Patrick Simpauco

Jenny Tran

Johnny Rosas

CS 250

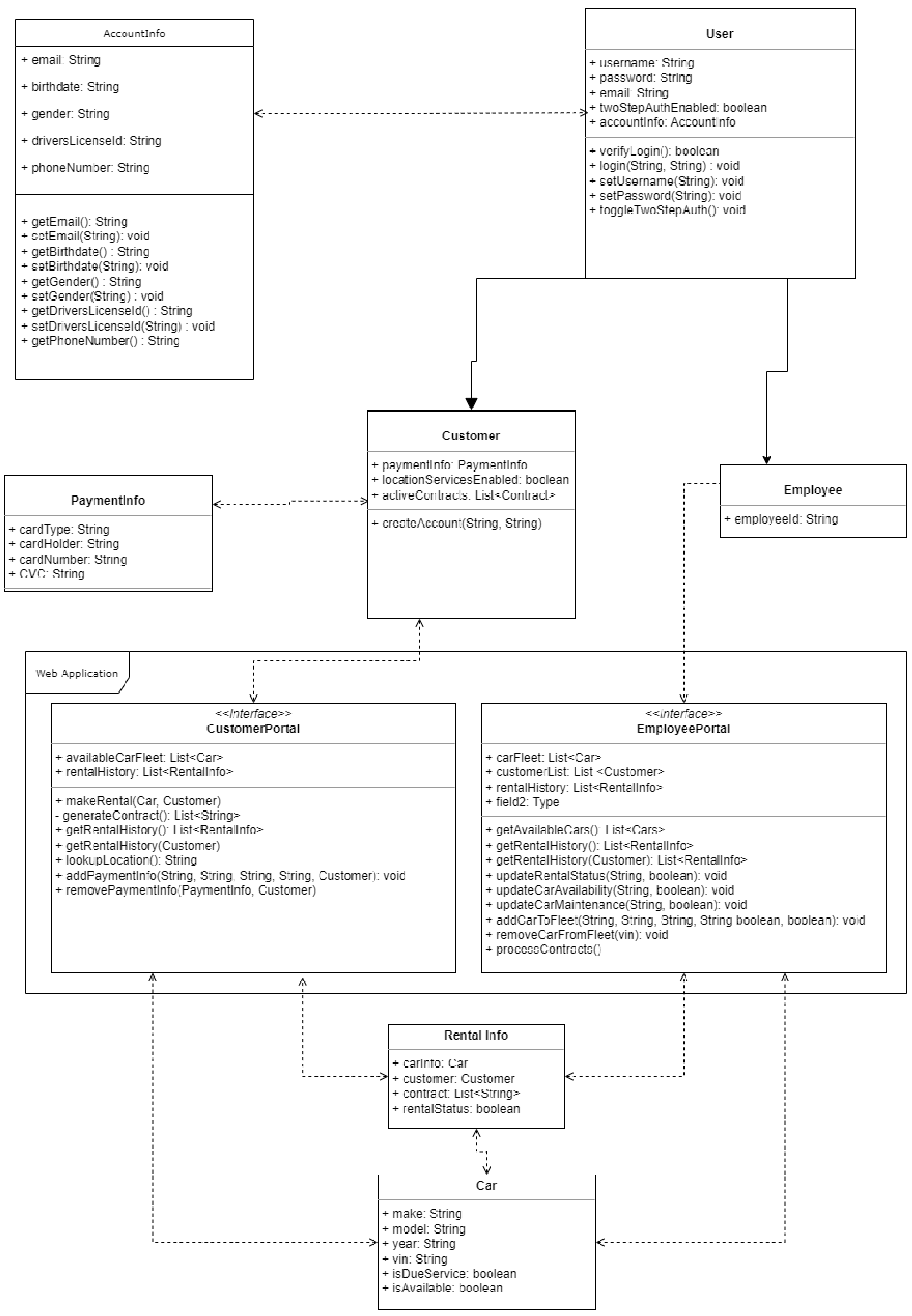
Professor Donyanavard

October 27, 2023

UML Diagram Changes

- Change params in addCarToFleet() method from Car object to (String, String, String, boolean, boolean)
    - Input should be the fields necessary to create a car object, not the car object itself
- Removing the setEmail() method within the User class
- Added vin field to Car class
    - Changes reflected to the parameters for the following methods within EmployeePortal Class:
        - updateRentalStatus()
        - updateCarMaintenance()
        - addCarToFleet()
        - removeCarFromFleet()
- CustomerPortal class field name change from carFleet to availableCarFleet
    - Customers should only be able to view the information of cars that are available

Updated diagram:


AccountInfo
+ email: String
+ birthdate: String
+ gender: String
+ driversLicenseId: String
+ phoneNumber: String
+ getEmail(): String
+ setEmail(String): void
+ getBirthdate() : String
+ setBirthdate(String): void
+ getGender() : String
+ setGender(String) : void
+ getDriversLicenseId() : String
+ setDriversLicenseId(String) : void
+ getPhoneNumber() : String
User
+ username: String
+ password: String
+ email: String
+ twoStepAuthEnabled: boolean
+ accountInfo: AccountInfo
+ verifyLogin(): boolean
+ login(String, String) : void
+ setUsername(String): void
+ setPassword(String): void
+ toggleTwoStepAuth(): void
Customer
+ paymentInfo: PaymentInfo
+ locationServicesEnabled: boolean
+ activeContracts: List<Contract>
+ createAccount(String, String)
PaymentInfo
+ cardType: String
+ cardHolder: String
+ cardNumber: String
+ CVC: String
Employee
+ employeeId: String
Web Application
<<Interface>>
CustomerPortal
+ availableCarFleet: List<Car>
+ rentalHistory: List<RentalInfo>
+ makeRental(Car, Customer)
- generateContract(): List<String>
+ getRentalHistory(): List<RentalInfo>
+ getRentalHistory(Customer)
+ lookupLocation(): String
+ addPaymentInfo(String, String, String, String, Customer): void
+ removePaymentInfo(PaymentInfo, Customer)
<<Interface>>
EmployeePortal
+ carFleet: List<Car>
+ customerList: List <Customer>
+ rentalHistory: List<RentalInfo>
+ field2: Type
+ getAvailableCars(): List<Cars>
+ getRentalHistory(): List<RentalInfo>
+ getRentalHistory(Customer): List<RentalInfo>
+ updateRentalStatus(String, boolean): void
+ updateCarAvailability(String, boolean): void
+ updateCarMaintenance(String, boolean): void
+ addCarToFleet(String, String, String, String boolean, boolean): void
+ removeCarFromFleet(vin): void
+ processContracts()
Rental Info
+ carInfo: Car
+ customer: Customer
+ contract: List<String>
+ rentalStatus: boolean
Car
+ make: String
+ model: String
+ year: String
+ vin: String
+ isDueService: boolean
+ isAvailable: boolean

Unit Tests

1. **Method setEmail() in Account Info class**
   - Testing method: Partition
   - Inputs: Valid email (exists and can handle messages), invalid email (“nonexistent@null.com”), random string (“test”), numbers (“12345”)
   - Expected Output: The inputs can be divided into two categories: valid and invalid inputs. In all cases of invalid inputs, the system should notify the user that their email is not valid and prompt them to try again with a different email. This should be the case for the invalid email, random string, and numerical input. When a valid email is entered, the user will be able to confirm the change and the information should be reflected in the user’s profile under their account information. Furthermore, the user should be receiving emails from the system at the provided email address. This can be verified with a “test” account creation and sending a notification to that email.
   - Test reasoning: Within the setEmail() method, there should be logic to verify that the input for the email is valid and exists. For example, the email “nonexistent@null.com” may follow the general format of an email, but it isn’t an email that one would be able to send and receive messages from. This is the same case for the random string and numerical inputs, as it would result in the removal of email as a form of communication with users if the field is changed.
   - Coverage: This test set covers the expected successes and failures that a user might encounter when they want to change their email address. In the instance that the user incorrectly types in their email address, they should not be able to confirm the change to their account information. An email change can be confirmed when an existing email is entered. As previously mentioned, without this logic in place, it cuts off email as a form of communication with the user.

2. **Method setBirthdate() in accountInfo class**

   - Testing method: partition
   - Inputs:
     - Valid input
     - Invalid inputs including:
       - Nonexistent date (2/29)
       - Date that has not happened yet (e.g. year 2033)
       - Birthdates that are less than 16 years old
       - Wrong data type
       - Not a date (e.g. apples)
   - Expected Output:
     - Valid inputs: successfully set birthdate to inputted date and continue signing up for new account or saving the changes that they’ve made

        - Invalid inputs: for all invalid cases, the system should display an error message saying invalid date, please try again and make the user to input another, valid date
    - Test Reasoning: this test is made with the consideration of common mistakes usually happens when a user enters their birthdate. The input sets are made to make sure that the system detects and handle correctly to both valid and invalid input.
    - Coverage: This test covers the process of a user account (both employee and customer) setting up their birthdate for when they first sign up or making changes to the date for existing accounts.

Integrations Tests

1. **Method addCartoFleet(String, String, String, String, boolean, boolean) in EmployeePortal class**
    - Testing method: Partition
    - Inputs:
        - Valid input:
            - (alphabetical string, alphabetical string, numeric string, alphanumeric string, false, false)
            - (alphabetical string, alphabetical string, numeric string, alphanumeric string, true, false)
            - (alphabetical string, alphabetical string, numeric string, alphanumeric string, false, true)
        - Invalid Input:
            - Any string that doesn't conform to the template above and the boolean input: (true, true)
    - Expected Output:
        - From valid input: Car object is created and added to carFleet list in the EmployeePortal class. Depending on the value of the object's isAvailable field, if true, the object is added to the CustomerPortal availableCarFleet list.
        - From invalid input: Message, "Invalid input, please re enter (incorrectly entered field)"
            - Car is not added to any list, object is not created
    - Test Reasoning: The fields "make" and "model" should only be populated with alphabetical characters because they can't be defined by numerical or special characters. The same logic for only allowing numeric strings is true for the "year" field. For the boolean fields "isDueService" and "isAvailable", a car requiring service shouldn't be available to rent. Therefore, a Car object can't be created with both fields being true. It only makes sense for a car to be added to either the availableCarFleet list or the carFleet list if both booleans are false or only one is true.

- Coverage: This test will cover the case where an employee would like to add a car into the car fleet. This test ensures that the entries are not limited by make, model, or year as well as cars initially being unavailable for purchase. Available cars will be up for purchase and viewable in the availableCarFleet list within the CustomerPortal

2. **updateCarAvailability(String, boolean) in EmployeePortal (interacts w/Car class)**
    - Testing method: Partition
    - Inputs: (valid VIN, true), (valid VIN, false), (invalid VIN, true), & (invalid VIN, false)
    - Expected Output: For the inputs where the VIN is valid, the boolean field isAvailable should be set to the value of the argument passed. For the inputs with the invalid VIN, the system should notify the employee that the VIN entered is invalid and to try again. The isAvailable field should not be changed when the method call passes an invalid VIN.
      When the VIN input is valid and the boolean parameter is false, the updates should be reflected by removing the Car from the availableCarFleet list in the CustomerPortal Class. In the EmployeePortal Class, the isAvailable field of the Car object should be updated, but the Car object should remain in the list.
    - Test reasoning: The method is meant to change the boolean field isAvailable in a Car object from the EmployeePortal Class. The proper car can be identified from searching for the car's VIN from the EmployeePortal class's carFleet list. Once the corresponding car is identified, the Car's isAvailable field is adjusted appropriately. It's impossible to adjust the availability of a Car object that doesn't exist within the system. This explains why the employee is prompted to enter a valid VIN if the entered one doesn't exist.
      User's should only have the ability to view the information of cars that are available, explaining why the Car object is removed from the availableCarFleet list when the availability of the car is set to false. The same isn't true for the EmployeePortal's carFleet list because employees have the ability to view all cars within the system.
    - Coverage: This test set cover's the possibilities when a car may not be available to rent. While the particular reason can vary, the importance lies in the user not being able to rent an unrentable car. This isAvailable boolean field should be -adjusted simultaneously with the other boolean fields in a Car object. For example, if a car requires maintenance or is rented, it will be unavailable to rent.

System Tests

1. Rental functionality

- Testing Method: Partition
- Inputs:
    - Customer login, car selection, payment method linking
        - Scenario 1: all valid inputs
        - Scenario 1: invalid inputs
- Expected Output:
    - Valid inputs:

        - Successful rental process with valid inputs
    - Invalid inputs for:
        - Customer login (not logged in): redirect user to login/sign up page
        - Car selection:
            - Invalid or unavailable car selections: display error message
            - Has an active rental contract: don't allow customer to rent another car and display error message
        - Payment method linking:
            - Not linked: add payment method window pop up
            - Invalid method: Error message, display option for linking a different method
- Test Reasoning: This test validates the rental process, ensuring that users can successfully rent a car with valid inputs. It also checks how the system handles errors and links payment methods securely.
- Coverage: The test covers the entire rental process from the user's point of view, including user login, car selection and payment, checking for both successful and unsuccessful scenarios

2. **Employee data access and management**
    - Testing method: Partition
    - Inputs:
        - Employee login, review rental status and contract details, update customer information, update car information
            - Scenario 1: all valid inputs/actions
            - Scenario 2: invalid inputs/actions
    - Expected Output:
        - Valid inputs/actions:
            - Successful access to employee portal upon logging in.
            - Successful view of rental, contract, customers and car information
            - Successful modification of customers and cars information
        - Invalid inputs/actions:
            - Employee login
                - Not logged in: redirect employee to log in page
                - Invalid login information: display error and retry message
                - Ex-employee login in: display error message
            - View information:
                - Invalid/nonexisting information for customer/car/contract/rental detail: display error message/redirect to error page
            - Update/modify information:
                - Try to update invalid data type: display error and retry message
                - Specific to Customer:

        - Update a customer information that employees don’t have the authority to: display error message (e.g. “Don’t have the authority to change user’s ____”)
        -
    - Car:
        - Removing cars from fleet that are not allowed to removed: display error message
    - Contracts:
        - Process contract that customers hasn’t submit: display error message
- Test Reasoning: This test verifies that employees can access the employee portal successfully with the valid login information, manage customer data, including searching for profiles, reviewing rental status and contract details, updating customer information and managing cars information. This test checks how the system handles invalid inputs and errors related to employee access and make sure the system reacts accordingly.
- Coverage: The test covers the entire management function of a typical employee’s, including employee login, managing contracts, rentals, customers and cars information, checking for both successful and unsuccessful scenarios.