16-2-2025

Gustavo Andrés García Anguiano

Pruebas de software y aseguramiento de la calidad

Actividad 6.2

INDICE

Main File

```
""""
This module implements a simple hotel reservation system.
It allows creating hotels, customers, and reservations.g
It also provides functionality to cancel reservations.
"""
import unittest
from pyclases.hotel_class import Hotel
from pyclases.customer_class import Customer
from pyclases.reservation_class import Reservation

class TestHotelReservationSystem(unittest.TestCase):
    """
    Unit tests for the hotel reservation system.
    """
    def __init__(self, methodName="runTest"):
        super().__init__(methodName)
        self.customer_name = "Gustavo"
        self.hotel_name = "Hotel California"
        self.mail = "a01795493@tec.mx"
        self.location = "Mexico"
        self.rooms = 10

    def test_create_hotel(self):
        """
        Test the creation of a hotel.
        """
        result = Hotel.create_hotel(
            self.hotel_name,
            self.location,
            self.rooms)
        self.assertIn(result, [
            "Hotel created successfully.",
            "Hotel already exists."
        ])

    def test_modify_hotel(self):
        """
        Test the modification of a hotel.
        """
        result = Hotel.modify_hotel_info(
            self.hotel_name,
            self.location,
            self.rooms)
        self.assertIn(
            result, [
                "Hotel modified successfully.",
                "Hotel not found."
            ])

    def test_get_hotel_info(self):
        """
        Test the retrieval of hotel information.
        """
        result = Hotel.display_hotel_info(self.hotel_name)
        self.assertIn(
            result, [
                "Hotel not found.",
                "Hotel information retrieved successfully."
            ])

    def test_create_customer(self):
        """
        Test the creation of a customer.
        """
        result = Customer.create_customer(
            self.customer_name,
            self.mail)
        self.assertIn(
            result, [
                "Customer created successfully.",
                "Customer already exists."
            ])

    def test_modify_customer(self):
        """
        Test the modification of a customer.
        """
        result = Customer.modify_customer(
            self.customer_name,
            self.mail)
        self.assertIn(
            result, [
                "Customer information modified successfully.",
                "Customer not found."
            ])

    def test_get_customer_info(self):
        """
        Test the retrieval of customer information.
        """
        result = Customer.display_customer(self.customer_name)
        self.assertIn(
            result, [
                "Customer not found.",
                "Customer information retrieved successfully."
            ])

    def test_create_reservation(self):
        """
        Test the creation of a reservation.
        """
        result = Reservation.create_reservation(
            self.customer_name, self.hotel_name)
        self.assertIn(
            result, [
                "Reservation created successfully.",
                "Customer already have a reservation.",
                "No rooms available."
            ])

    def test_cancel_reservation(self):
        """
        Test the cancellation of a reservation.
        """
        result = Reservation.cancel_reservation(
            self.customer_name, self.hotel_name)
        self.assertIn(
            result, [
                "Reservation cancelled successfully.",
                "Reservation not found."
            ])

    def test_delete_customer(self):
        """
        Test the deletion of a customer.
        """
        result = Customer.delete_customer(self.customer_name)
        self.assertIn(
            result, [
                "Customer deleted successfully.",
                "Customer not found."
            ])

    def test_delete_hotel(self):
        """
```

```python
        Test the deletion of a hotel.
        """
        result = Hotel.delete_hotel(self.hotel_name)
        self.assertIn(
            result, [
                "Hotel deleted successfully.",
                "Hotel not found."
            ])

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

CLASSES

Init Class

```python
"""
Module for handling hotel management system classes.
"""
import os
import glob

modules = glob.glob(os.path.join(os.path.dirname(__file__), "*.py"))
__all__ = [
    os.path.basename(f)[:-3]
    for f in modules
    if f.endswith(".py") and os.path.basename(f) != '__init__.py'
    ]
```

Hotel Class

```python
"""
Methods to handle the next persistent behaviors (stored in files):
1. Hotels
    a. Create Hotel
    b. Delete Hotel
    c. Display Hotel information
    d. Modify Hotel Information
    e. Reserve a Room
    f. Cancel a Reservation
"""
import uuid
from pyclases.handling_helpers import HandlingHelpers
from pyclases.reservation_class import Reservation

class Hotel:
    """
    A class to represent a hotel.
    """
    class_folder = "hotel_info"
    file = "hotel.json"
    helpers = HandlingHelpers()
    reservation = Reservation()

    @classmethod
    def create_hotel(cls, name, location, rooms):
        """
        Create a new hotel.
        Args:
            hotel_id (str): The ID of the hotel.
            name (str): The name of the hotel.
            location (str): The location of the hotel.
            rooms (int): The number of rooms in the hotel.
        Returns:
            str: Success message.
        """
        data = cls.helpers.load_data(cls.class_folder, cls.file)
        hotel_id = None
        for h_id, hotel in data.items():
            if hotel["name"] == name.lower():
                hotel_id = h_id
                break
        if not hotel_id or data == {}:
            hotel_id = str(uuid.uuid4())
            data[hotel_id] = {
                "name": name.lower(),
                "location": location.lower(),
                "rooms": rooms
            }
            cls.helpers.save_data(cls.class_folder, cls.file, data)
            return "Hotel created successfully."
        return "Hotel already exists."

    @classmethod
    def delete_hotel(cls, name):
        """
        Delete an existing hotel.
        Args:
            name (str): The name of the hotel.
        Returns:
            str: Success message or error message.
        """
        data = cls.helpers.load_data(cls.class_folder, cls.file)
        hotel_id = None
        for h_id, hotel in data.items():
            if hotel["name"] == name.lower():
                hotel_id = h_id
                break
        if hotel_id:
            del data[hotel_id]
            cls.helpers.save_data(cls.class_folder, cls.file, data)
            return "Hotel deleted successfully."
        return "Hotel not found."

    @classmethod
    def display_hotel_info(cls, name):
        """
        Display hotel information.
        Args:
            name (str): The name of the hotel.
        Returns:
            str: Hotel information or error message.
        """
        data = cls.helpers.load_data(cls.class_folder, cls.file)
        for hotel in data.values():
            if hotel["name"] == name.lower():
                return(
                        f"Hotel Name: {hotel['name']}, "
                        f"Location: {hotel['location']}, "
                        f"Rooms: {hotel['rooms']}")
        return "Hotel not found."

    @classmethod
    def modify_hotel_info(cls, name=None, location=None, rooms=None):
        """
        Modify hotel information by hotel name.
        Args:
            name (str): The new name of the hotel.
            location (str): The new location of the hotel.
            rooms (int): The new number of rooms in the hotel.
        Returns:
            str: Success message or error message.
        """
        data = cls.helpers.load_data(cls.class_folder, cls.file)
        hotel_id = None
        for h_id, hotel in data.items():
            if hotel["name"] == name.lower():
                hotel_id = h_id
                break
        if hotel_id:
            if name:
                data[hotel_id]["name"] = name.lower()
            if location:
                data[hotel_id]["location"] = location.lower()
            if rooms:
                data[hotel_id]["rooms"] = rooms
            cls.helpers.save_data(cls.class_folder, cls.file, data)
            return "Hotel information modified successfully."
        return "Hotel not found."

    @classmethod
    def reserve_room(cls, customer_name, hotel_name):
        """
        Reserve a room in a hotel.
        Args:
            customer_name (str): The name of the customer.
            hotel_name (str): The name of the hotel.
        Returns:
            str: Success message or error message.
        """
        cls.reservation.create_reservation(customer_name, hotel_name)
        return "Room reserved successfully."

    @classmethod
    def cancel_reservation(cls, customer_name, hotel_name):
        """
        Cancel a reservation.
        Args:
            customer_name (str): The name of the customer.
            hotel_name (str): The name of the hotel.
        Returns:
            str: Success message or error message.
        """
        cls.reservation.cancel_reservation(customer_name, hotel_name)
        return "Reservation cancelled successfully."
```

Customer Class

```
"""
Methods to handle the next persistent behaviors (stored in files):
1. Customers
    a. Create Customer
    b. Delete Customer
    c. Display Customer information
    d. Modify Customer Information
"""
import uuid
from pyclases.handling_helpers import HandlingHelpers

class Customer:
    """
    A class to represent a customer.
    """
    class_folder = "customer_info"
    file = "customers.json"
    helpers = HandlingHelpers()

    @classmethod
    def create_customer(cls, name, email):
        """"
        Create a new customer.
        Args:
            customer_id (str): The ID of the customer.
            name (str): The name of the customer.
            email (str): The email of the customer.
        Returns:
            str: Success message.
        """
        data = cls.helpers.load_data(cls.class_folder, cls.file)
        customer_id = None
        for c_id, customer in data.items():
            if customer["name"] == name.lower():
                customer_id = c_id
                break
        if not customer_id or data == {}:
            customer_id = str(uuid.uuid4())
            data[customer_id] = {
                "name": name.lower(),
                "email": email.lower()
            }
            cls.helpers.save_data(cls.class_folder, cls.file, data)
            return "Customer created successfully."
        return "Customer already exists."

    @classmethod
    def delete_customer(cls, customer_name):
        """
        Delete an existing customer.
        Args:
            customer_name (str): The name of the customer.
        Returns:
            str: Success message or error message.
        """
        data = cls.helpers.load_data(cls.class_folder, cls.file)
        customer_id = None
        for c_id, customer in data.items():
            if customer["name"] == customer_name.lower():
                customer_id = c_id
                break
        if customer_id:
            del data[customer_id]
            cls.helpers.save_data(cls.class_folder, cls.file, data)
            return "Customer deleted successfully."
        return "Customer not found."

    @classmethod
    def display_customer(cls, customer_name):
        """
        Display customer information.
        Args:
            customer_name (str): The name of the customer.
        Returns:
            str: Customer information or error message.
        """
        data = cls.helpers.load_data(cls.class_folder, cls.file)
        for _, customer in data.items():
            if customer["name"] == customer_name.lower():
                return(f"Customer Name: {customer['name']}, "
                      f"Customer Email: {customer['email']}")
        return "Customer not found."

    @classmethod
    def modify_customer(cls, name, email):
        """
        Modify customer information.
        Args:
            name (str): The name of the customer.
            email (str): The new email of the customer.
        Returns:
            str: Success message or error message.
        """
        data = cls.helpers.load_data(cls.class_folder, cls.file)
        customer_id = None
        for c_id, customer in data.items():
            if customer["name"] == name.lower():
                customer_id = c_id
                break
        if customer_id:
            data[customer_id]["email"] = email.lower()
            cls.helpers.save_data(cls.class_folder, cls.file, data)
            return "Customer information modified successfully."
        return "Customer not found."
```

Reservation Class

```
""""
 A class to represent a reservation system.
    It allows creating, deleting, and modifying reservations.
    It also provides functionality to check
    - available rooms
    - cancel reservations.
"""
import uuid
from pyclases.handling_helpers import HandlingHelpers

class Reservation:
    """
    A class to represent a customer.
    """
    class_folder = "reservation_info"
    hotel_folder = "hotel_info"
    file = "reservation.json"
    hotel_file = "hotel.json"
    customer_file = "customers.json"
    helpers = HandlingHelpers()

    @classmethod
    def create_reservation(cls, customer_name, hotel_name):
        """
        A class to create a new reservation.
        """
        reservation_id = str(uuid.uuid4())
        data = cls.helpers.load_data(cls.class_folder, cls.file)
        hotels = cls.helpers.load_data(cls.hotel_folder, cls.hotel_file)
        hotels_id = None
        reservation_id = None
        for r_id, reservation in data.items():
            if (
                reservation["hotel_name"] == hotel_name.lower() and
                reservation["customer_name"] == customer_name.lower()
            ):
                reservation_id = r_id
            break
        if reservation_id:
            return "Customer already have a reservation."
        for h_id, hotel in hotels.items():
            if hotel["name"] == hotel_name.lower():
                hotels_id = h_id
                break
        if hotels[hotels_id]["rooms"] == 0:
            return "No rooms available."
        data[reservation_id] = {
            "customer_name": customer_name.lower(),
            "hotel_name": hotel_name.lower(),
        }
        hotels[hotels_id]["rooms"] -= 1
        cls.helpers.save_data(cls.hotel_folder, cls.hotel_file, hotels)
        cls.helpers.save_data(cls.class_folder, cls.file, data)
        return "Reservation created successfully."

    @classmethod
    def cancel_reservation(cls, customer_name, hotel_name):
        """
        A class to cancel a reservation.
        Args:
            customer_name (str): The name of the customer.
            hotel_name (str): The name of the hotel.
        Returns:
            str: Success message or error message.
        """
        data = cls.helpers.load_data(cls.class_folder, cls.file)
        hotels = cls.helpers.load_data(cls.hotel_folder, cls.hotel_file)
        hotels_id = None
        reservation_id = None
        for r_id, reservation in data.items():
            if (
                reservation["customer_name"] == customer_name.lower() and
                reservation["hotel_name"] == hotel_name.lower()
            ):
                reservation_id = r_id
                break
        if reservation_id:
            del data[reservation_id]
            cls.helpers.save_data(cls.class_folder, cls.file, data)
            for h_id, hotel in hotels.items():
                if hotel["name"] == hotel_name.lower():
                    hotels_id = h_id
                    break
            if hotels_id:
                hotels[hotels_id]["rooms"] += 1
            cls.helpers.save_data(cls.hotel_folder, cls.hotel_file, hotels)
            return "Reservation cancelled successfully."
        return "Reservation not found."
```

Handling Helpers Class

```
""""
handling_helpers.py
This module provides helper functions to handle data loading and saving.
It includes methods to load data from a JSON file and save data to a JSON file.
"""
import json
import os

class HandlingHelpers:
    """
    A class to handle data loading and saving.
    """
    def __init__(self):
        """
        Initialize the HandlingHelpers class.
        """
        base_dir = os.path.dirname(os.path.abspath(__file__))
        self.project_dir = os.path.abspath(os.path.join(base_dir, ".."))

    def load_data(self, class_folder, filename):
        """
        Load data from a JSON file.
        Args:
            filename (str): The name of the file to load data from.
            Args:
```

```
            data (dict): The data to save to the file.
        Returns:
            dict: The loaded data.
        """
        file_path = os.path.join(self.project_dir, class_folder, filename)
        if os.path.exists(file_path):
            try:
                with open(file_path, 'r', encoding='utf-8-sig') as file:
                    return json.load(file)
            except json.JSONDecodeError:
                return {}
        return {}

    def save_data(self, class_folder, filename, data):
        """
        Save data to a JSON file.
        Args:
            filename (str): The name of the file to save data to.
            data (dict): The data to save to the file.
        """
        file_path = os.path.join(self.project_dir, class_folder, filename)
        try:
            with open(file_path, 'w', encoding='utf-8-sig') as file:
                json.dump(data, file, indent=4)
        except json.JSONDecodeError:
            with open(file_path, 'w', encoding='utf-8-sig') as file:
                json.dump({}, file, indent=4)
```