OOP Modeling Project: Designing a System from Scratch

Overview

In this project, you'll work with your group to design the architecture of an object-oriented system based on a real-world scenario. You **won't write code**. Your goal is to plan and explain how the system could be structured using object-oriented principles.

You'll apply what you've learned about:

- Designing well-named and focused classes
- Using inheritance and composition (has-A) appropriately
- Applying abstraction and polymorphism
- Making design decisions with clear justification

Learning Objectives

By the end of this project, you should be able to:

- Identify and justify the set of classes needed for a project
- Assign properties and methods that reflect clear responsibilities
- Distinguish between inheritance (is-a) and composition (has-a)
- Determine which classes might be abstract and why
- Anticipate where polymorphism and method overriding might apply

---

Project Steps & Checkpoints

1. Choose a Scenario

Pick one of the project scenarios. Each includes a list of required features to guide your thinking. Try not to go beyond the scope.

---

2. Brainstorm Classes (Checkpoint 1)

As a group, list out potential classes. Focus on **nouns** in the scenario. If you want to start off with a UML diagram, that is fine, just keep in mind you'll make changes to it. For each class, think about:

- What does this class represent?
- What information should it store?
- What actions or behaviors might it need?

Instructor Checkpoint 1:
Once you have a rough list of classes and short descriptions of what each one represents, check in with your instructor to finalize the list before moving forward.

---

3. Assign Responsibilities

Now that you've finalized your class list, define for each class:

- What does it **know**? (properties/fields)
- What does it **do**? (methods)
- Is there any **overlap** between classes that needs to be resolved?
- Are there any missing responsibilities that another class shouldn't have?

Be specific. "Does something" is too vague—ask what it does *specifically*, even in plain English.

Don't worry about the "HOW" yet. How it gets done is what the junior and mid-level developers will figure out later.

---

4. Identify Relationships (Checkpoint 2)

Now focus on how the classes relate to one another:

- Which classes **inherit** from others (is-a)?
- Which classes **use or contain** other classes (has-a)?

- Are there any many-to-one or many-to-many relationships?
- Are any of your original class responsibilities now redundant or misplaced?

Instructor Checkpoint 2:
Before continuing, check in again with your instructor. You'll review the class relationships and get feedback on whether inheritance and composition are being used correctly. It's okay if your classes changed, were removed, or new ones were added.

---

5. Abstract vs. Concrete

Now determine:

- Which classes should never be instantiated on their own (abstract)?
- Are there behaviors (methods) that should be shared across all child classes?
- Are there methods that should be defined differently in each subclass?

This is your opportunity to consider **polymorphism**. Ask:

- Are there any methods that will need to be **overridden**?
- Can you imagine calling a method on a parent class reference but having different child behavior?

Examples could include: makeSound(), calculateFee(), applyEffect(), display() — general actions that would be common across multiple types.

---

6. Final Presentation

Prepare to present your model in a readable format:

- A rough diagram of your class structure (class names and relationships)
- Notes that explain why you designed it this way
- At least one example where polymorphism could be used

Pick 1 out of the following scenarios

Scenario 1: Pet Adoption System

Overview: You're building a system for a local pet shelter that keeps track of adoptable animals and the people who want to adopt them.

Requirements:

- The shelter houses different types of animals (dogs, cats, birds, etc.)
- Each animal has a name, age, species, breed, and adoption status.
- Some animals have special care instructions.
- People can apply to adopt a pet and must meet certain criteria.
- Staff members review adoption applications.

Design Hints:

- Not all animals behave the same way.
- What would make sense to share across all animals?
- What kind of object might represent an adoption process?

Scenario 2: Video Game Inventory System

Overview: Design the inventory system for a fantasy role-playing game.

Requirements:

- The player can carry items like weapons, potions, armor, and keys.
- Items have a name, description, and value.
- Weapons and armor may have different effects or stats.
- Potions can be consumed and have specific effects (like healing or invisibility).
- The inventory has a weight limit.

Design Hints:

- Not every item is used the same way.
- Think about what makes one item different from another.

- What kinds of things might be usable vs. wearable?

Scenario 3: Food Delivery App

Overview: Model a simplified food delivery system like Uber Eats or DoorDash.

Requirements:

- Users can browse restaurants and place orders.
- Each restaurant has a name, location, and menu.
- Menu items have a name, price, and category (entree, drink, dessert).
- Delivery drivers are assigned to orders and track delivery status.
- Orders can contain multiple menu items and belong to a specific user.

Design Hints:

- Think about who the system is for: customers, restaurants, and drivers.
- How would you represent a menu? An order?
- What's the relationship between orders and menu items?

Scenario 4: Fitness Tracker App

Overview:
You're designing the architecture for a fitness tracking app. The app allows users to log workouts, track goals, and monitor their progress.

Requirements:

- Users can log different types of workouts (running, swimming, weightlifting, etc.)
- Each workout has a date, duration, and type-specific information (e.g., distance, reps, laps).
- Users can set fitness goals (e.g., run 50 miles this month, do 10 workouts in a week).
- The system can calculate progress toward a goal.
- Some workouts are individual, others are group-based (e.g., group runs or classes).

Design Hints:

- Think about what all workouts have in common vs. what varies.
- Is there an opportunity for an abstract Workout class?
- Where might polymorphism show up when calculating or displaying workout details?

Scenario 5: School Course Catalog

Overview: You're designing a system to manage courses, instructors, and students for a school.

Requirements:

- Courses have a name, code, instructor, and a list of enrolled students.
- Students can register for multiple courses.
- Instructors can teach more than one course.
- Some courses are online; others are in-person.
- Courses have a maximum capacity.

Design Hints:

- What might make an online course different from an in-person one?
- Is there a relationship between users and roles like "student" and "instructor"?
- How should enrollment be tracked?

Scenario 6: Theme Park Ride Management

Overview: You're helping design a system to track rides, guests, and ride access at an amusement park.

Requirements:

- Rides have a name, type (rollercoaster, carousel, etc.), minimum height, and capacity.
- Guests have names, heights, and ticket types (single-day, season pass).

- Some rides have restrictions (e.g., height, health warnings).
- A ride can have a queue of waiting guests.
- Operators can view who is next in line and remove them when they ride.

Design Hints:

- How could different ride types be modeled?
- How would you track which guests are waiting for a ride?
- What kinds of responsibilities would a ride have?

Scenario 7: Music Festival Planner

Overview:
You're designing a system to help manage a multi-day music festival. The system tracks performances, stages, artists, and attendees.

Requirements:

- Artists perform on scheduled stages at specific times.
- Performances can be solo acts, bands, or guest collaborations.
- Stages have a name, capacity, and type (indoor, outdoor).
- Attendees can build a custom schedule of performances they want to see.
- Some performances have age restrictions or require special tickets.

Design Hints:

- What do all performances have in common? What varies?
- Think about relationships between artist(s), stage, and schedule.
- Would a generic Performance class make sense? If so, could it be abstract?
- What kind of behaviors might performances have that are different depending on the type?