LU10.L04: Vererbung mit vielen Facetten

main.py

```
from customer import Customer
from bankaccount import BankAccount
from salary_account import SalaryAccount


if __name__ == '__main__':
    # zuerst die benötigten Objekte erstellen.
    # beide Konti mit Initialwert 1000 erzeugen. Der Überzug beträgt
500. Der Zins ist beliebig.
    pass
    #
    print('Angaben zu Kunde')
    # print aller Kundendaten (Customer-Objekte)
    pass
    print('Angaben zum den Konti')
    # print aller Konten
    pass
    #-------------------------------------------------------------------
----------------------------
    print('\nvon jedem Konto 750.0 beziehen')
    # von beiden Konten geldbeziehen und den Saldo ausgeben
    pass
    #-------------------------------------------------------------------
----------------------------
    print('\nvon jedem Konto noch einmal 400.0 beziehen')
    # Erneut Geld beziehen mit unterschiedlichen Auswirkungen
    pass
    #-------------------------------------------------------------------
----------------------------
    print('\nEnde Monat: 3000.- Lohn wird eingezahlt')
    # Geld einzahlen und den Effekt prüfen.
```

account_index_exception.py

```
class AccountIndexException(Exception):

    def __init__(self, index, max):
        super().__init__(f'Ungültiger Indexwert: {max} Einträge
vorhanden, Nummer {index} gefordert')
```

bankaccount.py

```
""" provides a class for a bank account """
from customer import Customer


class BankAccount:
    """
    A normal bank account

    Attributes:
    -----------
    - owner: Customer
      the owner of the bank account
    - balance: float
        the current balance of the bank account
    - interest: float
        the interest rate of the bank account
    - type: str
        the type of the bank account. 'Standard bank account' is the
default
    """

    def __init__(self, owner, initial_balance, interest):
        """
        Initializes the object
        :param owner: The owner of the bank account
        :param initial_balance: The initial balance of the bank account
        :param interest: The interest rate
        """
        self._owner = owner
        self._balance = initial_balance
        self._interest = interest
        self._type = 'Standard bank account'
        owner.add_bank_account(self)

    def has_available(self, amount):
        """
        Checks if the account has enough money to withdraw the amount
        """
        return amount <= self._balance

    def deposit_money(self, amount):
        """
        Deposits money to the account
        :param amount: The amount to deposit
        """
        self._balance += amount

    def withdraw_money(self, amount):
        """
```

```
        Withdraws money from the account
        :param amount: The amount to withdraw
        """
        if self.has_available(amount):
            self._balance -= amount
            return amount
        else:
            print('Not enough money available')
            return 0

    @property
    def owner(self):
        """
        Returns the owner of the bank account
        :return: The owner of the bank account
        """
        return self._owner

    @property
    def balance(self):
        """
        Returns the balance of the bank account
        :return: The balance of the bank account
        """
        return self._balance

    @property
    def interest(self):
        """
        Returns the interest rate of the bank account
        :return: The interest rate of the bank account
        """
        return self._interest

    @interest.setter
    def interest(self, interest):
        """
        Sets the interest rate of the bank account
        :param interest: The interest rate of the bank account
        """
        self._interest = interest

    @property
    def type(self):
        """
        Returns the type of the bank account
        :return: The type of the bank account
        """
        return self._type

    def __str__(self):
```

```
        output = '-------------------------------------------------\n'
        output += f'Kunde    : {self.owner.name}' + '\n'
        output += f'Kontotyp : {self._type}' + '\n'
        output += f'\tSaldo: {self.balance}' + '\n'
        output += f'\tZins : {self.interest}'
        return output


# Test
if __name__ == '__main__':
    p = Customer('Pia', 24)
    ac = BankAccount(p, 3500.00, 1.75)
    print(ac)
```

customer.py

```
""" This module contains the class Customer.  """
from person import Person
from account_index_exception import AccountIndexException



class Customer(Person):
    """
    Defines a customer as a person who has one or more bank accounts.

    Attributes
    ----------
    - name: str
        The full name of the customer
    - age: int
        The age of the customer
    - accounts: list
        A list of bank accounts the customer has
    """

    def __init__(self, name, age):
        """
        Initializes the object
        :param name: Name of the customer
        :param age: The age of the customer
        """
        super().__init__(name, age)
        self._accounts = []

    def take_bank_account(self, index):
        """
        Returns the bank account at the given index
        :param index: The index of the bank account
```

```
        :return: The bank account at the given index
        :raises AccountIndexException: If the index is out of range
        """
        if index < self.number_of_accounts:
            return self._accounts[index]
        raise AccountIndexException(index, len(self._accounts))

    def add_bank_account(self, account):
        """
        Adds a bank account to the customer
        :param account: The bank account to add
        """
        self._accounts.append(account)

    @property
    def current_assets(self):
        """
        Returns the current assets of the customer
        :return: The current assets of the customer
        """
        asset = 0
        for account in self._accounts:
            asset += account.balance
        return asset


    def __str__(self):
        """
        Returns a string representation of the customer and his/her
accounts
        """
        output = super().__str__() + '\n'
        for account in self._accounts:
            output += account + '\n'


if __name__ == '__main__':
    from bankaccount import BankAccount
    from salary_account import SalaryAccount
    c  = Customer('Pia', 34)
    bc = BankAccount(c, 1000.00, 1.5)
    sc = SalaryAccount(c, 5000.00, 2.25, 2500.0)
    print(c)
```

person.py

```
""" provides a class to describe a person """


class Person:
```

```
    """
    Defines a "general" person with

    Attributes
    ----------
    - name: str
        The full name of the person
    - birth_year: int
        The year the person was born
    - address: str
        The address of the person
    """

    def __init__(self, name, year):
        """
        Initializes the object
        :param name: Name of the person
        :param year: The year the person was born
        """

        self._name = name
        self._birth_year = year
        self._address = ''

    @property
    def name(self):
        """
        Returns the name of the person
        :return: Name of the person
        """
        return self._name

    @property
    def birth_year(self):
        """
        Returns the age of the person
        :return: Age of the person
        """
        return self._birth_year

    @property
    def address(self):
        """
        Returns the address of the person
        :return: Address of the person
        """
        return self._address

    @address.setter
```

```
    def address(self, address):
        """
        Sets the address of the person
        :param address: The address of the person
        """
        self._address = address

    def __str__(self):
        """
        Returns a string representation of the object
        """
        return f'Person: {self._name}\n\tAlter :
{self.birth_year}\n\tAdresse : {self.address}'


# nur für Test
if __name__ == '__main__':
    person = Person('Max', 2000)
    person.address = 'Musterdorf'
    print(person)
```

salary_account.py

```
""" provides a class for a salary account """

from bankaccount import BankAccount
from customer import Customer

class SalaryAccount(BankAccount):
    """
    A salary account

    Attributes:
    -----------
    - owner: Customer
        the owner of the bank account
    - balance: float
        the current balance of the bank account
    - interest: float
        the interest rate of the bank account
    - overdraw: float
        the maximum amount that can be overdrawn
    - type: str
        the type of the bank account. 'Salary bank account' is the
default
    """
    def __init__(self, owner, deposit, interest, overdraft):
        """
        Initializes the object
        :param owner: The owner of the bank account
```

```
        :param deposit: The initial balance of the bank account
        :param interest: The interest rate
        :param overdraft: The maximum amount that can be overdrawn

        """
        super().__init__(owner, deposit, interest)
        self.overdraft = overdraft
        self._type    = 'Salary bank account'

    def has_available(self, amount):
        """
        Checks if the account has enough money to withdraw the amount
        """
        return amount <= (self._balance + self._overdraft)

    @property
    def overdraft(self):
        """
        Returns the overdraft of the account
        :return: The overdraft of the
        """
        return self._overdraft

    @overdraft.setter
    def overdraft(self, overdraft):
        """
        Sets the overdraft of the account
        :param overdraft: The overdraft of the account
        """
        self._overdraft = overdraft


    def __str__(self):
        """
        Returns a string representation of the account
        """
        output = super().__str__() + '\n'
        output += f'\tÜberzug: {self.overdraft}'
        return output

#Test
if __name__ == '__main__':
    p  = Customer('Pia', 24)
    sc = SalaryAccount(p, 3500.00, 1.75, 1000.00)
    print(sc)
```

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/loesungen/bankaccount

Last update: **2026/10/08 06:44**