Assignment 3 - Final Project

Abdullah M. Awad Aljeaidi - 202112794

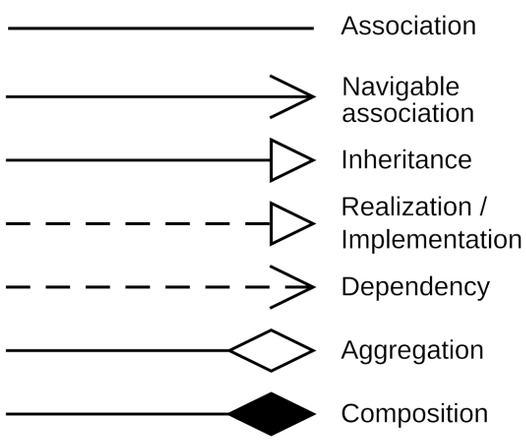
Saif Abdulla Alhammadi - 202220746

College of Interdisciplinary Studies

ICS220: Programming Fundamentals

Prof. Afshan Parkar

December 5, 2024



VIPExperiencePass

- ticket_id: String
- price: Float (default: 550)
- validity: String (default: "1 Day")
- description: String (default: "Includes expedited access and reserved seating.")

+ get_ticket_id(): String
+ set_ticket_id(ticket_id: String): void
+ get_price(): Float
+ set_price(price: Float): void
+ get_validity(): String
+ set_validity(validity: String): void
+ get_description(): String
+ set_description(description: String): void

ChildTicket

-ticket_id: String
-price: Float (default: 185)
-validity: String (default: "1 Day")
-description: String (default: "Discounted ticket for children (3-12 years).")

+get_ticket_id(): String
+set_ticket_id(ticket_id: String): void
+get_price(): Float
+set_price(price: Float): void
+get_validity(): String
+set_validity(validity: String): void
+get_description(): String
+set_description(description: String): void

SingleDayPass

-ticket_id: String
-price: Float (default: 275)
-validity: String (default: "1 Day")
-description: String (default: "Access for one day.")

+get_ticket_id(): String
+set_ticket_id(ticket_id: String): void
+get_price(): Float
+set_price(price: Float): void
+get_validity(): String
+set_validity(validity: String): void
+get_description(): String
+set_description(description: String): void

TicketBookingSystem

- tickets: List<Ticket>
- users: List<User>
- admins: List<Admin>
- ticket_sales: Dict<String, Integer>

+get_tickets(): List<Ticket>
+set_tickets(tickets: List<Ticket>): void
+get_users(): List<User>
+set_users(users: List<User>): void
+register_user(user: User): void
+add_ticket(ticket: Ticket): void
+purchase_ticket(user: Customer, ticket:
+Ticket): void
+save_data(filename: String): void
+load_data(filename: String): Object

Ticket

#ticket_id: String
#price: Float
#validity: String
#description: String

+get_ticket_id(): String
+set_ticket_id(ticket_id: String): void
+get_price(): Float
+set_price(price: Float): void
+get_validity(): String
+set_validity(validity: String): void
+get_description(): String
+set_description(description: String): void
+display_ticket_info(): String

User

user_id: Integer
name: String
email: String
purchase_history: List<Ticket>

+ get_user_id(): Integer
+ set_user_id(user_id: Integer): void
+ get_name(): String
+ set_name(name: String): void
+ get_email(): String
+ set_email(email: String): void
+ get_purchase_history(): List<Ticket>
+ add_purchase(ticket: Ticket): void
+ display_user_info(): String

TwoDayPass

-ticket_id: String
-price: Float (default: 480)
-validity: String (default: "2 Days")
-description: String (default: "Access for two consecutive days.")

+get_ticket_id(): String
+set_ticket_id(ticket_id: String): void
+get_price(): Float
+set_price(price: Float): void
+get_validity(): String
+set_validity(validity: String): void
+get_description(): String
+set_description(description: String): void

GroupTicket

-ticket_id: String
-price: Float (default: 220)
-validity: String (default: "1 Day")
-description: String (default: "Special rate for groups of 10 or more.")

+get_ticket_id(): String
+set_ticket_id(ticket_id: String): void
+get_price(): Float
+set_price(price: Float): void
+get_validity(): String
+set_validity(validity: String): void
+get_description(): String
+set_description(description: String): void

AnnualMembership

-ticket_id: String
-price: Float (default: 1840)
-validity: String (default: "1 Year")
-description: String (default: "Unlimited access for one year.")

+get_ticket_id(): String
+set_ticket_id(ticket_id: String): void
+get_price(): Float
+set_price(price: Float): void
+get_validity(): String
+set_validity(validity: String): void
+get_description(): String
+set_description(description: String): void

Admin

- admin_id: Integer
- name: String

+ get_admin_id(): Integer
+ set_admin_id(admin_id: Integer): void
+ get_name(): String
+ set_name(name: String): void

Customer

- purchase_history: List<Ticket>

+ get_purchase_history(): List<Ticket>
+ set_purchase_history(purchase_history: List<Ticket>): void
+ add_to_history(ticket: Ticket): void
+ view_purchase_history(): String

Class 1	Class 2	Association Type	Cardinality	Relationship Description
TicketBookingSystem	User	Aggregation	1 (TicketBookingSystem) : * (User)	The TicketBookingSystem manages multiple users (both Admin and Customer).
TicketBookingSystem	Ticket	Aggregation	1 (TicketBookingSystem) : * (Ticket)	The TicketBookingSystem maintains a collection of tickets available for purchase.
User	Ticket	Composition	1 (User) : * (Ticket)	A User has a purchase history composed of multiple tickets they own directly.
User	Admin	Inheritance	1 (User) : 1 (Admin)	Admin inherits from the User class, adding attributes and methods specific to administration.
Ticket	SingleDayPass	Inheritance	1 (User) : 1 (Customer)	Customer inherits from the User class, adding attributes and methods specific to ticket purchasing.
Ticket	TwoDayPass	Inheritance	1 (Ticket) : 1 (TwoDayPass)	TwoDayPass is a specific type of Ticket with its own predefined attributes such as price and description.
Ticket	AnnualMembership	Inheritance	1 (Ticket) : 1 (AnnualMembership)	AnnualMembership is a specific type of Ticket with its own predefined attributes such as price and description.
Ticket	ChildTicket	Inheritance	1 (Ticket) : 1 (ChildTicket)	ChildTicket is a specific type of Ticket with its own predefined attributes such as price and description.
Ticket	GroupTicket	Inheritance	1 (Ticket) : 1 (GroupTicket)	GroupTicket is a specific type of Ticket with its own predefined attributes such as price and description.
Ticket	VIPExperiencePass	Inheritance	1 (Ticket) : 1 (VIPExperiencePass)	VIPExperiencePass is a specific type of Ticket with its own predefined attributes such as price and description.

ticket_system.py:

```
import pickle

# Base Ticket class
class Ticket:
    # Ticket attributes include ID, price, validity, and description
    def __init__(self, ticket_id, price, validity, description):
        self._ticket_id = ticket_id
        self._price = price
        self._validity = validity
        self._description = description

    # Getters for Ticket attributes
    def get_ticket_id(self):
        return self._ticket_id

    def get_price(self):
        return self._price

    def get_validity(self):
        return self._validity

    def get_description(self):
        return self._description

    # Setters for Ticket attributes
    def set_ticket_id(self, ticket_id):
        self._ticket_id = ticket_id

    def set_price(self, price):
        self._price = price

    def set_validity(self, validity):
        self._validity = validity

    def set_description(self, description):
```

```
        self._description = description

    # Method to display ticket information
    def display_ticket_info(self):
        return f"ID: {self._ticket_id}, Price: {self._price} DHS, Validity:
{self._validity}, Description: {self._description}"

# Subclasses for specific ticket types
class SingleDayPass(Ticket):
    # Single day pass ticket with fixed attributes
    def __init__(self, ticket_id):
        super().__init__(ticket_id, 275, "1 Day", "Access for one day.")

class TwoDayPass(Ticket):
    # Two day pass ticket with fixed attributes
    def __init__(self, ticket_id):
        super().__init__(ticket_id, 480, "2 Days", "Access for two
consecutive days.")

class AnnualMembership(Ticket):
    # Annual membership with fixed attributes
    def __init__(self, ticket_id):
        super().__init__(ticket_id, 1840, "1 Year", "Unlimited access for
one year.")

class ChildTicket(Ticket):
    # Discounted ticket for children
    def __init__(self, ticket_id):
        super().__init__(ticket_id, 185, "1 Day", "Discounted ticket for
children (3-12 years).")

class GroupTicket(Ticket):
    # Group ticket with fixed attributes
    def __init__(self, ticket_id):
        super().__init__(ticket_id, 220, "1 Day", "Special rate for groups
of 10 or more.")
```

```
class VIPExperiencePass(Ticket):
    # VIP experience pass with fixed attributes
    def __init__(self, ticket_id):
        super().__init__(ticket_id, 550, "1 Day", "Includes expedited
access and reserved seating.")

# Base User class
class User:
    # User attributes include ID, name, email, and purchase history
    def __init__(self, user_id, name, email):
        self._user_id = user_id
        self._name = name
        self._email = email

    # Getters for User attributes
    def get_user_id(self):
        return self._user_id

    def get_name(self):
        return self._name

    def get_email(self):
        return self._email

    # Setters for User attributes
    def set_user_id(self, user_id):
        self._user_id = user_id

    def set_name(self, name):
        self._name = name

    def set_email(self, email):
        self._email = email

    # Method to display user information
```

```
    def display_user_info(self):
        return f"User ID: {self._user_id}, Name: {self._name}, Email:
{self._email}"

# Admin subclass inherits from User
class Admin(User):
    def __init__(self, user_id, name, email):
        super().__init__(user_id, name, email)

# Customer subclass inherits from User
class Customer(User):
    def __init__(self, user_id, name, email):
        super().__init__(user_id, name, email)
        self._purchase_history = []

    # Getters and setters for purchase history
    def get_purchase_history(self):
        return self._purchase_history

    def set_purchase_history(self, purchase_history):
        self._purchase_history = purchase_history

    # Add a ticket to purchase history
    def add_to_history(self, ticket):
        self._purchase_history.append(ticket)

    # View all tickets in purchase history
    def view_purchase_history(self):
        return [ticket.display_ticket_info() for ticket in
self._purchase_history]

# TicketBookingSystem class manages users and tickets
class TicketBookingSystem:
    def __init__(self):
        self._users = {}
        self._tickets = []
```

```
        self.load_data()  # Load data from file if it exists

    # Getters and setters for tickets and users
    def get_users(self):
        return self._users

    def get_tickets(self):
        return self._tickets

    def set_users(self, users):
        self._users = users

    def set_tickets(self, tickets):
        self._tickets = tickets

    # Register a new user
    def register_user(self, user):
        self._users[user.get_user_id()] = user
        self.save_data()

    # Add a ticket to the system
    def add_ticket(self, ticket):
        self._tickets.append(ticket)
        self.save_data()

    # Save data to file using pickle
    def save_data(self):
        with open("data.pkl", "wb") as file:
            pickle.dump({"users": self._users, "tickets": self._tickets},
file)

    # Load data from pickle file
    def load_data(self):
        try:
            with open("data.pkl", "rb") as file:
                data = pickle.load(file)
```

```
                self._users = data["users"]
                self._tickets = data["tickets"]
        except FileNotFoundError:
            self._users = {}
            self._tickets = []

    # Retrieve a user by ID
    def get_user(self, user_id):
        return self._users.get(user_id)

    # Delete a user by ID
    def delete_user(self, user_id):
        if user_id in self._users:
            del self._users[user_id]
            self.save_data()

    # Modify user details
    def modify_user(self, user_id, new_name, new_email):
        if user_id in self._users:
            user = self._users[user_id]
            user.set_name(new_name)
            user.set_email(new_email)
            self.save_data()
```

gui.py:

```
import tkinter as tk
from tkinter import messagebox
from ticket_system import TicketBookingSystem, Customer, Admin,
SingleDayPass, TwoDayPass, AnnualMembership, ChildTicket, GroupTicket,
VIPExperiencePass

class TicketBookingGUI:
    def __init__(self, master):
        # Initialize the Ticket Booking System and the GUI
```

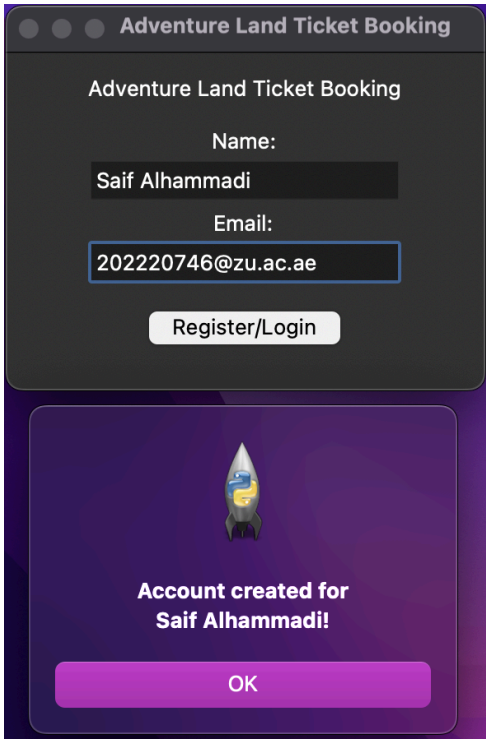
```
        self.system = TicketBookingSystem()
        self.master = master
        self.master.title("Adventure Land Ticket Booking")
        self.frame = tk.Frame(self.master)
        self.frame.pack()

        self.user = None  # Currently logged-in user
        self.login_screen()  # Start with the login screen

    def login_screen(self):
        # Clear the frame and show the login/registration screen
        self.clear_frame()

        tk.Label(self.frame, text="Adventure Land Ticket
Booking").pack(pady=10)
        tk.Label(self.frame, text="Name:").pack()
        self.name_entry = tk.Entry(self.frame)  # Entry for name
        self.name_entry.pack()

        tk.Label(self.frame, text="Email:").pack()
        self.email_entry = tk.Entry(self.frame)  # Entry for email
        self.email_entry.pack()

        tk.Button(self.frame, text="Register/Login",
command=self.register_or_login).pack(pady=10)

    def register_or_login(self):
        # Handle user registration or login
        name = self.name_entry.get()
        email = self.email_entry.get()

        if not name or not email:
            # Show an error if fields are empty
            messagebox.showerror("Error", "Both fields are required!")
            return
```

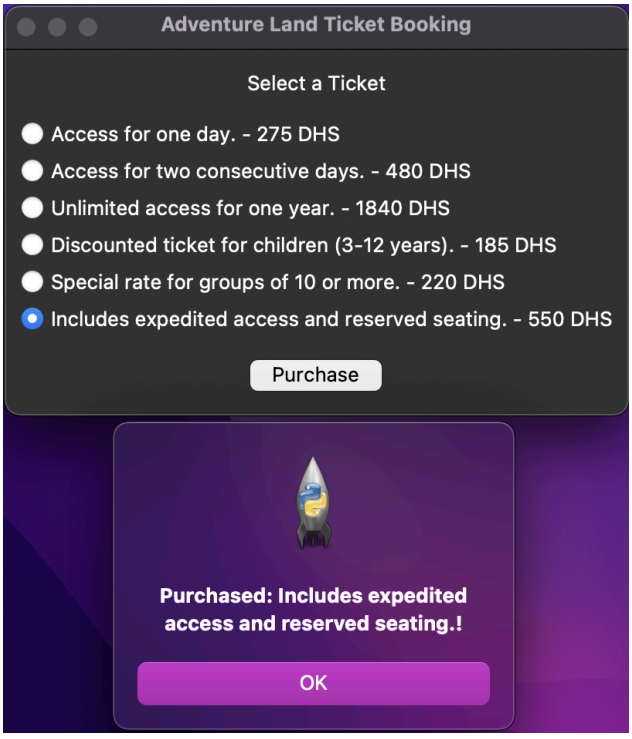
```
        user_id = f"{name.lower().replace(' ', '')}_{email.split('@')[0]}"
        user = self.system.get_user(user_id)
        if user:
            # Existing user, log them in
            self.user = user
            messagebox.showinfo("Welcome Back", f"Welcome back,
{self.user.get_name()}!")
        else:
            # New user, register them
            self.user = Customer(user_id, name, email)
            self.system.register_user(self.user)
            messagebox.showinfo("Registration Successful", f"Account
created for {self.user.get_name()}!")

        self.ticket_selection_screen()

    def ticket_selection_screen(self):
        # Show the ticket selection screen
        self.clear_frame()

        tk.Label(self.frame, text="Select a Ticket").pack(pady=10)

        self.ticket_var = tk.StringVar()  # Variable to hold selected
ticket type
        for ticket_class in [SingleDayPass, TwoDayPass, AnnualMembership,
ChildTicket, GroupTicket, VIPExperiencePass]:
            ticket = ticket_class(ticket_class.__name__)
            # Display ticket options as radio buttons
            tk.Radiobutton(
                self.frame, text=f"{ticket.get_description()} -
{ticket.get_price()} DHS",
                variable=self.ticket_var, value=ticket_class.__name__
            ).pack(anchor="w")

        tk.Button(self.frame, text="Purchase",
command=self.purchase_ticket).pack(pady=10)
```

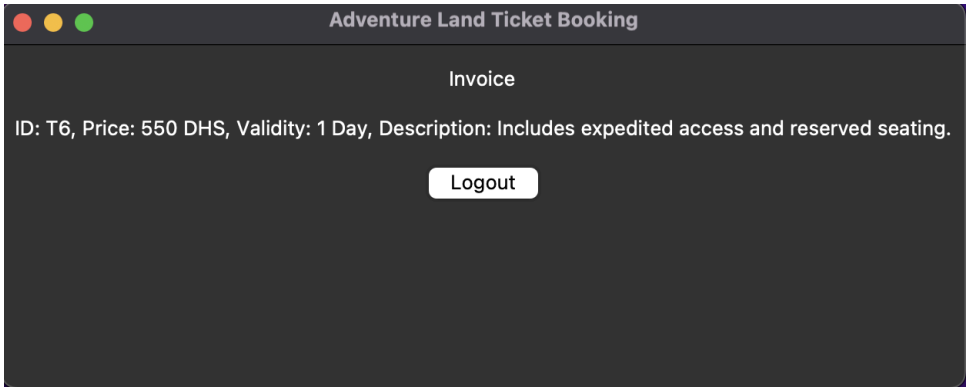
```
    def purchase_ticket(self):
        # Handle ticket purchase
        ticket_class_name = self.ticket_var.get()
        if not ticket_class_name:
            # Show an error if no ticket is selected
            messagebox.showerror("Error", "Please select a ticket!")
            return

        ticket_class = globals()[ticket_class_name]
        ticket = ticket_class(f"T{len(self.system.get_tickets()) + 1}")  # Create a new ticket
        self.user.add_to_history(ticket)  # Add ticket to user's purchase history
        self.system.add_ticket(ticket)  # Add ticket to the system
        messagebox.showinfo("Success", f"Purchased: {ticket.get_description()}!")
        self.invoice_screen()

    def invoice_screen(self):
        # Show the invoice screen
        self.clear_frame()

        tk.Label(self.frame, text="Invoice").pack(pady=10)
        for ticket in self.user.get_purchase_history():
            # Display all purchased tickets
            tk.Label(self.frame, text=ticket.display_ticket_info()).pack(anchor="w")

        tk.Button(self.frame, text="Logout", command=self.login_screen).pack(pady=10)

    def clear_frame(self):
        # Clear all widgets from the current frame
        for widget in self.frame.winfo_children():
            widget.destroy()
```

```python
if __name__ == "__main__":
    root = tk.Tk()  # Create the main Tkinter window
    app = TicketBookingGUI(root)  # Initialize the Ticket Booking GUI
    root.mainloop()  # Start the main GUI event loop
```

GUI Implementation:







1. **(Registration/Login Screen):**

   The user is prompted to enter their name and email to register or log in. After entering the details, a message confirms the account creation for "Saif Alhammadi."

2. **(Ticket Selection Screen):**

   The user selects a ticket from multiple options. In this case, the "VIP Experience Pass" is selected, and a confirmation message is displayed upon purchase.

3. **(Invoice Screen):**

   The invoice for the purchased ticket is displayed, showing details such as ticket ID, price, validity, and description. A "Logout" button is available to return to the login screen.

Github Link:

https://github.com/saif-alh/ICS220-Final-Project-Saif-and-Abdulla.git

Summary of learnings:

This final assignment provided a deep insight into designing and developing a real-world software application using Object-Oriented principles:

#LO1_OOAD: Using UML diagrams, we learned how to analyze real-world problems and represent them visually. We identified the entities, attributes, and relationships, such as inheritance, aggregation, and composition, to create a clear system blueprint.

#LO2_OOProgramming: To implement the UML design, we learned to write clean, object-oriented Python programs. By creating classes, attributes, and methods, we transformed the diagram into functional code that could solve problems and handle errors easily.

#LO3_SWImplementation: We developed a multi-layered program with a user-friendly GUI that allows users to create, read, update, and delete data. We ensured the software was interactive and met all functional requirements, such as managing tickets, accounts, and sales.

#LO4_SWDocumentation: We improved our abilities to write clear documentation and explanations and ensured they were understandable. This included the UML diagram, the code structure, the GUI features, and the testing methods. This made sure everything was easy to understand and maintain.

Overall, this project helped us understand how to design and build a real-world system using UML diagrams, Python programming, and a user-friendly GUI. We learned to turn ideas into functional software while ensuring it is well-documented and meets all user needs. This final assignment improved our skills in problem-solving, teamwork, and creating clear and easy solutions.