1. Einleitung

Die vorliegende Dokumentation beschreibt die Architektur und Implementierung der Softwareanwendung „Kontoverwaltung“. Bei dem System handelt es sich um eine objektorientierte Windows Forms-Anwendung, entwickelt in C#. Ziel der Anwendung ist die Verwaltung von Bankkunden, Konten sowie die Abwicklung und Historisierung von Finanztransaktionen. Besonderes Augenmerk liegt auf der persistenten Datenhaltung mittels CSV-Dateien und der Umsetzung objektorientierter Paradigmen wie Vererbung und Polymorphie.

2. Systemarchitektur

Die Anwendung folgt einer Schichtenarchitektur, unterteilt in:

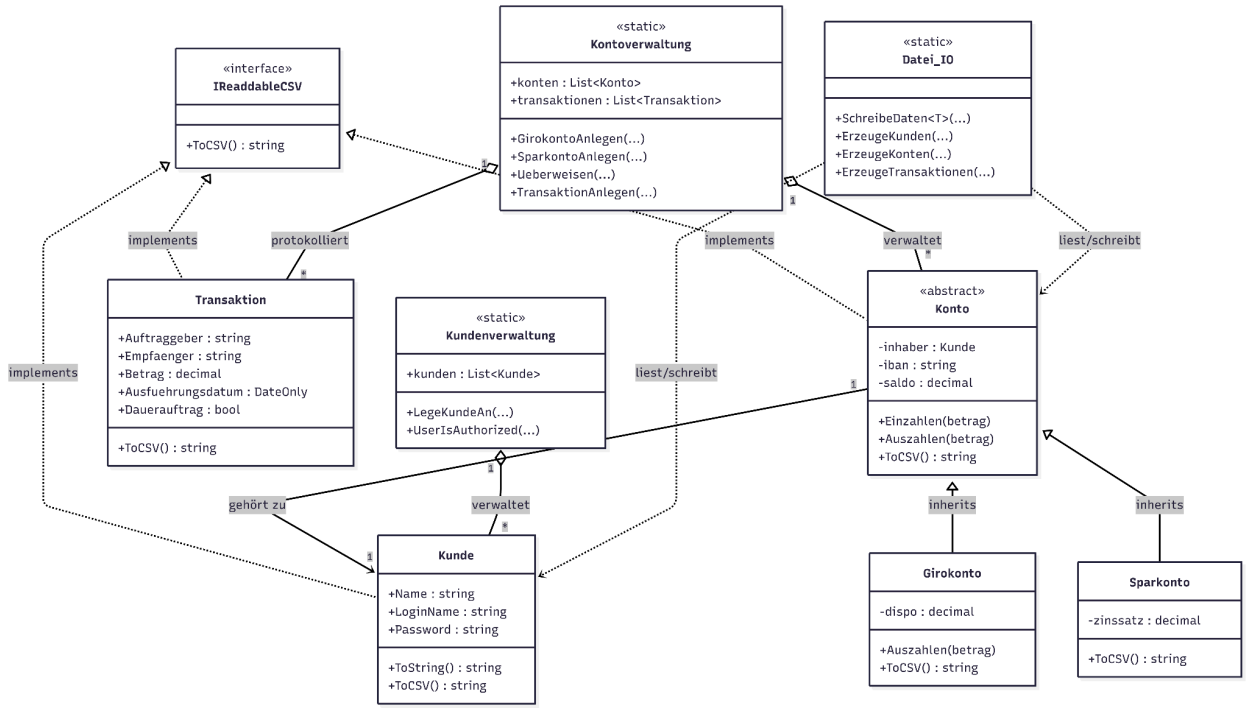
1. **Präsentationsschicht (GUI):** Windows Forms (Hauptfenster, KontoPortal, CreateKonto, etc.) zur Benutzerinteraktion.

2. **Logikschicht:** Geschäftslogik in statischen Verwaltungsklassen (Kontoverwaltung, Kundenverwaltung) und Entitätsklassen (Konto, Kunde).

3. **Datenhaltungsschicht:** Die Klasse Datei_IO übernimmt das Lesen/Schreiben von CSV-Dateien, deren Pfade zentral in Config definiert sind.

3. Klassendiagramm

Die folgende Abbildung visualisiert die statische Struktur der Anwendung. Sie verdeutlicht die Vererbungsbeziehungen zwischen den Kontoklassen sowie die Implementierung der Schnittstelle IReaddableCSV.



Die zentralen Beziehungen sind:

- Girokonto und Sparkonto sind Spezialisierungen der abstrakten Klasse Konto.
- Sowohl Kunde, Konto als auch Transaktion implementieren IReaddableCSV, um Speicherfähigkeit zu garantieren.
- Die statischen Klassen Kundenverwaltung und Kontoverwaltung aggregieren Listen der jeweiligen Entitäten zur Laufzeit.

4. Datenmodell und Implementierungsdetails

4.1. Entitäten und Vererbung

Das Datenmodell basiert auf den Klassen Kunde, Konto und Transaktion.

- **Konto:** Eine abstrakte Basisklasse definiert Grundattribute wie IBAN und Saldo[4].

- **Polymorphie:** Girokonto und Sparkonto erben von Konto. Girokonto erweitert die Logik um einen Dispokredit [5], während Sparkonto einen Zinssatz führt[6].

Beispiel der polymorphen Implementierung in Girokonto:

```
C#
public class Girokonto : Konto
{
   private decimal dispo;
   // Konstruktor ruft Basis-Konstruktor auf
   public Girokonto(Kunde inhaber, string iban) : base(inhaber, iban)
   {
      Dispo = 100M;
   }

   // Überschriebene Auszahlen-Methode prüft Dispo
   public override void Auszahlen(decimal betrag)
   {
      if (this.Dispo + base.Saldo < betrag)
         throw new ArgumentException("Nicht genug Saldo vorhanden");
      base.Saldo -= betrag;
   }
}
```

4.2. Schnittstellen (Interfaces)

Das Interface IReaddableCSV erzwingt eine Methode zur CSV-Konvertierung für alle speicherbaren Klassen.

```
C#
public interface IReaddableCSV
{
   string ToCSV();
}
```

5. Funktionale Komponenten

5.1. Datenpersistenz (Datei_IO)

Die Klasse Datei_IO interagiert mit dem Dateisystem. Sie nutzt Generics (SchreibeDaten<T>), um Listen beliebiger Objekte zu speichern, die das Interface IReaddableCSV implementieren. Beim Laden (ErzeugeKonten) wird anhand eines Typ-Indikators in der CSV ("G" oder "S") die korrekte Klasse instanziiert .

Um die Datenintegrität unabhängig von den lokalen Systemeinstellungen (Regionseinstellungen) des Betriebssystems zu gewährleisten, wird bei allen Lese- und Schreibvorgängen explizit die Klasse `System.Globalization.CultureInfo` verwendet.

Insbesondere bei der Serialisierung von Gleitkommazahlen (Salden, Zinssätze) und Datumsangaben in das CSV-Format stellt dies sicher, dass Trennzeichen (z. B. Dezimalpunkt vs. Komma) konsistent behandelt werden. Dies verhindert Laufzeitfehler (`FormatException`), wenn die Anwendung auf Systemen mit abweichenden Ländereinstellungen ausgeführt wird.

5.2. Transaktionslogik

Finanztransaktionen werden durch die Klasse Kontoverwaltung gesteuert. Die Methode Überweisen koordiniert Einzahlen und Auszahlen atomar zwischen zwei Konten. Jede Bewegung wird zudem als Transaktion-Objekt protokolliert.

5.3. Validierung und Sicherheit

Bei der Registrierung (Register.cs) wird die Passwortstärke mittels Regular Expressions (Regex) validiert, um Sicherheitsstandards zu erfüllen:

C#
```
private Regex _Password = new
Regex("^(?=.*[a-z].*)(?=.*[A-Z].*)(?=.*[0-9].*)(?=.*\\W.*).{8,}$");
```

6. Zusammenfassung

Das Projekt realisiert ein funktionsfähiges Kontoverwaltungssystem. Durch den Einsatz von Vererbung ist das System flexibel für zukünftige Erweiterungen (z. B. neue Kontoarten). Die Datenhaltung via CSV und die klare Trennung von GUI und Logik entsprechen gängigen Architekturmustern für Desktop-Anwendungen.