LU10.A04 - Vererbung mit vielen Facetten

Setzen Sie verschiedene Facetten der Vererbung in einer Applikation ein.

Hinweise

Diese Aufgabe umfasst viel Wissen und Können, das Sie bisher angewendet haben. Schauen Sie sich die entsprechenden Ausführungen der Theorie und Übungen an!

Auftrag

Die Ausgaben in den Beschreibungen sind als Beispiele zu verstehen.

Ausgangslage

In unterklasse wird gezeigt, was Überschreiben (overwriting) bewirken soll. Die Aufgabenstellung basiert auf dem Beispiel "BankAccount".

Gegeben ist dazu das folgende Klassendiagramm.



Vorgehen Teil 1

Schritt 1

Akzeptieren Sie das Assignment im GitHub Classroom.

Testen Sie die Klasse `Person` mit der Testklasse `TestPerson` in `test_person.py`. Dieser Test muss fehlerfrei ablaufen, da die Klasse `Person` ja bereits besteht.

Schritt 2

Erstellen Sie die Exception-Klasse `AccountIndexException`.

Die Exception erwartet zwei Parameter:

- Die Anzahl Konten des Kunden `max`.
- Den Index des Kontos, das gelesen/gelöscht werden sollte `index`

Die Fehlermeldung soll wie folgt aussehen:

```
Ungültiger Indexwert: {max} Einträge vorhanden, Nummer {index} gefordert
```

Um die Exception zu testen, können Sie ein einfaches Skript am Ende der Datei einfügen, z.B.:

```
...
if __name__ == '__main__':
    raise AccountIndexException(5, 9)
```

Schritt 3

Erstellen Sie nun die Klasse `Customer` und implementieren die Methoden gemäss der Beschreibung. Debei setzen wir nun voll auf die OO-Technik. Wir verwenden nämlich die Klasse `Person` aus einer früheren Aufgabe als Basis und leiten Sie ab. Daher muss die Klasse `Customer` nur noch die Methoden zur Verwaltung der Referenzen zu den Konto-Objekten realisieren.

- `current_assets` liefert das summierte Vermögen aller Konten.
- `add_bank_account` fügt eine Referenz für ein Konto zu.
- `take_bank_account` liefert die Referenz zu einem Konto, das durch den index angegeben wird. Dabei kann dies eine Referenz auf `BancAccount` oder aber auch `SalaryAccount` sein. Bei falschem Index soll eine `AccountIndexException` geworfen werden.
- `str` Gibt die Angaben zum Kunden und seiner Konti aus.
    - Die Angaben des Kunden werden aus der `str`-Methode der Klasse `Person` geholt.
    - Die Angaben zu einem Konto holen wir uns über die `str`-Methode der Klasse `Account`.

```
Kunde    : Pia
Kontotyp : Standard bank account
    Saldo: 1000.0
    Zins : 1.5
```

Hinweise

- Die Klasse `Person` ist im Repo bereits vorhanden.
- Sie können diese Klasse bisher nur teilweise testen, da die Klasse `BankAccount` fehlt. Das werden Sie später aber nachholen.

Schritt 4

Erstellen Sie nun die Klasse `BankAccount` und implementieren die Methoden gemäss der Beschreibung.

- Im Konstruktor muss die Methode `add_bank_account` beim Kunden aufgerufen werden.
- `has_available` prüft ob der angeforderte Betrag kleiner oder gleich dem Saldo (`balance`) ist.
- `withdraw_money` reduziert den Saldo (`balance`) um den angegebenen Betrag, sofern dies möglich ist.

    - Nutzen Sie die Methode `has_available` um zu prüfen, ob der Saldo gross genug ist.
    - Der Returnwert ist der geforderte Betrag oder 0, falls der Bezug nicht möglich war.
- `deposit_money` erhöht den Saldo (`balance`) um den angegebenen Betrag.
- `str` gibt den Namen des Kunden sowie den aktuellen Saldo und den Zinsatz zurück.

Testen Sie die Klasse `BankAccount`, bevor Sie diese für die weitere Arbeit nutzen. Dazu verwenden Sie die Tests in `test_bank_account.py`.

Schritt 5

Erstellen Sie nun die Klasse `SalaryAccount` und implementieren die Methoden gemäss der Beschreibung.

- `overdraft` enthält den maximal zulässigen Überzug.
- `has_available` prüft ob der angeforderte Betrag kleiner oder gleich dem verfügbaren Betrag (`balance + overdraft`) ist.
- `str` gibt den Namen des Kunden, den aktuellen Saldo, den maximalen Überzug und den Zinsatz zurück.

```
Kunde    : Pia
Kontotyp : Salary bank account
    Saldo: 5000.0
    Zins : 2.25
    Überzug: 2500.0
```

Testen Sie die Klasse `SalaryAccount` mit den Tests in `test_salary_account.py`.

Vorgehen Teil 2

Implementieren Sie die `main`-Methode in `main.py` gemäss dem Sequenzdiagramm.

- Ergänzen Sie wenn nötig den Konstruktor von `BankAccount` für die Übergabe der Referenz an die Klasse `Customer`.
- Den Teil „money draw“ im Diagramm führen Sie zwei Mal aus.
    - Einmal mit einem Bezug von 700
    - Einmal mit einem Bezug von 400.

Hier liefert Ihnen die Methode `withdraw_money()` den Wert `True`, wenn ein Bezug von Geld möglich ist. Wenn nicht - weil Saldo zu klein - liefert sie `False`. Dem entsprechend soll die Ausgabe am Bildschirm sein. Vergleichen Sie dazu unten den Printscreen.

Erwartetes Ergebnis

Wenn Sie die Klassen gemäss Klassendiagramm und den Ablauf gemäss Sequenzdiagramm implementiert haben, sollten Sie - unter Berücksichtigung einiger erklärender `print()`-Befehle in

`main` - ein vergleichbares Ergebis erhalten.

```
Angaben zu Kunde
    Name: Pia
    Alter: 23
    Adresse: Hochdorf
Angaben zum den Konti
     Standard bank account
     Salary bank account

von jedem Konto 750.0 beziehen
    Bezug von Standard bank account
    Saldo = 250.0
    Bezug von Salary bank account
    Saldo = " 250.0
Aktuelles Vermögen: 500.0

von jedem Konto noch einmal 400.0 beziehen
    Bezug von Standard bank account
    Fehler: Bezug ist zu hoch für Saldo von 250.0
    Bezug von Salary bank account
    Saldo = -150.0
Aktuelles Vermögen: 100.0

Ende Monat: 3000.- Lohn wird eingezahlt
Aktuelles Vermögen: 3100.0
```

Dauer

3-5 Stunden

Abgabe

Als Push im GitHub Classroom

M320-LU10

René Probst, bearbeitet durch Marcel Suter

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/de/modul/m320_2024/learningunits/lu10/aufgaben/vererbung?rev=1788091538

Last update: **2026/08/30 14:05**