Grand Prix Ticketing Experience

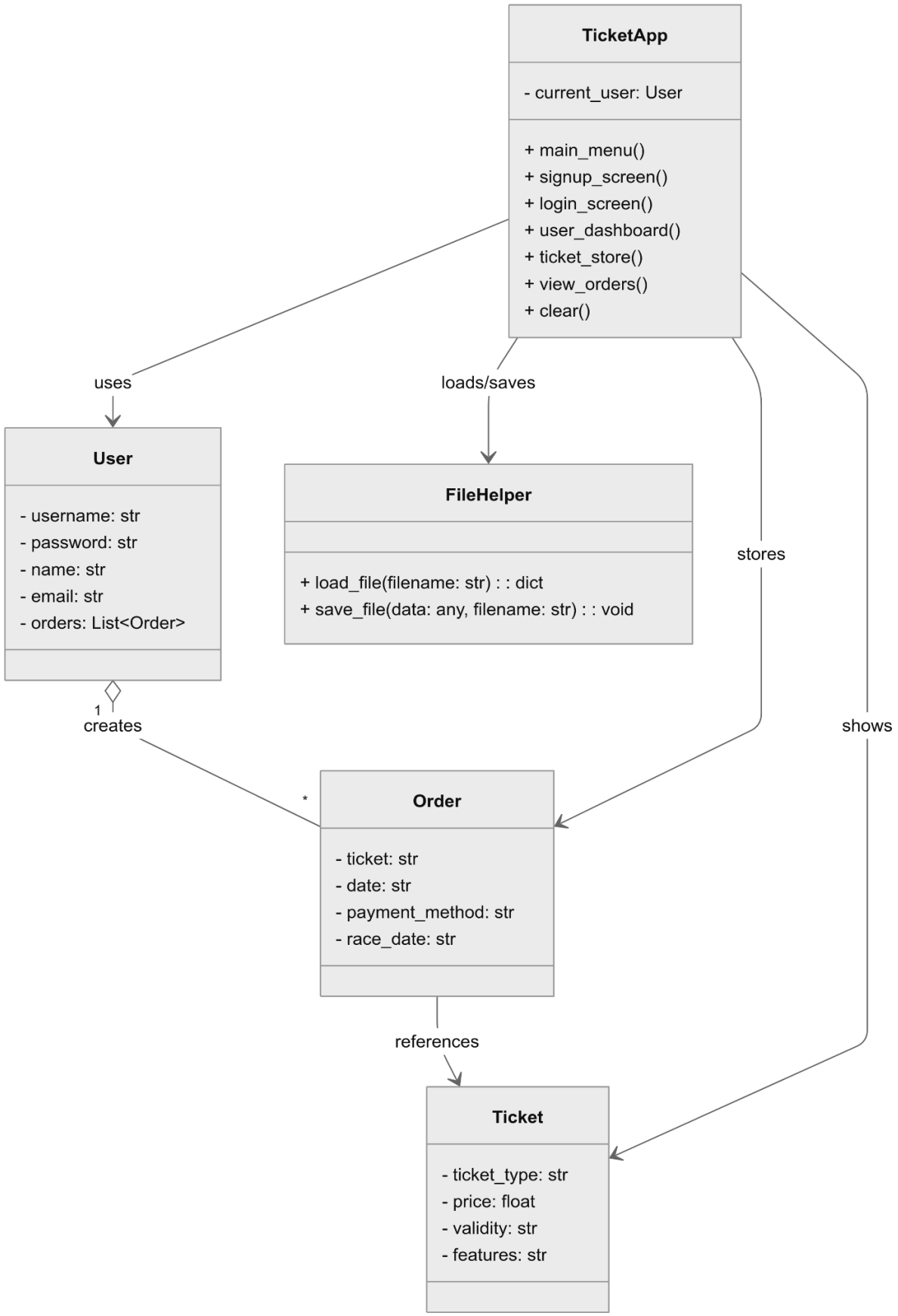
Mohammad Ismail 202105609

Nasser Lootah 202221929

ICS220 > 21383 Program. Fund.

May 13, 2025

UML Class Diagram:



User

- Attributes: username, password, name, email, and a list of orders
- Each User can create multiple Order objects. These are stored in the orders list.
- The class models a customer account and handles individual purchase history.

Each user object plays a key role in personalizing the application experience and ensures that the user specific data such as past purchases and the credentials too, are maintained securely.

Order

- Attributes: ticket, date, payment_method, and race_date
- Each Order object stores details of a single ticket purchase made by a user.
- It references ticket type as a string, not a direct object, to keep storage simple.

This design avoids circular references and simplifies the serialization process using pickle. Each order instance serves as a transaction record for the audit or tracking purposes.

Ticket

- Attributes: ticket_type, price, validity, and features
- Represents the available ticket options in the system.
- Tickets are predefined in the code and not editable by the admin.

These ticket instances are constants used across the application and help standardize the available options for all of the users. They include descriptive features just like access privileges or the time validity to help users make informed decisions.

TicketApp

- Attribute: current_user for session tracking
- Methods handle all GUI views and user actions: login, signup, ticket purchase, admin panel
- Manages system logic and controls access to users, orders, and tickets

TicketApp acts as the main orchestrator, linking the front-end interface with the underlying data models and storage. It ensures a smooth user journey and handles the flow of information between different components of the system.

Relationships

- User has a one-to-many relationship with Order (composition)
- Order is associated with Ticket (by ticket type string, not object reference)
- TicketApp acts as the controller, handling GUI events, data saving, and class interaction

So each user has multiple orders and if the user is deleted, then the orders are removed too (composition). Each order stores which type of Ticket was purchased, but it does not connect directly to a ticket object to keep storage simple. TicketApps connect all of the parts together and control what the user sees and what happens in the backend when users interact with the system. There is no direct object to object link between Order and Ticket or Users and Ticket opps, so the relationships are managed through method call and data matching.

Assumptions

- Tickets are hardcoded for simplicity and to meet the requirement of disabling admin ticket creation
- Data is stored using pickle in separate binary files: users.pkl, orders.pkl, and discount.pkl
- No need for inheritance as all classes serve distinct roles without overlapping behavior
- Errors like invalid input or missing files are handled gracefully using basic exception handling

It assumes that only trusted users will access the system, so advanced security features like encryption or multi-user permission are not implemented. The use of pickle is based on the assumption of a controlled environment where binary files are not tampered with.

Full Code:

```
import tkinter as tk
from tkinter import messagebox
import pickle
import os
from datetime import datetime

class User:
    def __init__(self, username, password, name, email):
        self.username = username
        self.password = password
        self.name = name
        self.email = email
        self.orders = []

class Ticket:
```

```
    def __init__(self, ticket_type, price, validity, features):
        self.ticket_type = ticket_type
        self.price = price
        self.validity = validity
        self.features = features

class Order:
    def __init__(self, ticket, date, payment_method, race_date):
        self.ticket = ticket
        self.date = date
        self.payment_method = payment_method
        self.race_date = race_date

# -------------------------
# File Handling
# -------------------------

def load_file(filename):
    if os.path.exists(filename):
        with open(filename, 'rb') as f:
            return pickle.load(f)
    return {}

def save_file(data, filename):
    with open(filename, 'wb') as f:
        pickle.dump(data, f)

users = load_file('users.pkl')
orders = load_file('orders.pkl')
# No admin or discount features yet

tickets = {
    'Single Race': Ticket('Single Race', 100.0, '1 Day', 'Access to 1 race'),
    'Weekend Pass': Ticket('Weekend Pass', 250.0, '3 Days', 'All weekend
races'),
    'Season Membership': Ticket('Season Membership', 1000.0, 'Full Season',
'All races'),
    'Group Discount': Ticket('Group Discount', 80.0, '1 Day', 'Minimum 5
people')
}



class TicketApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Grand Prix Ticket Booking - Part 2")
        self.root.geometry("700x500")
        self.current_user = None
        self.main_menu()

    def main_menu(self):
        self.clear()
```

```
        tk.Label(self.root, text="Grand Prix Ticket Booking", font=("Arial",
20)).pack(pady=20)
        tk.Button(self.root, text="Login", width=30,
command=self.login_screen).pack(pady=10)
        tk.Button(self.root, text="Sign Up", width=30,
command=self.signup_screen).pack(pady=10)

    def signup_screen(self):
        self.clear()
        tk.Label(self.root, text="Create Account", font=("Arial",
14)).pack(pady=10)
        entries = {}
        for field in ["Username", "Password", "Name", "Email"]:
            tk.Label(self.root, text=field).pack()
            entry = tk.Entry(self.root, show="*" if field == "Password" else
"")
            entry.pack()
            entries[field.lower()] = entry

        def create_user():
            u = entries['username'].get()
            if u in users:
                messagebox.showerror("Error", "Username already exists")
                return
            users[u] = User(
                u,
                entries['password'].get(),
                entries['name'].get(),
                entries['email'].get()
            )
            save_file(users, 'users.pkl')
            messagebox.showinfo("Success", "Account created")
            self.main_menu()

        tk.Button(self.root, text="Create Account",
command=create_user).pack(pady=10)
        tk.Button(self.root, text="Back", command=self.main_menu).pack()

    def login_screen(self):
        self.clear()
        tk.Label(self.root, text="Login", font=("Arial", 14)).pack(pady=10)
        tk.Label(self.root, text="Username").pack()
        username_entry = tk.Entry(self.root)
        username_entry.pack()
        tk.Label(self.root, text="Password").pack()
        password_entry = tk.Entry(self.root, show="*")
        password_entry.pack()

        def login():
            u = username_entry.get()
            p = password_entry.get()
            if u in users and users[u].password == p:
                self.current_user = users[u]
```

```
                self.user_dashboard()
            else:
                messagebox.showerror("Error", "Invalid login")

        tk.Button(self.root, text="Login", command=login).pack(pady=10)
        tk.Button(self.root, text="Back", command=self.main_menu).pack()

    def user_dashboard(self):
        self.clear()
        tk.Label(self.root, text=f"Welcome {self.current_user.name}",
font=("Arial", 16)).pack(pady=10)
        tk.Button(self.root, text="Buy Ticket", width=30,
command=self.ticket_store).pack(pady=5)
        tk.Button(self.root, text="View My Orders", width=30,
command=self.view_orders).pack(pady=5)
        tk.Button(self.root, text="Logout", width=30,
command=self.main_menu).pack(pady=5)

    def ticket_store(self):
        self.clear()
        tk.Label(self.root, text="Available Tickets", font=("Arial",
16)).pack(pady=10)
        tk.Label(self.root, text="Race Date (YYYY-MM-DD)").pack()
        race_date_entry = tk.Entry(self.root)
        race_date_entry.pack()

        payment_var = tk.StringVar(value="Credit Card")
        tk.Label(self.root, text="Select Payment Method").pack()
        for method in ["Credit Card", "PayPal", "Digital Wallet"]:
            tk.Radiobutton(self.root, text=method, variable=payment_var,
value=method).pack(anchor="w")

        def purchase(ticket_key):
            race_date = race_date_entry.get()
            if not race_date:
                messagebox.showerror("Error", "Race date is required")
                return
            ticket = tickets[ticket_key]
            new_order = Order(ticket.ticket_type,
datetime.now().strftime('%Y-%m-%d'), payment_var.get(), race_date)
            self.current_user.orders.append(new_order)
            orders.setdefault(self.current_user.username,
[]).append(new_order)
            save_file(users, 'users.pkl')
            save_file(orders, 'orders.pkl')
            messagebox.showinfo("Success", "Ticket purchased")
            self.user_dashboard()

        for key, t in tickets.items():
            label = f"{t.ticket_type}: ${t.price} | {t.validity} |
{t.features}"
            tk.Button(self.root, text=label, command=lambda k=key:
purchase(k)).pack(pady=3)
```

```
        tk.Button(self.root, text="Back",
command=self.user_dashboard).pack(pady=10)

    def view_orders(self):
        self.clear()
        tk.Label(self.root, text="My Orders", font=("Arial",
16)).pack(pady=10)
        if not self.current_user.orders:
            tk.Label(self.root, text="No orders found.").pack()
        else:
            for order in self.current_user.orders:
                info = f"{order.ticket} | {order.race_date} |
{order.payment_method} (Ordered: {order.date})"
                tk.Label(self.root, text=info).pack()
        tk.Button(self.root, text="Back",
command=self.user_dashboard).pack(pady=10)

    def clear(self):
        for widget in self.root.winfo_children():
            widget.destroy()

root = tk.Tk()
app = TicketApp(root)
root.mainloop()
```

File Structure Explanation (Part 2)

The system uses the `pickle` library to store data persistently in binary format. The following files are created and managed during program execution:

1. users.pkl

- Stores all registered user accounts.
- Each `User` object includes:
    - `username`, `password`, `name`, `email`, and a list of `Order` objects representing purchased tickets.

So this file ensures that all user information is retained between sessions and can be retrieved efficiently whenever the user logs in or interacts with the system.

2. orders.pkl

- Stores ticket orders placed by users.
- Each entry is linked to a user by username and contains:
    - `ticket type`, `purchase date`, `payment method`, and `race date`.

It acts as a transaction log for all ticket-related actions, helping in order tracking and generating reports if needed.

File Handling Logic

- On startup, the program loads existing data using `load_file()`.
- Whenever a user registers or buys a ticket, the corresponding file is updated using `save_file()`.
- Each file is stored separately for modularity and easier data tracking.

This separation also allows developers to debug and maintain individual components of the system more effectively without risking data integrity of other parts.

Assurance

- The system checks if each `.pkl` file exists using `os.path.exists()`.
- If a file does not exist, it initializes with an empty dictionary to ensure smooth first-time usage.
- This protects the program from crashing due to missing files.

It also provides a fallback mechanism, allowing the program to remain functional even in unexpected environments or after accidental file deletions.

Github repository link: https://github.com/NasserLootah/Nasser-and-Mohammed/tree/main