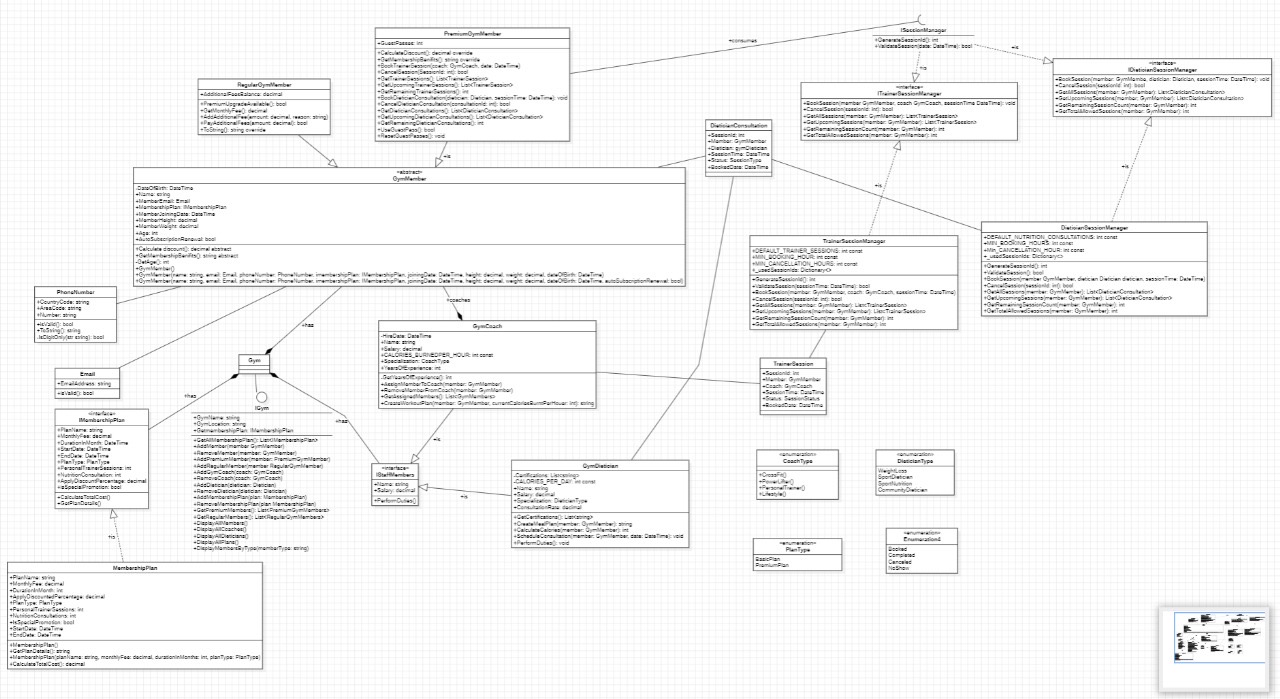
|  |  |
| --- | --- |
| cid:3461564935_486740 | Faculty of Natural and Applied Sciences  Computer Science Department |

|  |
| --- |
| **CSC 323: Object-Oriented Design**  **Project – Report** |
| Submitted by:   |  |  |  | | --- | --- | --- | | Student ID | Name | Major | | 20231256 | **Christy Khalifé** | **Computer Science** | | 20241747 | **Kristen Kamouh** | **Computer Science** | |
|  |

**Project Description** (around 500 words)

|  |
| --- |
| It is a highly flexible and powerful software program which simplifies gym management, improves the user experience, and increases the efficiency of scheduling and member management. The system emulates the actual-life gym setup such that members are allowed to subscribe to any number of members' schemes, book schedules for workout as well as appointments with a dietician, and communicate with trainers as well as nutritionists. Multiple gym member types (regular and premium) are allowed, and the facilities and benefits are customized accordingly. Basic Object-Oriented programming concepts like abstraction, inheritance, polymorphism, encapsulation, and interfaces are used while system design.  At the core of the system is the abstract base class GymMember, which has two specialized subclasses, namely RegularGymMember and PremiumGymMember. Derived classes, through overriding, alter core methods such as CalculateDiscount and GetMembershipBenefits, hence enabling the system to provide differential services to the two member types. For example, the premium members are permitted separate sessions with the coach as well as sessions with the dietitians, which the regular members are not.  It holds all the gym information of the central hub, i.e., the gym name, the address, as well as the members, staff, and membership it holds. It tightly associates with other classes like the GymCoach, the GymDietician, and the MembershipPlan. It holds the ability of the following CRUD operations: add/delete members, add staff, add sessions, as well as print data. In such a manner, the gym management gets a real-time perception of all the activities and the services. AllMembershipPlans have significant properties such as name, price, term, benefits, and allowances (consultation/sessions, etc.). Members are automatically enrolled to a such a plan when becoming members. An enumeration named PlanType categorizes differences among such types as Basic, Standard, and Premium and serves to enforce logic about the plan elsewhere in the system.  The framework enables the use of advanced session management with TrainerSessionManager and DieticianSessionManager, which are implementations of the ISessionManager and the IDieticianSessionManager interfaces, respectively. These interfaces establish a contract for the creation, cancelation, and acquiring of sessions. Sessions themselves (TrainerSession and DieticianConsultation) are represented as objects containing information such as session times, a status, and attendees. They are instantiated and controlled programmatically depending upon member requisitions and the availability of the staff.  Support classes such as PhoneNumber and Email neatly and safely save and correctly validate member contact information. Staff-role-related abilities, like their sessions, schedules, and certifications, are encapsulated into the classes GymDietician and GymCoach. Coaches are specialized down further by the CoachType, an enumeration which categorizes the coaching as a personalized, strength, and cardio coaching.  It uses composition (e.g., a gym has many members), inheritance (e.g., regular/paid members), and interfaces (e.g., session operations, staff activities) that are a reflection of a pure, extensible design. Polymorphism is provided, thus the freedom of calling the overriding methods like CalculateTotalCost() or BookSession() according to the actual type of the object at runtime.  In short, the Gym Management System is a well-structured, scalable program deserving of actual use at a gym. It puts the easy work into taking care of members, trainers, dietitians, and sessions, yet the system itself remains comprehensible, reusable, and extensible through appropriate use of OOP techniques. |

**Class Diagram (UML)**

Figure 1. Gym Management System (GMS) Class Diagram

**Source Code (C#)**



Figure 2. CoachType.cs - Enum



Figure 3. Gym.cs – Abstract class

****

Figure 4. IstaffMember.cs - Interface

**Descriptions of how and where you have used key concepts**

|  |  |
| --- | --- |
| **Concept** | **Description** (around 30 words each) refer to the class diagram and source code figures as you see fit |
| Abstraction (abstract classes and/or interfaces) | GymMember is an abstract class providing a common template for all member types. IMembershipPlan interface defines contract requirements without implementation details, allowing for different plan implementations. |
| Concrete classes | RegularGymMembers and PremiumGymMembers are concret implementations of the abstract GymMember class, providing specific behaviors like session booking and fee calculation for different membership types. |
| Namespaces | The project uses namespaces like CSC323FinalProject.Models and CSC323FinalProject.Services to organize related classes, preventing naming conflicts and creating logical groupings of functionality. |
| Methods | Methods like BookSession, CancelSession, and GetRemainingSessionCount encapsulate behaviors, hiding implementation details while providing clear interfaces for client code to interact with objects. |
| Properties | Properties like MembershipPlan, PersonalTrainerSessions, and AdditionalFeesBalance encapsulate data with controlled access, enabling validation logic like preventing negative values for fees or sessions. |
| Overriding of methods and/or properties | CalculateDiscount() is overridden in RegularGymMembersand PremiumGymMembers to provide specific discount calculations (5% vs. 15%) based on membership type. |
| Constructor overloading | GymMember class offers multiple constructors with different parameter sets, allowing objects to be created with varying initial states, such as with or without auto-renewal status. |
| Enumeration (Enum) | PlanType enum (BasicAccess, PremiumAccess) provides type-safe constants representing membership levels, improving code readability and preventing invalid assignments when setting plan types. |
| Collections (e.g., Lists) | List<TrainerSession> stores and manages training sessions, enabling operations like finding upcoming sessions, counting used sessions, and validating bookings against coach availability. |
| Loops (e.g., foreach) | Foreach loops iterate through member collections to find members by name, calculate remaining sessions, and validate scheduling conflicts when booking new sessions. |

|  |  |
| --- | --- |
| **SOLID Principle** | **Description of where the principle is realized and for what purpose** (around 50 Words) |
| 1) Single Responsibility | The project separates concerns by having dedicated classes for specific functionalities. TrainerSessionManager handles only session booking logic, MemberManagementService focuses solely on member operations, and MembershipPlan deals exclusively with plan details. This separation ensures each class has exactly one reason to change, making maintenance easier and reducing side effects. |
| 2) Open-Closed | The gym system is designed to be open for extension but closed for modification. New member types can be added by extending the GymMember class without modifying existing code. Similarly, the enum-based PlanType system allows adding new membership tiers without changing how plans are processed, demonstrating extensibility without disrupting existing functionality. |
| 3) Liskov Substitution | Both RegularGymMembers and PremiumGymMembers can be used anywhere their parent GymMember type is expected. This is demonstrated in member management code where \_gym.GetRegularMembers().Cast<GymMember>().Concat(\_gym.GetPremiumMembers()) treats all members uniformly despite their different implementations, ensuring type hierarchies maintain consistent behavior. |
| 4) Interface Segregation | The project uses focused interfaces like IMembershipPlan that contain only methods relevant to their clients. Instead of one large interface, specialized interfaces ensure implementing classes aren't forced to provide methods they don't need. This creates more maintainable, cohesive components with clearer dependencies and reduces coupling between system parts. |
| 5) Dependency Inversion | High-level modules like SessionManagementService depend on abstractions (interfaces or abstract classes) rather than concrete implementations. The \_gym parameter is passed to constructors rather than created internally, allowing different gym implementations to be injected. This facilitates testing, enables configuration changes without code modifications, and decouples system components. |

|  |  |
| --- | --- |
| **GOF Design Pattern** | **Description of where the pattern is realized and for what purpose** (around 50 Words) |
| 1) Factory Method | The system implements Factory Method pattern in membership creation, where MemberManagementService acts as the creator that instantiates different types of members (Regular or Premium) based on user input. This pattern centralizes object creation logic, making it easier to maintain and extend by adding new member types without modifying client code that uses these objects. |
| 2) Singleton | The Gym class appears to follow a Singleton-like pattern, ensuring only one instance manages all gym operations throughout the application. Services like MemberManagementService and SessionManagementService receive this shared instance, allowing centralized access to members, coaches, and other resources while maintaining consistent state across the system. |
| 3) Strategy Pattern | The discount calculation system demonstrates the Strategy pattern. Different member types (Regular vs Premium) implement different discount calculation strategies (5% vs 15%). The abstract GymMember class provides the context for these strategies, allowing client code to use a uniform interface while different concrete classes execute their specific algorithms. |

**User Interface (Windows Forms)**

A screenshot of a computer

AI-generated content may be incorrect.

Figure n. User Interface caption

|  |
| --- |
| **Brief Description of UI** (around 50 words) |
| This user interface is a Gym Management System designed with categorized functionality. It allows admins to manage members, coaches, and dieticians. Premium members can book or cancel training and dietician sessions, while regular members can pay additional fees or upgrade. The layout is clean, organized, and clearly divided by access type. |