Universidad Carlos III de Madrid

SOFTWARE DEVELOPMENT

Bachelor's Degree in Computer Science & Engineering

GE2

Academic Year 2023/2024

Salvador Ayala Iglesias | 100495832 | 100495832@alumnos.uc3m.es | g89
Inés Fuai Guillén Peña | 100495752 | 100495752@alumnos.uc3m.es | g89
Kritika Pantha | 100531231 | 100531231@alumnos.uc3m.es | g89

TABLE OF CONTENTS

Pylint

Changes made in pylintrc:

1. argument-naming-style=snake_case
2. attr-naming-style=snake_case
3. class-attribute-naming-style=snake_case
4. class-naming-style=PascalCase
5. function-naming-style=snake_case
6. method-naming-style=camelCase
7. module-naming-style=PascalCase
8. variable-naming-style=snake_case
9. max-args=8
10. max-attributes=8
11. max-locals=20
12. max-public-methods=40
13. max-line-length=120
14. max-branches=15

Warnings raised by pylint considered exceptions:

- Method names: "<__methodName> doesn't conform to camelCase naming style". Found in "HotelStay" (private methods start with "__" and all tests (since test methods have to start with "test_"
- Module names: "<ModuleName> doesn't conform to PascalCase naming style". Found in "main" and all test files, since they have to be named like test_ModuleName_tests.
- Import errors: "Unable to import <ModuleName>". Found in "main" and all tests. All imports work properly, pylint raises an error anyway.

Function 1

Equivalence classes and boundary values:

1. **idcard:**
    a. Equivalence Classes: Valid ID cards, invalid ID cards.
    b. Boundary Values: Minimum and maximum length of a valid ID card.
2. **creditcard:**
    a. Equivalence Classes: Valid credit card numbers, invalid credit card numbers.
    b. Boundary Values: Minimum and maximum length of a valid credit card number (16), it also has to follow the Luhn's algorithm.
3. **name:**
    a. Equivalence Classes: Valid name and surname, invalid name and surname.
    b. Boundary Values: Minimum and maximum length of the name and surname, special characters, etc.
4. **phonenum (phone number):**
    a. Equivalence Classes: Valid phone numbers, invalid phone numbers.
    b. Boundary Values: Length of a valid phone number (9) .
5. **arrival:**
    a. Equivalence Classes: Valid arrival dates, invalid arrival dates.
    b. Boundary Values: Arrival date with different format (d/m/Y).
6. **rtype (room type):**
    a. Equivalence Classes: Single, Double, Suite.
    b. Boundary Values: No specific boundary values.
7. **days:**
    a. Equivalence Classes: Valid number of days, invalid number of days.
    b. Boundary Values: Minimum and maximum valid number of days (1 and 10), number of days less than 1 and greater than 10.

Description of the test cases:

For this function, we have carried out a total of 17 test cases:

1. **Valid input:** Provide valid inputs for all parameters within their respective ranges.
2. **Invalid phone number:** Test with an invalid phone number (wrong length), we must verify that the function returns the expected exception.
3. **Invalid room type:** Test with an invalid room type value, we must verify that the function returns the expected exception.
4. **Invalid number of days:** Test with an invalid number of days (i.e. 0 and 11), we must verify that the function returns the expected exception.

5. **Invalid name and surname:** Test with a name and surname too short, too long and with invalid characters, we must verify that the function returns the expected exception.
6. **Invalid arrival date:** Test with an invalid arrival date (incorrect format, dates not possible), we must verify that the function returns the expected exception.
7. **Invalid credit card:** Test with an invalid credit card number (incorrect length, doesn't follow Luhn's algorithm), we must verify that the function returns the expected exception.
8. **Invalid ID Card:** Test with an invalid ID card (incorrect length, format, letter), we must verify that the function returns the expected exception.
9. **Duplicate Reservation:** Test making a reservation with the same ID card as an existing reservation, we must verify that the function returns the expected exception.

Function 2

Same equivalence classes and boundary values as in function 1.

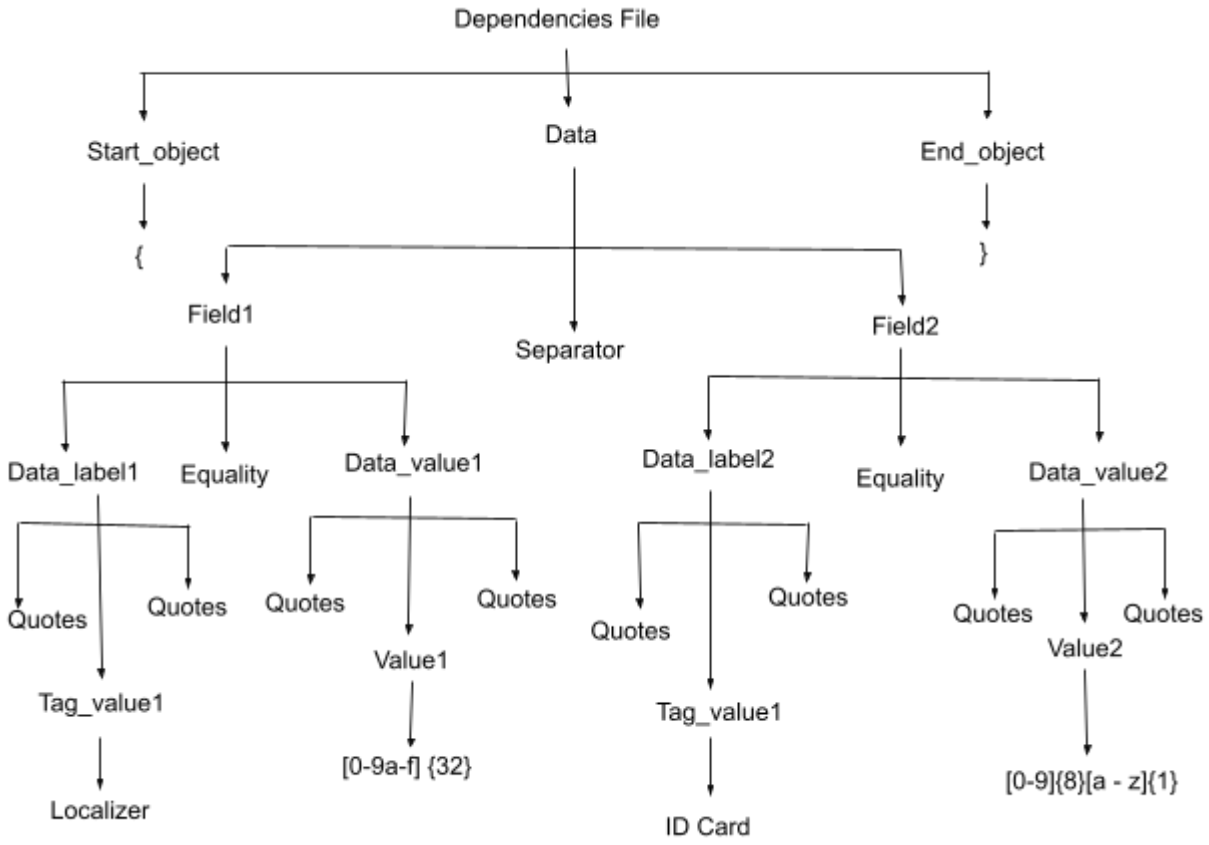
Grammar and the derivation tree:

Grammar Definition

File ::= Start_object Data End_object **Start_object** ::= { **End_object** ::= }
Data ::= Field1 Separator Field2 **Field1** ::= Data_label1 Equality Data_value1 **Field2** ::= Data_label2 Equality Data_value2 **Separator** ::= , **Equality** ::= :
Data ::= Field1 Separator Field2 **Field1** ::= Data_label1 Equality Data_value1 **Field2** ::= Data_label2 Equality Data_value2 **Separator** ::= , **Equality** ::= :
Data_label1 ::= Quotes Tag_value1 Quotes **Tag_value1** ::= Localizer **Data_value1** ::= Quotes Value1 Quotes **Value1** ::= [0-9a-f] {32} **Quotes** ::= "
Data_label2 ::= Quotes Tag_value2 Quotes **Tag_value2** ::= ID Card

Data_value2 ::= Quotes Value2 Quotes
Value2 ::= [0-9]{8}[a - z]{1}
Quotes ::= "

Derivation Tree



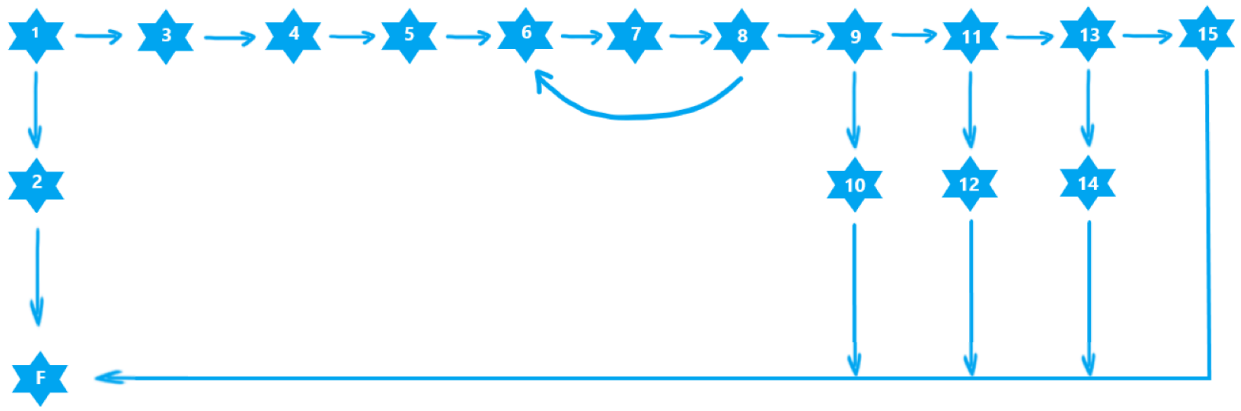
Description of the test cases:

1. **Valid input data:** Test the function with valid input data to ensure that it can successfully process a correct reservation request.
2. **Invalid Localizer:** Test the function with an invalid localizer (localizer not given) to verify that it raises the expected exception.
3. **Invalid ID card:** Test the function with an invalid ID card number (ID not given) to verify that it raises the expected exception.
4. **Invalid file (JSON file cannot be found):** Test the function when there is no input data (no file given) to verify that it raises the expected exception.
5. **Invalid file (keys are not found in the JSON file):** Test the function when the input data (keys) are given with a different name, we have to verify that it raises the expected exception.
6. **Invalid arrival date:** Test the function with an invalid arrival date (incorrect format, past dates) to ensure proper handling of invalid arrival dates.

Function 3

Control Flow Graph (CFG):

```
def guestCheckOut(self, room_key):
    """This method is used to check out a guest from the hotel. It receives a room key and returns True if the
    guest has checked out successfully. It raises an exception if the room key is not valid or if the departure
    date is not today."""
    # Validate the input data
1   if not isinstance(room_key, str) or len(room_key) != 64:
        raise HotelManagementException("Invalid room key format") 2

    # Search the check-ins (stays) store file for the room key
    found = False
    stays_file = self.stays_file

    # Open file
3   try:
        with open(stays_file, "r", encoding="utf-8") as file: 4
            stays = json.load(file) 5

            # Search for the guest in the stays file using the room key
        6 for stay in stays:
                if stay["room_key"] == room_key: 7
                    found = True
                8   departure_date = datetime.strptime(stay["departure"], "%d/%m/%Y")
                    break

    # If there is no stays file
9   except FileNotFoundError as e:
    10  raise HotelManagementException("Stays file does not exist") from e

    # If the key is not found, raise an exception
11  if not found:
    12  raise HotelManagementException("Invalid room key")

    # If the key exists, verify that the departure date is today
13  if departure_date.date() != datetime.utcnow().date():
    14  raise HotelManagementException("Departure date does not match today's date")

    # If the above is correct, save the data in the checkouts file
    checkout_data = {"departure_date": departure_date.strftime("%d/%m/%Y"), "room_key": room_key}
15  return self.saveCheckOutInFile(checkout_data)
```



Definition of the basic path

1. Start → Invalid room key → Raise Error
2. Start → Valid room key → Check departure date → Departure date matches today → Save data → Return True
3. Start → Valid room key → Check departure date → Departure date doesn't match today → Raise Error

Description of the test cases:

1. **Valid input data:** Test the function with valid input data to ensure that it can successfully process a correct reservation request.
    a. File doesn't exist
    b. File exists (empty)
    c. File exists (not empty)
2. **Invalid room key:** Test the function with an invalid room key (length lower than 64) to verify that it raises the expected exception.
3. **Invalid room key:** Test the function with an invalid room key (not string) to verify that it raises the expected exception.
4. **Invalid input:** Test the function with an invalid input (not given) to verify that it raises the expected exception.
5. **Invalid departure date:** Test the function with an invalid departure date to ensure proper handling of invalid departure dates.
6. **Invalid Stays file:** Test the function when there is an error due to the Stays file (it does not exist) to verify that it raises the expected exception.