PR1 VU Programmierung 1	Abschlussklausur	13.07.2021

Sparverein

Implementieren Sie die Klassen **`Zahlung`** und **`Logbuch`** zur Verwaltung der Zahlungen der Mitglieder eines kleinen Vereins.

Eine **`Zahlung`** umfasst den eingezahlten Betrag (ganze Zahl >0 und <100), den noch verfügbaren Betrag (ganze Zahl >=0 und <= eingezahlter Betrag) , einen Namen (String mit Länge>0; der Einfachheit halber darf davon ausgegegangen werden, dass Namen in diesem Beispiel eindeutig sind), und einen Zweck. Der Zweck ist ein Wert aus der vordefinierten Enumeration **`Zweck`** (**`Zweck::Stammtisch`**, **`Zweck::Film`**, **`Zweck::Sport`** und **`Zweck::Reise`**).
Folgende Methoden und Operatoren sind für die Klasse **`Zahlung`** zu implementieren:

- Konstruktor(en) mit zwei und drei Parametern. Eingezahlter Betrag, Name und Zweck in dieser Reihenfolge. Der Zweck ist optional mit Defaultwert **`Zweck::Stammtisch`**. Der verfügbare Betrag von neu angelegten **`Zahlung`**-Objekten ist immer gleich dem eingezahlten. Entspricht einer der Parameter nicht den Vorgaben (z.B. Leerer Name oder Betrag nicht im erlaubten Bereich) so ist eine Exception vom Typ **`runtime_error`** zu werfen.
- **`int bezahlen(int zu_zahlender_betrag)`**: Reicht der verfügbare Betrag aus, um den zu zahlenden Betrag abzudecken, so ist der verfügbare Betrag entsprechend zu reduzieren und 0 zurückzuliefern. Andernfalls wird der verfügbare Betrag auf 0 gesetzt und der noch zu zahlende Teilbetrag (Differenz aus zu zahlendem Betrag und verfügbarem Betrag) wird retourniert. Ist der zu zahlende Betrag negativ, so ist eine Exception vom Typ **`runtime_error`** zu werfen.
- **`int get_guthaben() const`**: Liefert den noch verfügbaren Betrag zurück.
- **`string get_name() const`**: Liefert den in der Zahlung gespeicherten Namen zurück.
- **`Zweck get_zweck() const`**: Liefert den der Zahlung zugeordneten Zweck zurück.
- **`operator<<`**: Die Ausgabe eines Objekts des Typs **`Zahlung`** muss in der Form [*Name*: *Zweck*, *verfügbarer Betrag/eingezahlter Betrag*] erfolgen. Der vordefinierte Vektor **`zweck_namen`** kann für die Ausgabe der Enumerationswerte verwendet werden, z.B.: [Franz: Film, 40/60].

Für ein **`Logbuch`** ist eine Bezeichnung (String mit Länge >0) und die Liste der eingegangenen Zahlungen zu verwalten. Folgende Methoden und Operatoren sind für die Klasse **`Logbuch`** zu implementieren:

- Konstruktor(en) mit einem Parameter oder zwei Parametern. Bezeichnung und Liste von Zahlungen in dieser Reihenfolge. Der zweite Parameter ist optional. Wird keine Zahlungsliste an den Konstruktor übergeben, so ist ein **`Logbuch`**-Objekt mit einer leeren Zahlungsliste zu erstellen. Ist die Bezeichnung leer (Länge gleich 0), so ist eine Exception vom Typ **`runtime_error`** zu werfen.
- **`void eintragen(const Zahlung&)`**: Die im Parameter übergebene Zahlung wird in der Liste der Zahlungen am Ende eingefügt.
- **`bool abgedeckt(const string& name, Zweck zweck, int betrag) const`**: Liefert **`true`**, wenn der gesamte verfügbare Betrag aller Zahlungen, die dem gegebenen Namen und Zweck zugeordnet sind, ausreicht, um den als Parameter erhaltenen Wert zu bezahlen, **`false`** sonst. Ist der Parameterwert **`betrag`** negativ, so ist eine Exception vom Typ **`runtime_error`** zu werfen.
- **`operator<<`**: Die Ausgabe eines Objekts des Typs **`Logbuch`** soll in der Form [*Bezeichnung* {*Liste der Zahlungen*}] erfolgen, z.B.: [Jahresbuch2018 {[Franz: Film, 40/60], [Georg: Stammtisch, 0/70]}].
- Zusatz für 10 Punkte: **`void bezahlen(const string& name, Zweck zweck, int betrag)`**: Falls die Zahlung durchgeführt werden kann (siehe Methode **`abgedeckt`**), so ist die Zahlung durchzuführen, andernfalls ist eine Exception vom Typ **`runtime_error`** zu werfen. Zur Durchführung der Zahlung sind alle relevanten Zahlungen (gleicher Name und gleicher Zweck) in der Reihenfolge, in der sie in der Zahlungsliste eingetragen wurden, zu durchlaufen und die verfügbaren Beträge jeweils entsprechend zu verringern (Methode **`Zahlungen::bezahlen`** kann dazu verwendet werden), bis der Gesamtbetrag bezahlt ist.
- Zusatz für 15 Punkte: **`bool uebertragen(Logbuch& quelle, const string& name, bool vorne)`**: Alle Zahlungen aus der Quelle, die dem spezifizierten Namen zugewiesen sind, werden in die Zahlungsliste des this-Objekts übertragen. Die übertragenen Zahlungen sind aus der Quelle zu entfernen. Der dritte Parameterwert **`vorne`** ist optional mit Defaultwert **`false`**. Er legt fest, ob die übertragenen Zahlungen in der Zahlungsliste des this-Objekts am Beginn (vorne ist **`true`**) oder am Ende (vorne ist **`false`**) einzufügen sind. Die relative Reihenfolge der übertragenen Zahlungen ist in jedem Fall beizubehalten. Die Methode retourniert **`true`**, wenn zumindest eine Zahlung übertragen wurde, **`false`** sonst

Implementieren Sie die Klassen **`Zahlung`** und **`Logbuch`** mit den notwendigen Konstruktoren, Methoden und Operatoren, sodass jedenfalls das Rahmenprogramm kompiliert und ausgeführt werden kann und die gewünschten Ergebnisse liefert. Achten Sie in Ihren Konstruktoren darauf, dass nur gültige Objekte erstellt werden können. Werfen Sie gegebenenfalls eine Exception vom Typ **`runtime_error`**.

Für Ihr Programm dürfen Sie **nur** die im vorgegebenen Rahmenprogramm angeführten include-Dateien verwenden!

Instanzvariablen sind **`private`** zu definieren und die Verwendung globaler Variablen ist (abgesehen von im Rahmenprogramm eventuell bereits definierten) nicht erlaubt! Die Datenkapselung darf nicht durchbrochen werden. Es ist daher unter anderem nicht erlaubt, Referenzen oder Pointer auf private Instanzvariablen einer Klasse nach außen zu vermitteln, **`friend`**-Deklarationen (mit Ausnahme bei Operatorfunktionen) zu verwenden, oder setter-Methoden zu implementieren, die die Integrität der Daten nicht gewährleisten. Interpretationsspielraum in der Angabe können Sie zu Ihren Gunsten nutzen.

Die Teilaufgaben, bei denen keine Punkteanzahl angegeben ist, gelten als Basisfunktionalität. Für eine positive Beurteilung ist zumindest die Basisfunktionalität zu implementieren. Diese wird mit 30 Punkten bewertet.
Die übrigen Teilaufgaben müssen nicht unbedingt implementiert werden, führen aber im Falle einer korrekten Implementierung zu einer entsprechenden Erhöhung der Punkteanzahl.