Name: Gloire Boaz Mpoyi Mpoyi

Student ID: 0231990525

Project: Creative Habit Tracker App

Abstract

This app is for someone who wants to commit to hobbies or a craft whilst they continue to be accountable and organized to complete their tasks on time. The application is meant to help with organization and consistency by introducing a gamified approach to maintain engagement using a reward system and customizable avatars.

Project Overview

Through interaction of the habit tracker users stay accountable and organized in their creative endeavors, promoting consistency and completion of tasks through a gamified experience. Users can complete tasks, earn rewards, and customize their avatars as they achieve milestones, fostering long-term engagement with their hobbies or creative projects.

Project Goals

Task Management - Users should be able to create, update, and complete tasks to track their progress on creative projects.

Reward System - Implement a reward mechanism where users unlock items or customization options for their avatar upon completing tasks.

Avatar Customization - Allow users to personalize their avatar using earned rewards, enhancing motivation and engagement.

OOP Principles - Demonstrate core object-oriented programming concepts such as inheritance, polymorphism, and encapsulation to ensure modular and scalable design.

Object-oriented design

The main classes expected for my implementation - Task, User, Avatar, and Reward class.

Reward class – defines common properties and methods for rewards that can be granted to users. This class is abstract because the specific reward could vary for each task or avatar, but the concept remains the same.

Properties: rewardName, rewardType, effectOnAvatar.

Methods: unlockReward(), applyReward()

Subclasses:

1. **BadgeReward** - Represents a badge unlocked after completing specific tasks.
2. **AvatarAccessoryReward** - Represents accessories or items to customize the avatar.

Relationship Type:

Inheritance (**Reward** → **BadgeReward**, **Reward** → **AvatarAccessoryReward**)

Task class - Represents a task that the user needs to complete. This class extends the base class reward, so it is represented as a concrete class because specific instances of tasks are required.

Properties: taskName, description, deadline, isCompleted.

Methods: markAsComplete(), updateTaskDetails().

Subclasses:

1. **UserDefinedTask** (Abstract Subclass) - A template class allowing user-defined creative task categories.
2. **CreativeTask** (Concrete Subclass) - Extends **UserDefinedTask** and represents any task that relates to a user-chosen creative activity, such as painting, writing, sculpting, or photography.

Relationship Type:

Task → **UserDefinedTask** - Inheritance (Task is a base class for user-defined task templates).

UserDefinedTask → **CreativeTask** - Inheritance (Allows creative task customization).

CreativeTask → (Concrete User-Defined Subclasses) - Subtyping, enabling polymorphic behavior when working with different creative tasks.

User class - Represents the user of the program who has tasks to complete and an avatar to update based on rewards. This concrete class manages and avatar states

Properties: userName, taskList, completedTasks, avatar.

Methods: addTask(), completeTask(), viewProgress()

Avatar class - Represents the user's avatar that changes based on completed tasks. This concrete class modifies avatar properties as tasks are completed.

Properties: appearance, accessories, customizations.

Methods: updateAppearance(), addAccessory(), displayAvatar().

Subclasses:

1. **CustomizableAvatar** - Extends Avatar, adding more features for personalization.
2. **ThemedAvatar** – Extends Avatar, more specific customization

Relationship Type:

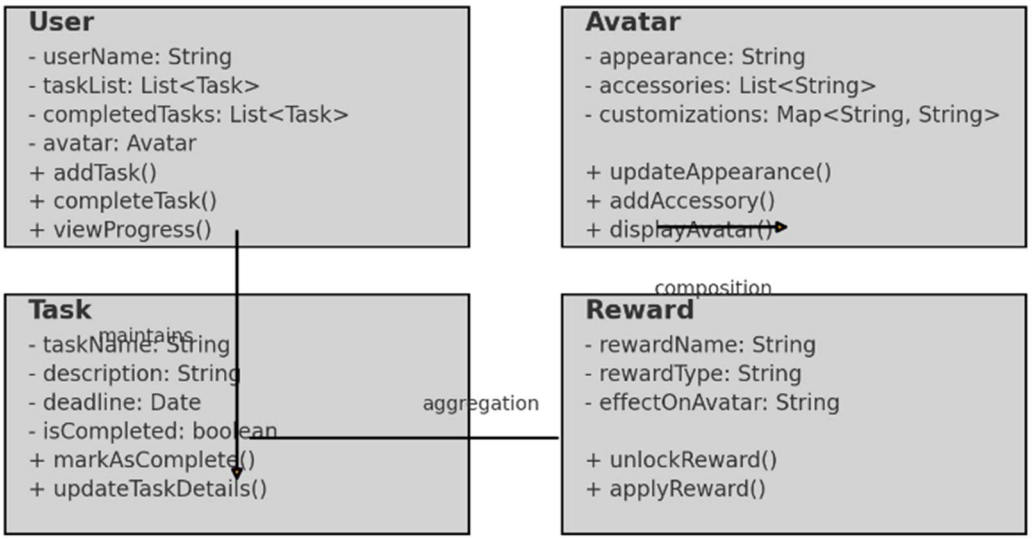
Inheritance (**Avatar → CustomizableAvatar**)

<u>Class Relationships</u>

User → Task - Aggregation — The User maintains a list of Task objects that can exist independently.
User → Reward - Aggregation — The User keeps track of Reward objects that are unlocked.
Task → Reward - Composition — Certain tasks have embedded reward logic, meaning Reward objects are tied to specific tasks.

Avatar → User - Composition — The Avatar is a part of the User and is dependent on it.
Reward → Avatar - Association — Rewards interact with the Avatar to update properties.

UML Class Diagram



User maintains a list of tasks and interacts with rewards.

Task and **Reward** have an aggregation relationship.

User has a composition relationship with **Avatar** as it forms part of the user's identity.

Polymorphism

The Task class can be extended to specialized types like TimedTask or PriorityTask, each with unique behaviors (e.g., different completion criteria or notifications).

The User class can handle lists of tasks and rewards polymorphically, allowing seamless iteration and method calls such as task.completeTask() or reward.applyReward().
The Avatar class can be extended to ThemedAvatar subclasses, where each subclass applies different customization rules (e.g., seasonal themes or artistic styles).

Implementing Polymorphism

To implement polymorphism effectively, the system will:

1. Use method overriding in subclasses to customize task completion logic or reward application. For instance, a completeTask() method in TimedTask could override the base Task class's implementation to add a timer check.
2. Ensure the User class interacts with a collection of Task objects that are of different derived types. This allows for methods such as displayAllTasks() to call the appropriate overridden method in each specific subclass.
3. Implement dynamic method dispatch, where references to the base class (Task, Reward, or Avatar) are used to invoke subclass-specific behavior at runtime. This maintains code simplicity while supporting complex, differentiated operations