**PRACTICAL EXAMINATION**

**Subject code: PRN212 - SU25**  
**Duration: 85 minutes**

**Question 1 (3.5 marks):**

**1. Create a Console Project with name VehicleRentalApp (0.5 mark)**

**2. Create a Generic Class (0.5 mark):**

* Create a generic class called **RentalRecord<T>** that can store rental-related data of any type.
* The class should have:
  + A property **Records** of type **List<T>** to store a collection of rental records.
  + A method **AddRecord(T record)** to add a record to the list.
  + A method **GetAllRecords()** to return an **IEnumerable<T>** containing all records.
  + A method **RemoveRecord(T record)** to remove a specific record from the list.
  + A method **GetRecordCount()** to return the total number of records.

**3. Create a Vehicle class (1.0 mark):**

* The **Vehicle** class should have properties: **VehicleID** (int), **Brand** (string), **Model** (string), **Year** (int), **RentalPricePerDay** (decimal).
* **GetVehicleAge():** Returns vehicle age as int (current year - year).
* Override the **ToString()** method to display vehicle information in the format:  
  **"VehicleID - Brand Model (Year) - $RentalPricePerDay/day"**

**4. Use the Generic Class (1.5 mark):**

In the Main function, you must implement the following tasks:

* Create an instance of **RentalRecord<string>** and add 3 rental status descriptions using **AddRecord()** method.
* Display all statuses and the total count using **GetAllRecords()** and **GetRecordCount()** methods.
* Create another instance of **RentalRecord<Vehicle>** and add 4 Vehicle objects using **AddRecord()** method.
* Use **LINQ** to filter and display vehicles with **rental price greater than $40**.
* Display vehicles grouped by **vehicle age** using LINQ GroupBy (group by: "New" for age ≤ 3, "Used" for age > 3).

**Question 2 (6.5 marks):**

**EasyCar Rental** wants to develop a vehicle rental management system to track information about vehicles, customers, and rental bookings. The specific requirements are as follows:

**1. Create a Library Project named EasyCarDataAccess within a solution named EasyCarRental (1.0 mark):**

Build an application using Entity Framework Core. Create a database using either the **Code First or Database First** approach with the following classes and their relationships:

* **Customer.cs** (contains: CustomerID, CustomerCode, FullName, Email, Password)
* **Vehicle.cs** (contains: VehicleID, LicensePlate, Brand, Model, RentalPricePerDay)
* **RentalBooking.cs** (contains: BookingID, CustomerID, VehicleID, StartDate, EndDate, Status)

**Notes:** Booking statuses include: "Active", "Completed" and "Cancelled"

**Relationships:**

* One **Customer** can have multiple **RentalBookings** (One-to-Many).
* One **Vehicle** can have multiple **RentalBookings** (One-to-Many).

**2. Create database DBEasyCar and Build relational tables (0.5 mark)**

**Notes:** The tables must be related as shown in the requirements above, and these relationships are automatically generated through Code First or Database First.

**3. Create a Library project named EasyCarBusiness (0.5 Mark)**

**Notes:** EasyCarDataAccess should be containing any EF, DBContext, DAL, Model classes. EasyCarBusiness should be containing all application business logics.

**4. Create a WPF application named EasyCarApp (4.5 Mark)**

**Implement Customer Login System (1.0 mark)**

* **Create a login form** where customers enter their **CustomerCode** and **Password**.
* **Validate credentials** by checking the database.
  + If login is **successful**, redirect to the **MainWindow Form**.
  + If login **fails**, show error message: **"Invalid CustomerCode or Password"**.

**MainWindow with Menu (0.5 mark)**

* After successful login, navigate to **MainWindow** containing menu with:
  + Vehicle Management
  + My Bookings

**Vehicle Management (1 mark)**

* **Display all vehicles** with their basic information (License Plate, Brand, Model, Price/Day).
* **Add new vehicles** to the system.
* **Edit vehicle information**.

**My Bookings (1.5 marks)**

* **Display customer's bookings** with:
  + Vehicle information (License Plate, Brand, Model)
  + Start Date, End Date
  + Status (Active, Completed, Cancelled)
* **Allow customers to update** booking status (Complete or Cancel it).
* **Sort bookings** ascending by start date.
* **Create new bookings** for vehicles.

**Logout Functionality (0.5 mark)**

* **Logout button** that returns user to login screen.

**Important Notes:**

**Functionality Requirement:** Your submission must be fully functional. Projects that fail to compile or run will receive a score of zero. Ensure all features are correctly implemented and thoroughly tested before submission.

**Submission Requirements:**

* Submit all project files and **backup of the SQL Server database script** (\*.sql) – if you use DatabaseFirst approach.
* Ensure that no **personal identifiers** (name or student ID) are included in the submission.

**Additional Instructions:**

* You may use **local resources**, but internet access is restricted to **downloading the exam, submitting your work, and installing packages via NuGet**.
* Fully implemented according to software architecture and design principles, transition from tight coupling to loose coupling such as **BusinessLogic, DataAccess**.