LU02.A03 - Klassendiagramm "BankAccount" umsetzen

Implementieren Sie das Klassendiagramm als Sourcecode.

Auftrag

1. Implementieren Sie das Klassendiagramm `BankAccount` als Python-Code.
   Orientieren Sie sich für die Umsetzung am Beispiel "Door".
2. Halten Sie die BZZ Codingstandards für Python ein.
   Die Aufgabe ist nur dann vollständig gelöst, wenn Sie auch die BZZ-Vorgaben bezüglich «guter Programmierung» einhalten.

Vorgehen

Beim Lösen der Programmieraufgaben orientieren wir uns am TDD (test driven development). Die nötigen Testfälle werden durch die Lehrpersonen in Ihrem GitHub Repository bereitgestellt.

Sie finden weiter unten die genauen Anleitung für das jeweilige Vorgehen. Dieses ist bei der Implementation in allen kommenden Übungen anzuwenden.
Es ist wichtig, dass Sie sich dieses Vorgehen aneignen, da auch die Prüfungen so umgesetzt sind und die erreichte Note durch die Anzahl korrekter Tests gegeben ist.

Dauer

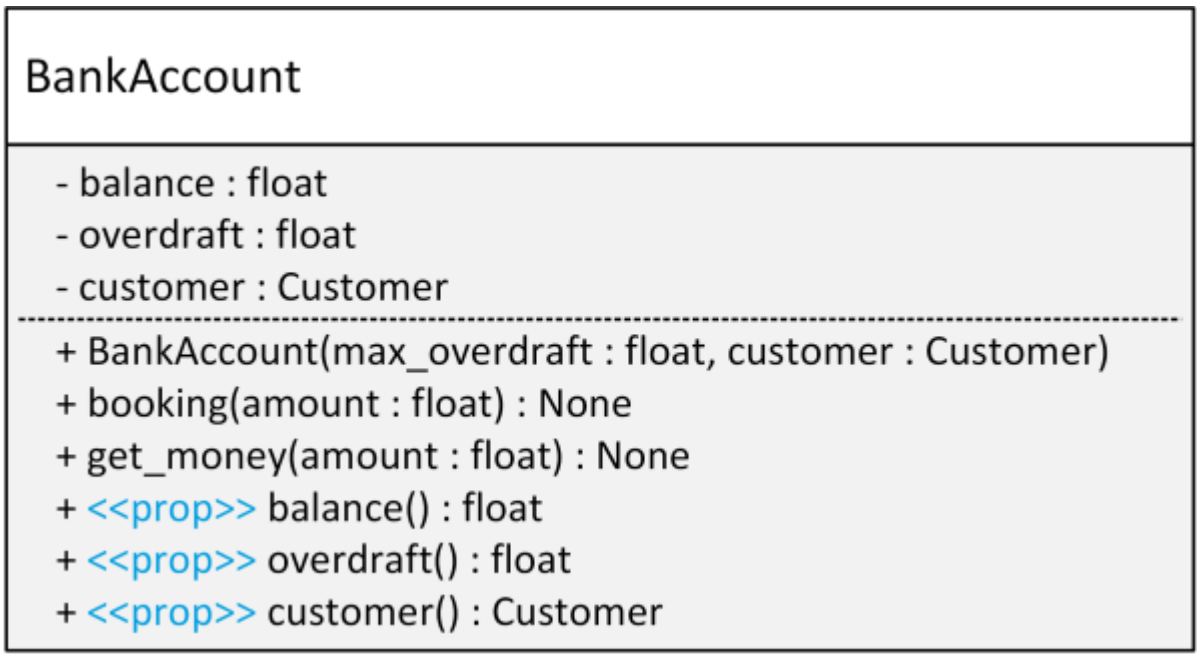
20 Minuten + Hausaufgabenzeit

Abgabe

Commit und Push auf github

Anleitung zu Klasse BankAccount

Klassendiagramm



Beschreibung

Die Klasse BankAccount beschreibt ein Bankkonto für einen Kunden (`Customer`). Das Konto kann innerhalb eines bestimmten Wertes überzogen werden, d.h. dass auch ein negativer Saldo möglich ist.

Attribute

- `balance` gibt den aktuellen Kontostand (Saldo) wieder.
- `overdraft` legt fest, um welchen Betrag das Konto überzogen werden darf, d.h. welcher Minusbetrag möglich ist. (typisch für ein Kreditkonto)
- `customer` ist die Referenz auf ein Objekt der Klasse `Customer`.

Methoden

- `__init__(…)` (der Konstruktor der Klasse BankAccount) initialisiert den Saldo (`balance`) auf 0.0 und legt den Überzug (`overdraft`) sowie den referenzierten Kunden (`customer`) fest.
- `balance()` liefert den aktuellen Saldo des Kontos (kann auch negativ sein).
- `overdraft()` liefert den max. Betrag, um den das Konto überzogen werden darf.
- `customer()` liefert die Referenz zu einem Customer-Objekt.
- `booking(…)` bucht einen Betrag (`amount`) ins Konto ein und erhöht somit den Saldo (`balance`).
- `get_money(…)` bucht einen Betrag (`amount`) vom Konto ab. Dabei darf der Betrag max. so gross sein, dass Saldo + Überzug nicht überschritten werden. Ist der Betrag zu gross, liefert die Methode den Wert 0.0 zurück (= kein Bezug möglich).

Vorgehen

1. Implementieren Sie den Konstruktor (`__init__(…)`) und initialisieren Sie die Attribute gemäss Beschreibung.
2. Erstellen Sie die getter-Methdode (als `@property`) für das Attribut `balance` und testen Sie dies mit der Testmethode `test_initial_balance` in der Datei `test_BankAccount_class.py`.
   Der Test muss fehlerfrei ausgeführt werden. Pushen Sie ihren Code.
3. Erstellen Sie die getter_Methode für das Attribut `overdraft` und testen Sie diese mit `test_initial_overdraft`.
   Der Test muss fehlerfrei ausgeführt werden. Pushen Sie ihren Code.
4. Erstellen Sie die getter-Methode für das Attribut `customer` und testen Sie diese mit `test_customer`.
   Der Test muss fehlerfrei ausgeführt werden. Pushen Sie ihren Code.
5. Erstellen Sie die Methode `booking(…)` gemäss der Beschreibung. Testen Sie die Methode mit `test_booking`.
   Der Test muss fehlerfrei ausgeführt werden. Pushen Sie ihren Code.
6. Erstellen Sie die Methode `get_money(…)` gemäss der Beschreibung. Achten Sie darauf, wie der angeforderte Betrag bezüglich Saldo (`balance`) und Überzug (`overdraft`) geprüft werden muss.
7. Testen Sie nun die Methode der Reihe nach mit
   - `test_get_money_available` für einen korrekten Bezug.
   - `test_get_money_not_avaiable` für einen nicht gültigen Bezug.
   - `test_get_money_overdraft` für einen Bezug innerhalb der Kredit-Limite.
   - `test_balance_after_transaction` für die Kontrolle des Saldos.
   Führen Sie nach jedem der Tests einen commit und push aus!
8. Führen Sie nun die Testklasse als Ganzes aus.
   Das Testergebnis muss wie folgt aussehen:

```
============================= test session starts ==============================
collecting ... collected 8 items

test_BankAccount_class.py::TestBankAccount::test_initial_balance PASSED  [ 12%]
test_BankAccount_class.py::TestBankAccount::test_initial_overdraft PASSED [ 25%]
test_BankAccount_class.py::TestBankAccount::test_customer PASSED         [ 37%]
test_BankAccount_class.py::TestBankAccount::test_booking PASSED          [ 50%]
test_BankAccount_class.py::TestBankAccount::test_get_money_available PASSED [ 62%]
test_BankAccount_class.py::TestBankAccount::test_get_money_not_available PASSED [ 75%]
test_BankAccount_class.py::TestBankAccount::test_get_money_overdraft PASSED [ 87%]
test_BankAccount_class.py::TestBankAccount::test_balance_after_transactions PASSED [100%]

============================== 8 passed in 0.02s ===============================
```

⇒ *GitHub Repo für externe Besucher*

GitHub Repository https://github.com/templates-python/m320-lu02-a03-bankaccount

Lernende am BZZ müssen den Link zum GitHub Classroom Assignment verwenden