6.2 Programming Exercise 3 & Unit Tests

TC4017.10 Software testing and quality assurance

Héctor Segura Quintanilla

A01154108

ITESM - MNA V

RESULTS OF MAIN.PY BASED ON REQUIREMENTS

Req 1. Classes were created so to satisfy consistency with the *two abstractions* segment of the requirement. The two abstractions are Hotel Management and Reservation handling. Although there might be redundancy given the specific elements required from each class, there is still clear responsibility delegation. An example that illustrates that the Hotel Management abstraction is aware of the Customer class, a reservation is attempted to be made for a customer that does not yet exist.

Here, the customers.txt is shown as evidence that the one we'll attempt to create a reservation for does not exist.

customers.txt

```
{
    "C1": {"name": "Hector Segura", "contact": "48110731610"},
    "C2": {"name": "Maria Gonzalez", "contact": "55510437892"},
    "C3": {"name": "James Anderson", "contact": "3125678901"},
    "C4": {"name": "Isabella Ramirez", "contact": "9841234567"},
    "C5": {"name": "Daniel Thompson", "contact": "2128765432"},
    "C6": {"name": "Sofia Martinez", "contact": "6649081723"},
    "C7": {"name": "Matthew Johnson", "contact": "4159927364"},
    "C8": {"name": "Camila Fernandez", "contact": "7293485567"},
    "C9": {"name": "Christopher Lee", "contact": "8475639082"},
    "C10": {"name": "Valentina Torres", "contact": "5589763421"}
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
hectorseguraq@Laptop-de-Hector Actividad 6.2 % python
Python 3.10.13 (main, Sep 11 2023, 08:16:02) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from main import Hotel, Customer, Reservation
>>> 
```

Here is the result from attempting to use the Hotel class to reserve a room for a non-listed customer.

```
hectorseguraq@Laptop-de-Hector Actividad 6.2 % python
Python 3.10.13 (main, Sep 11 2023, 08:16:02) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from main import Hotel, Customer, Reservation
>>> Hotel.reserve_room("H1", "C11")
Error: Customer ID C11 is not listed in the system.
>>> 
```

Req 2. Below results are listed for each of the actions that each class should be able perform.

I. HOTEL

a. Create a hotel.

These are the last 3 elements on the JSON file hotels.txt, following a sequential order from H1 to H10.

```
    "H8": {
        "name": "Skyline Suites",
        "location": "Chicago",
        "rooms": 130,
        "reservations": []
    },
    "H9": {
        "name": "Hotel Riviera",
        "location": "Cannes",
        "rooms": 160,
        "reservations": []
    },
    "H10": {
        "name": "Urban Retreat",
        "location": "London",
        "rooms": 140,
        "reservations": []
    }
}
```

We use the program to create the eleventh element on the list, H11:

```
>>> Hotel.create("H11", "La Tregua", "San Luis Potosi", 80)
```

```
    },
    "H9": {
        "name": "Hotel Riviera",
        "location": "Cannes",
        "rooms": 160,
        "reservations": []
    },
    "H10": {
        "name": "Urban Retreat",
        "location": "London",
        "rooms": 140,
        "reservations": []
    },
    "H11": {
        "name": "La Tregua",
        "location": "San Luis Potosi",
        "rooms": 80,
        "reservations": []
    }
}
```

b. Delete a hotel.

From the list shown above, the H9 will be deleted.

```
>>> Hotel.delete("H9")
```

The updated list with the last 3 elements on hotels.txt:

```
    },
    "H8": {
        "name": "Skyline Suites",
        "location": "Chicago",
        "rooms": 130,
        "reservations": []
    },
    "H10": {
        "name": "Urban Retreat",
        "location": "London",
        "rooms": 140,
        "reservations": []
    },
    "H11": {
        "name": "La Tregua",
        "location": "San Luis Potosi",
        "rooms": 80,
        "reservations": []
    }
}
```

c. Display hotel information.

We will display information for the hotel with ID H8. Comparison can be made with reference above.

```
>>> Hotel.display("H8")
{'name': 'Skyline Suites', 'location': 'Chicago', 'rooms': 130, 'reservations': []}
```

d. Modify hotel information.

We will change the name of hotel with ID H10 to “Rustic Retreat”. Other elements will remain the same.

```
>>> Hotel.modify("H10", name="Rustic Retreat")
>>>
```

```
},
"H10": {
    "name": "Rustic Retreat",
    "location": "London",
    "rooms": 140,
    "reservations": []
},
```

e. Reserve a room.

A room will be reserved for the newly added H11 hotel. For this, we’ll use the customer C5 – document reference is at the beginning of the document.

```
>>> Hotel.reserve_room("H11", "C5")
```

Now, the hotels.txt has been updated to include this new reservation.

```
"H11": {
    "name": "La Tregua",
    "location": "San Luis Potosi",
    "rooms": 80,
    "reservations": [
        "C5"
    ]
}
```

f. Cancel a reservation.

We will cancel the reservation that was just created.

```
>>> Hotel.cancel_reservation("H11", "C5")
```

Now, the hotels.txt is updated once again showing an empty reservations list.

```
"H11": {
    "name": "La Tregua",
    "location": "San Luis Potosi",
    "rooms": 80,
    "reservations": []
}
```

II. CUSTOMER

a. Create a customer.

A new customer with ID C11 will be created.

```
>>> Customer.create("C11", "Tycho Brahe", "4811231923")
```

Updated customers.txt JSON file.

```
"C11": {
    "name": "Tycho Brahe",
    "contact": "4811231923"
}
```

b. Delete a customer.

Customer C9 will be deleted.

```
>>> Customer.delete("C9")
```

Updated customers.txt JSON file with more elements shown for context.

```
},
"C7": {
    "name": "Matthew Johnson",
    "contact": "4159927364"
},
"C8": {
    "name": "Camila Fernandez",
    "contact": "7293485567"
},
"C10": {
    "name": "Valentina Torres",
    "contact": "5589763421"
},
"C11": {
    "name": "Tycho Brahe",
    "contact": "4811231923"
}
```

c. Display customer information.

Information will be displayed for customer with ID C4.

```
>>> Customer.display("C4")
{'name': 'Isabella Ramirez', 'contact': '9841234567'}
```

JSON segment for reference.

```
"C3": {
    "name": "James Anderson",
    "contact": "3125678901"
},
"C4": {
    "name": "Isabella Ramirez",
    "contact": "9841234567"
```

d. Modify customer information.

The name of customer C3 will be changed to Steve McKaine.

Unaltered customer.

```
"C2": {
    "name": "Maria Gonzalez",
    "contact": "55510437892"
},
"C3": {
    "name": "Robert Martin",
    "contact": "3125678901"
},
```

Performing the change.

```
>>> Customer.modify("C3", name="Steve McKaine")
```

Updated JSON customers.txt file.

```
"C2": {
    "name": "Maria Gonzalez",
    "contact": "55510437892"
},
"C3": {
    "name": "Steve McKaine",
    "contact": "3125678901"
```

III. RESERVATION

a. Create a reservation (Customer, Hotel).

A reservation will be created at hotel H7 for customer C3.

```
>>> Reservation.create("C3", "H7")
```

Updated hotels.txt JSON file.

```
"H7": {
    "name": "Gran Hotel Central",
    "location": "Madrid",
    "rooms": 220,
    "reservations": [
        "C3"
    ]
},
```

b. Cancel a reservation.

The recently created reservation will be canceled.

```
>>> Reservation.cancel("C3", "H7")
```

Updated hotels.txt JSON file.

```
"H7": {
    "name": "Gran Hotel Central",
    "location": "Madrid",
    "rooms": 220,
    "reservations": []
},
```

Req 3.

The following unit tests were created to evaluate the methods within the established classes.

```
class TestHotelSystem(unittest.TestCase):
    def setUp(self):
        with open("hotels.txt", "w") as f:
            f.write("{}")
        with open("customers.txt", "w") as f:
            f.write("{}")

    def test_create_hotel(self):
        Hotel.create("H1", "Grand Hotel", "NY", 100)
        self.assertEqual(Hotel.display("H1")["name"], "Grand Hotel")

    def test_create_customer(self):
        Customer.create("C1", "John Doe", "1234567890")
        self.assertEqual(Customer.display("C1")["name"], "John Doe")

    def test_hreserve_room(self):
        Hotel.create("H2", "Declaracion Hotel", "NY", 2)
        Customer.create("C2", "Jane De", "1234567890")
        Hotel.reserve_room("H2", "C2")
        self.assertIn("C2", Hotel.display("H2")["reservations"])

    def test_reserve_room(self):
        Hotel.create("H1", "Grand Hotel", "NY", 2)
        Customer.create("C1", "John Doe", "1234567890")
        Reservation.create("C1", "H1")
        self.assertIn("C1", Hotel.display("H1")["reservations"])

    def test_hcancel_reservation(self):
        Hotel.create("H1", "Grand Hotel", "NY", 2)
        Customer.create("C1", "John Doe", "1234567890")
```

```python
        Hotel.reserve_room("H1", "C1")
        Hotel.cancel_reservation("H1", "C1")
        self.assertNotIn("C1", Hotel.display("H1")["reservations"])

    def test_cancel_reservation(self):
        Hotel.create("H1", "Grand Hotel", "NY", 2)
        Customer.create("C1", "John Doe", "1234567890")
        Reservation.create("C1", "H1")
        Reservation.cancel("C1", "H1")
        self.assertNotIn("C1", Hotel.display("H1")["reservations"])

    def test_no_available_rooms(self):
        Hotel.create("H1", "Grand Hotel", "NY", 1)
        Customer.create("C1", "John Doe", "1234567890")
        Customer.create("C2", "Jane Doe", "0987654321")
        Reservation.create("C1", "H1")
        self.assertNotIn("C2", Hotel.display("H1")["reservations"])

    def test_delete_hotel(self):
        Hotel.create("H1", "Grand Hotel", "NY", 100)
        Hotel.delete("H1")
        self.assertEqual(Hotel.display("H1"), "Hotel not found.")

    def test_modify_hotel(self):
        Hotel.create("H1", "Grand Hotel", "NY", 100)
        Hotel.modify("H1", name="Grann Hotel II")
        self.assertEqual(Hotel.display("H1")["name"], "Grann Hotel II")

    def test_delete_customer(self):
        Customer.create("C1", "John Doe", "1234567890")
        Customer.delete("C1")
        self.assertEqual(Customer.display("C1"), "Customer not found.")

    def test_modify_customer(self):
        Customer.create("C1", "John Doe", "1234567890")
        Customer.modify("C1", contact="1112223333")
        self.assertEqual(Customer.display("C1")["contact"], "1112223333")

    def test_display_customer(self):
        Customer.create("C1", "John Doe", "1234567890")
        self.assertEqual(Customer.display("C1")["name"], "John Doe")

if __name__ == "__main__":
    unittest.main()
```

Req 4.

Once the testing file is run, the results are:

```
hectorseguraq@Laptop-de-Hector unit % python main_test.py
............
----------------------------------------------------------------------
Ran 12 tests in 0.005s

OK
```

Req 5.

Some examples of how invalid data is managed throughout the main.py program are shown below.

```python
@staticmethod
def _load_hotels():
    if not os.path.exists(Hotel.FILE_NAME):
        return {}
    try:
        with open(Hotel.FILE_NAME, "r") as file:
            return json.load(file)
    except json.JSONDecodeError:
        print("Error: Corrupted hotel data file.")
        return {}

@staticmethod
def _load_customers():
    customer_file = "customers.txt"
    if not os.path.exists(customer_file):
        return {}
    try:
        with open(customer_file, "r") as file:
            return json.load(file)
    except json.JSONDecodeError:
        print("Error: Corrupted customer data file.")
        return {}
```

Req 6.

Both the main and test programs were developed based on PEP8 guidelines.

Req 7.

Below the results of analysis by both PyLint and Flake8 after making all corrections and sensible exceptions for main.py and main_test.py:

main.py × main_test.py customers.txt

main.py > ...

```python
# pylint: disable=protected-access, line-too-long
"""Program that manages Hotels, Customers and Reservations."""

import json
import os


class Hotel:
    """Class for managing Hotels information"""
    FILE_NAME = "hotels.txt"

    @staticmethod
    def _load_hotels():
        if not os.path.exists(Hotel.FILE_NAME):
            return {}
        try:
            with open(Hotel.FILE_NAME, "r", encoding="utf-8") as file:
                return json.load(file)
        except json.JSONDecodeError:
            print("Error: Corrupted hotel data file.")
            return {}

    @staticmethod
    def _load_customers():
        customer_file = "customers.txt"
        if not os.path.exists(customer_file):
            return {}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
main.py:93:0: C0301: Line too long (109/100) (line-too-long)

------------------------------------------------------------------
Your code has been rated at 9.86/10 (previous run: 10.00/10, -0.14)

hectorseguraq@Laptop-de-Hector Actividad 6.2 % pylint main.py

------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 9.86/10, +0.14)

hectorseguraq@Laptop-de-Hector Actividad 6.2 % pylint main.py

------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)

hectorseguraq@Laptop-de-Hector Actividad 6.2 % flake8 main.py
hectorseguraq@Laptop-de-Hector Actividad 6.2 %
```

main.py main_test.py × customers.txt

test > unit > main_test.py > ...

```python

    def test_modify_hotel(self):
        """Tests if a hotel's information can be modified successfully."""
        Hotel.create("H1", "Grand Hotel", "NY", 100)
        Hotel.modify("H1", name="Grann Hotel II")
        self.assertEqual(Hotel.display("H1")["name"], "Grann Hotel II")

    def test_delete_customer(self):
        """Tests if a customer can be deleted successfully."""
        Customer.create("C1", "John Doe", "1234567890")
        Customer.delete("C1")
        self.assertEqual(Customer.display("C1"), "Customer not found.")

    def test_modify_customer(self):
        """Tests if a customer's contact information can be modified."""
        Customer.create("C1", "John Doe", "1234567890")
        Customer.modify("C1", contact="1112223333")
        self.assertEqual(Customer.display("C1")["contact"], "1112223333")

    def test_display_customer(self):
        """Tests if a customer's details can be retrieved successfully."""
        Customer.create("C1", "John Doe", "1234567890")
        self.assertEqual(Customer.display("C1")["name"], "John Doe")


if __name__ == "__main__":
    unittest.main()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
hectorseguraq@Laptop-de-Hector unit % pylint main_test.py

------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)

hectorseguraq@Laptop-de-Hector unit % flake8 main_test.py
hectorseguraq@Laptop-de-Hector unit %
```

REFERENCES

PEP 0 – Index of Python Enhancement Proposals (PEPs) | peps.python.org. (s. f.). Python Enhancement Proposals (PEPs). https://peps.python.org/

unittest — Unit testing framework. (s. f.). Python Documentation. https://docs.python.org/3/library/unittest.html