC8)

ID	Expected results	Path	Path definition
TC1	hotelManagementException: "Not processable room code."	1, 2, 16	len(room_key) != 64
TC2	hotelManagementException: "Not processable room code."	1, 3, 4, 5, 16	room_key is not in hexadecimal (loop 1 is only entered once)
TC3	hotelManagementException: "Not processable room code."	1, 3, 4, 5, 16	room_key is not in hexadecimal (loop 1 is entered more than once)
TC4	hotelManagementException: "Stay.json not found or it is empty."	1, 3, 4, 6, 7, 16	Stay.json not found or it is empty.
TC5	hotelManagementException: "Such key is not registered."	1, 3, 4, 6, 8, 9, 10, 11, 16	The key is valid (sha256 type string) but it doesn't match any reservation.
TC6	hotelManagementException: "Departure date is not valid."	1, 3, 4, 6, 8, 9, 10, 12, 13, 16	The departure date established does not match the actual date.
TC7	*Return True*	1, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16	Everything is correct (loop 2 is only entered once).
TC8	*Return True*	1, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16	Everything is correct (loop 2 is entered more than once).

TC1: The length of the string is not 64. We get a hotelManagementException (2) instantaneously, indicating that the room key is not processable and the function ends (16).

TC2, TC3: The string contains characters that are not in hexadecimal. We manage to enter the first loop, but we get to a hotelManagementException (5) eventually. Afterwards the function ends (16). Note that we might enter the loop just once (TC2) or more than once (TC3).

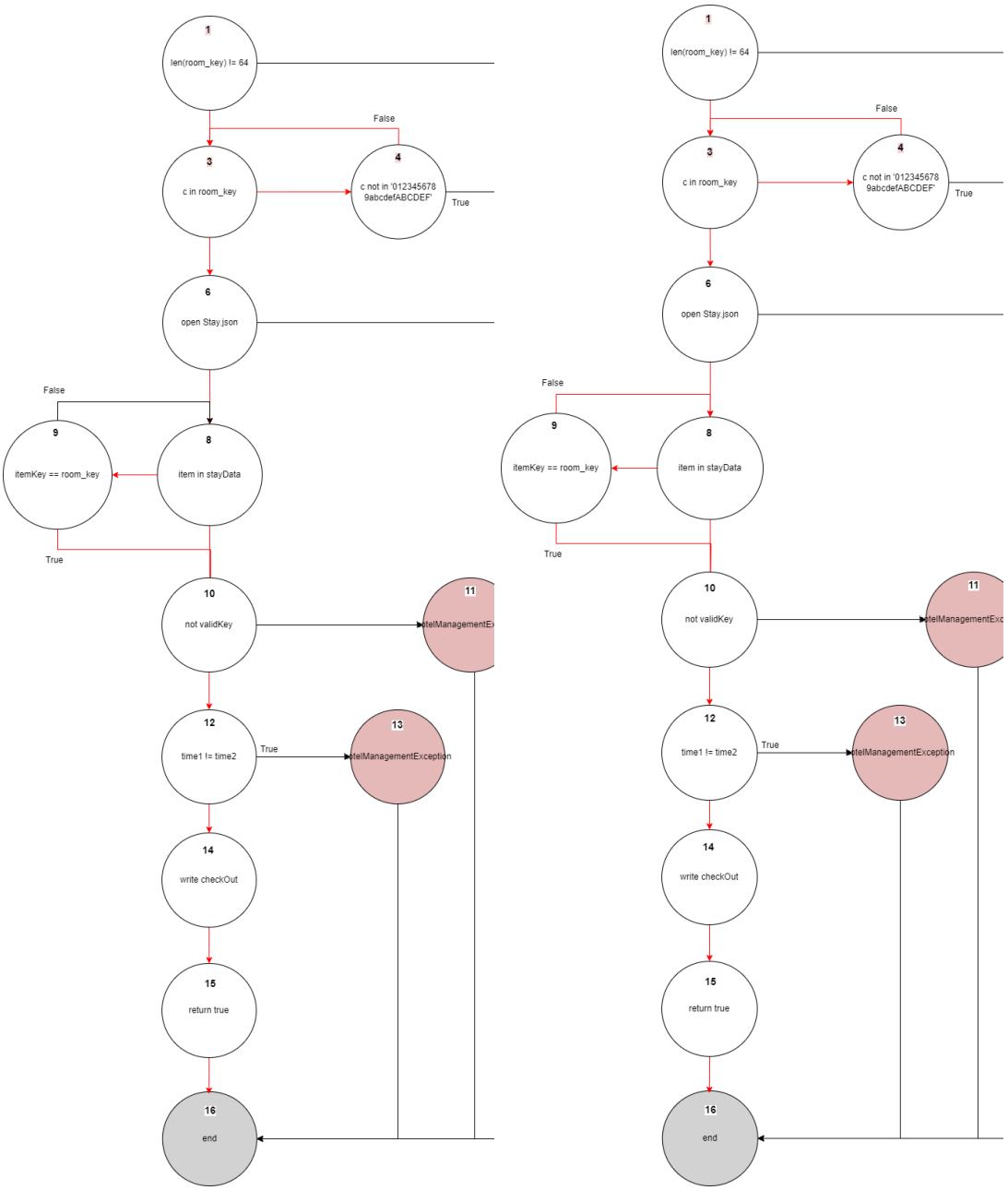


TC4: We encounter an error when opening Stay.json (6), because it is not found or empty. We get another hotelManagementException (7) and the function ends (16).

TC5: The key provided as input is valid, but we cannot find any key from our clients in Stay.json that matches that one. Since the validKey boolean is still False (10), we get the corresponding hotelManagementException (11) and the program ends (16).

TC6: The departure date written in the client's data does not match the actual date (12). We get an hotelManagementException (12) and the function ends (16).

TC7, TC8: Everything is correct (the key is valid, we open Stay.json successfully, the key matches a reservation and the departure date is correct). We write into checkOut.json the key and the departure date (14). Finally, we return True (15) and the program ends (16). Again, note that we might enter the second loop just once (TC7, if the input key matches the first client's key) or more than once (TC8).



Extra Test Cases (TC9 - TC18)

Extra IDs	Expected results (none)	Path	Path definition
TC9	-- can't be tested (the key must be 64-character-long to go through the code)	1, 3, 6, 7, 16	We don't go through loop 1 (for c in room_key) + Stay.json not found or empty
TC10	-- can't be tested (the key must be 64-character-long to go through the code)	1, 3, 6, 8, 9, 10, 11, 16	We don't go through loop 1 (for c in room_key) + key not found
TC11	-- can't be tested (the key must be 64-character-long to go through the code)	1, 3, 6, 8, 9, 10, 12, 13, 16	We don't go through loop 1 (for c in room_key) + wrong departure date
TC12	-- can't be tested (the key must be 64-character-long to go through the code)	1, 3, 6, 8, 9, 10, 12, 14, 15, 16	We don't go through loop 1 (for c in room_key) + correct
TC13	-- can't be tested (both exceptions together)	1, 3, 6, 8, 10, 11, 16	We don't go through either loop 1 or 2 + key not found
TC14	-- can't be tested (both exceptions together)	1, 3, 6, 8, 10, 12, 13, 16	We don't go through either loop 1 or 2 + wrong departure date
TC15	-- can't be tested (both exceptions together)	1, 3, 6, 8, 10, 12, 14, 15, 16	We don't go through either loop 1 or 2 + correct
TC16	-- can't be tested (reservations.json can't be empty)	1, 3, 4, 6, 8, 10, 11, 16	We don't go through loop 2 (for item in Reservations.json) + key not found
TC17	-- can't be tested (reservations.json can't be empty)	1, 3, 4, 6, 8, 10, 12, 13, 16	We don't go through loop 2 (for item in Reservations.json) + wrong departure date
TC18	-- can't be tested (reservations.json can't be empty)	1, 3, 4, 6, 8, 10, 12, 14, 15, 16	We don't go through loop 2 (for item in Reservations.json) + correct

TC9 - TC12: These test cases imply that we don't go through loop 1 (that is, that there are no characters in the string input). However, since we had established just before (node 1) the condition that the input string must be 64 characters long, it is impossible for the program not to enter the loop (we either enter it or we get an hotelManagementException from node 2).

TC16 - TC18: These test cases don't go through loop 2 (which means that they don't explore any item in Stay.json and, therefore, that Stay.json is empty). Since we had made sure just before that Stay.json can not be empty (node 10), it is impossible for these tests to be implemented (we will get a hotelManagementException from node 11 instead).

TC13 - TC15: These test cases not only do they not go through loop 1, but they also don't go through loop 2. Because of the reasons explained above, it is impossible for these tests to be implemented.