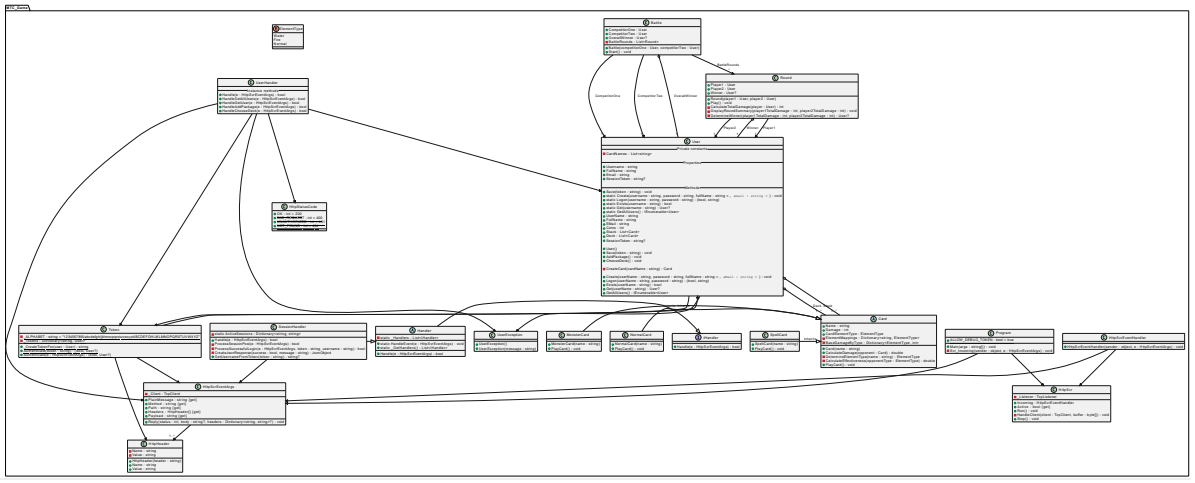
Protokoll zur Umsetzung des Klassendiagramms

1. Zielsetzung

Harmanpreet Chauhan

Meine Arbeit war ein Klassendiagramm zu entwerfen, das eine Anwendung modelliert, die mehrere Kernfunktionen integriert:

- Verwaltung von Benutzerdaten und Benutzersitzungen
- Entwicklung eines HTTP-Servers für die Kommunikation mit Clients
- Modellierung von Spielkarten sowie der Kampfmechanik für ein strategisches Kartenspiel



2. Vorgehensweise

2.1 Entwurf der zentralen Klassen

- **Card (Abstrakte Basisklasse):**
  - Diese Klasse definiert grundlegende Eigenschaften und Verhaltensweisen für alle Kartentypen, etwa `Damage` und `CardElementType`. Sie dient als Vorlage für die verschiedenen Kartentypen.
  - Abgeleitete Klassen wie `NormalCard`, `SpellCard` und `MonsterCard` spezifizieren zusätzliche Eigenschaften und Methoden, die für ihre jeweilige Art von Karte erforderlich sind.
  - Methoden wie `PlayCard` und `AssignDamage` regeln die Funktionsweise der Karten während des Spiels und steuern ihre Interaktionen im Spielverlauf.
- **User:**
  - Diese Klasse repräsentiert einen Benutzer im System und umfasst Attribute wie `Username`, `Deck`, `Coins` sowie `SessionToken`, die alle wesentlichen Informationen über den Benutzer enthalten.
  - Sie stellt Funktionen wie `AddPackage`, `CreateCard` und `Logon` zur Verfügung, um Benutzerdaten zu verwalten, Karten zu erstellen oder das Konto zu verwalten.
- **Battle:**

- Die Battle-Klasse modelliert den Ablauf eines Kampfes zwischen zwei Spielern. Sie definiert Methoden wie `Start` und `Play`, mit denen der Kampf initiiert und die einzelnen Runden gespielt werden.
- Innerhalb eines Kampfes wird unter anderem das Ausspielen von Karten sowie die Berechnung von Schäden durchgeführt.

2.2 Umsetzung des HTTP-Servers

- **HttpSvr:**
  - Diese Klasse ist für die Implementierung der grundlegenden Serverfunktionalitäten verantwortlich, darunter das Starten und Stoppen des Servers mit den Methoden `Run` und `Stop`.
  - Sie empfängt eingehende HTTP-Anfragen über die Methode `Incoming` und leitet die entsprechenden Antworten zurück an den Client.
- **HttpHeader und HttpStatusCode:**
  - Diese beiden Klassen bieten Funktionen zur Verwaltung und Interpretation von HTTP-Headern und Statuscodes.
  - Sie ermöglichen die korrekte Handhabung von Anfragen und die Interpretation von Antworten gemäß dem HTTP-Protokoll.

2.3 Verwaltung von Benutzersitzungen

- **Handler (Abstrakte Klasse):**
  - Die Handler-Klasse stellt die Basislogik für spezifische Handler wie `SessionHandler` und `UserHandler` zur Verfügung.
  - Mithilfe der Methode `HandleEvent` werden eingehende Ereignisse bearbeitet, und es wird entschieden, wie das System darauf reagieren soll.
- **SessionHandler:**
  - Der `SessionHandler` verwaltet die aktiven Benutzersitzungen. Zu seinen Aufgaben gehört es, Benutzernamen aus einer Sitzung zu extrahieren und die zugehörigen Daten zu verwalten.
- **UserHandler:**
  - Diese Klasse übernimmt die Verwaltung von Benutzerdaten und implementiert Methoden wie `HandleAddPackage` und `HandleGetUser`, um Funktionen wie das Hinzufügen von Kartendecks und das Abrufen von Benutzerdaten zu ermöglichen.

2.4 Token-basierte Authentifizierung

- **Token (Statische Klasse):**
  - Diese Klasse ist für die Authentifizierung der Benutzer zuständig. Sie erstellt und validiert Tokens, die zur Identifizierung und zur Sicherstellung der Benutzersicherheit dienen.
  - Die Methoden `CreateTokenFor` und `Authenticate` ermöglichen es, Benutzer zu authentifizieren und sicherzustellen, dass nur berechtigte Anfragen bearbeitet werden.

2.5 Erweiterung der Funktionalitäten

- **Round:**
  - Diese Klasse erweitert die Battle-Logik, indem sie einzelne Runden innerhalb eines Kampfes modelliert. Jede Runde wird als eigenständige Einheit betrachtet, in der Karten ausgetauscht und Kampfaktionen ausgeführt werden.
- **Enums und Delegates:**
  - **ElementType:** Diese Enumeration definiert die verschiedenen Kartentypen wie `Water`, `Fire` und `Normal`, die die Spielmechanik beeinflussen.
  - **HttpSvrEventHandler:** Ein Delegate, der verwendet wird, um bestimmte Ereignisse im HTTP-Server zu verarbeiten, etwa bei der Behandlung von eingehenden Anfragen.

3. Herausforderungen

Die Entwicklung dieses Systems brachte mehrere Herausforderungen mit sich, die sowohl technischer als auch konzeptioneller Natur waren:

- **Sitzungsmanagement und Authentifizierung:**
  - Ein weiterer wichtiger Aspekt war das Management aktiver Benutzersitzungen. Hierbei galt es sicherzustellen, dass alle Sitzungen korrekt erkannt und verwaltet werden, um sicherzustellen, dass Benutzer nicht gleichzeitig mit unterschiedlichen Identitäten auf das System zugreifen konnten. Besonders herausfordernd war das Token-Management, da diese Tokens eine zentrale Rolle in der Benutzerauthentifizierung spielen. Die Implementierung eines sicheren und effizienten Mechanismus zur Erstellung, Validierung und Verwaltung dieser Tokens erforderte viel Aufmerksamkeit auf Sicherheit und Performance.
- **Multithreading und Server-Management:**
  - Da der HTTP-Server in der Lage sein muss, gleichzeitig mehrere Anfragen von Benutzern zu verarbeiten, war die Implementierung eines effizienten Multithreading-Ansatzes von entscheidender Bedeutung. Das System musste so ausgelegt sein, dass es nicht nur Anfragen synchron und sequenziell verarbeiten konnte, sondern auch bei hoher Last stabil blieb. Dies stellte besonders bei der Verwaltung der Sitzungen und beim Handling gleichzeitiger HTTP-Anfragen eine Herausforderung dar.
- **Fehlerbehandlung und Robustheit:**
  - Die Fehlerbehandlung war ein weiterer kritischer Punkt. Besonders in einem Multiplayer-System, bei dem mehrere Benutzer gleichzeitig agieren, war es erforderlich, umfassende Validierungen und Fehlerbehandlungen zu integrieren, um Abstürze oder inkonsistente Zustände zu vermeiden. Hierzu mussten alle Eventualitäten in der Kommunikation zwischen Client und Server berücksichtigt werden, wie etwa fehlerhafte Anfragen, ungültige Sitzungen oder Token, sowie unerwartete Serverfehler.

4. Ergebnisse

Das entwickelte Klassendiagramm stellt eine solide Grundlage für die Weiterentwicklung der Anwendung dar. Es umfasst folgende wesentliche Funktionen:

- Ein flexibler HTTP-Server zur Kommunikation zwischen Clients und Server
- Eine detaillierte Abbildung der Karten- und Kampfmechanik

- Effiziente Verwaltung von Benutzerdaten und Sitzungen
- Implementierung eines sicheren Token-Systems für die Authentifizierung

Die durch das Klassendiagramm modellierten Komponenten ermöglichen es, das System schrittweise zu erweitern und weitere Funktionalitäten zu integrieren.

5. Nächste Schritte

- **Sicherheitsoptimierungen:** Zur Verbesserung der Datensicherheit sind erweiterte Mechanismen zur Verschlüsselung und Integritätssicherung notwendig, um den Schutz der Benutzerdaten und die Sicherheit der Transaktionen zu gewährleisten.
- **Erweiterung der Karteneffekte:** Die nächsten Entwicklungsschritte konzentrieren sich auf die Implementierung spezifischer Karteneffekte und ihrer Auswirkungen auf das Kampfgeschehen. Hierbei sollen auch Interaktionen zwischen verschiedenen Kartenelementtypen berücksichtigt werden.
- **Testautomatisierung:** Zur Sicherstellung der Funktionsfähigkeit und Stabilität des Systems sollen Unit-Tests entwickelt werden, die einzelne Komponenten des Systems abdecken und auf mögliche Fehlerquellen hinweisen.

6. Fazit

Das Klassendiagramm bildet eine solide Grundlage für die weitere Entwicklung der Anwendung. Es stellt sicher, dass die Hauptkomponenten – Benutzerverwaltung, Sitzungsmanagement, Authentifizierung und Kampfmechanik – korrekt modelliert und in das System integriert werden. Mit den definierten Strukturen und Erweiterungen ist das System gut für zukünftige Anpassungen und Erweiterungen gerüstet. Mein Github-Repository Link:

https://github.com/hchauhan39-FH/Monstercard_FH