# 1. Overview

The system consists of:

* **User Hierarchy**: Abstract User class with concrete subclasses for different user types.
* **Recommendation Hierarchy**: Abstract Recommendation class with concrete subments (DietPlan, ExercisePlan, Meal).
* **Factories**: Interfaces (IFactory, Factory) and concrete implementations (ConcreteFactory) to instantiate objects.

# 2. Class Responsibilities

### 2.1 User Hierarchy (Abstract & Subclasses)

User**(Abstract Class)**

* **Responsibility**: Base class storing common user attributes.
* **Attributes**:
  + Name, Age, Gender, Height, Weight
* **Methods**:
  + CalculateBMI(): float – Computes BMI (Weight / (Height^2)).

### 2.2 Recommendation Hierarchy (Abstract & Subclasses)

Recommendation**(Abstract Class)**

* **Responsibility**: Base class for diet/workout plans.
* **Attributes**:
  + PlanName: String
* **Methods**:
  + GeneratePlan(): void – Abstract method to create a plan.

### 2.3 Factory Components (Factory Method Pattern)

**Interfaces**

1. IFactory
   * **Method**: CreateProfile(): User  
     Instantiates a User subclass based on input

(e.g., "Beginner"→ Beginner object).

1. Factory (Interface for Recommendations)
   * **Method**: createRecommendation(user: User): Recommendation  
     Creates a DietPlan or ExercisePlan tailored to the user.

**Concrete Implementations**

* ConcreteFactory  
  Implements both interfaces to:
  + Create User profiles (e.g., BusyUser, FitnessUser).
  + Generate Recommendation objects (e.g., DietPlan for WeightMgmtUser).

# 3. Relationships Between Classes

1. **Inheritance (Generalization)**:
   * User ← Beginner, WeightMgmtUser, etc. (subclasses extend User).
   * Recommendation ← DietPlan, ExercisePlan, Meal.
2. **Factory Method Pattern**:
   * IFactory/Factory interfaces define creation methods.
   * ConcreteFactory implements them to decouple object creation from usage.
3. **Dependency**:
   * DietPlan depends on User to calculate needs (CalculateDailyNeeds(user)).
   * ConcreteFactory depends on User and Recommendation hierarchies.
4. **Association**:
   * Meal is associated with DietPlan (e.g., a DietPlan contains multiple Meal objects).

# 4. Key Dependencies

The Recommendation class and its subclasses rely on the User**object** to:

* Access user-specific attributes (e.g., Age, Weight, FitnessLevel).
* Call methods (e.g., CalculateBMI()) for calculations.
* Determine which type of plan to generate (e.g., DietPlan for WeightMgmtUser).

# 5. How User Attributes Drive Recommendations

**5.1 ExercisePlan Class**

**Method: SuggestExercises() → List<String>  
Logic:**

* **Checks user subclass to customize workouts:**
  + **Beginner: Low-intensity exercises (user.FitnessLevel = "low").**
  + **BusyUser: Short workouts (user.AvailableTimePerDay < 30 mins).**
  + **Elder: Balance-focused routines (user.MobilityLevel = "moderate").**

# 6. Factory Method’s Role in User-Specific Plans

The Factory**interface** ensures the correct Recommendation subclass is created for each User:

1. ConcreteFactory.createRecommendation(user) inspects the User object:
   * If user is WeightMgmtUser → Returns DietPlan.
   * If user is FitnessUser → Returns ExercisePlan with TrainingTypes.
2. The generated plan **automatically adapts** to the user’s attributes.

# 7. Example Scenarios

|  |  |  |
| --- | --- | --- |
| User Type | Recommendation Generated | Key Logic |
| WeightMgmtUser (TargetWeight=70kg, DietType="keto") | DietPlan with TargetCalories=1800, DietType="keto" | Uses user.Weight and DietType. |
| Beginner (FitnessLevel="low") | ExercisePlan with IntensityLevel="low", DurationPerSession=20 mins | Adjusts for fitness newbies. |
| Elder (MobilityLevel="low") | ExercisePlan with chair yoga, hydration reminders | Prioritizes safety. |