Codebase Tutorial Summary

Table of Contents

Codebase Tutorial Summary

Summary for `account.py`

- This code is defining a class named `Account`. An `Account` represents a bank account with an owner, balance, a record of transactions, and some functionalities.

When an `Account` is initialized with an owner and a balance, the balance is defaulted to 0 if not given, and an empty transactions list is also created.

The `Account` class has four methods: `deposit`, `withdraw`, `get_balance`, and `get_statement`.

The `deposit` method accepts an amount as an argument and increments the balance by that amount. Subsequently, it logs this activity by appending a new `Transaction` object (with the deposit amount and the string "deposit") to the transactions list.

The `withdraw` method accepts an amount as well. It first checks if the withdraw amount is more than the current balance, if so, it raises a ValueError indicating "Insufficient funds". If the balance is sufficient, it deducts the amount from the balance and records this transaction by appending a new `Transaction` object (with the withdrawal amount and the string "withdrawal") to the transactions list.

The `get_balance` method is a simple method that returns the current balance of the account.

The `get_statement` method returns the list of all transaction activities that has happened on the account by converting each `Transaction` object in the transactions list to a dictionary.

The code also imports `Transaction` class from the `.transaction` file, it is assumed that the `Transaction` class can create a transaction object and has a `to_dict` method to convert a `Transaction` object to a dictionary.

Summary for `bank.py`

- This Python code defines a `Bank` class that represents a basic bank system. The `Bank` class has the following methods:

1. `__init__`: This is the constructor method that gets called when a new object of the class is created. It initializes an empty dictionary called `accounts` that will store all bank accounts. In this dictionary, the `owner` (name of the account owner) is used as the key and an `Account` object is the value.

2. `create_account`: This method is used to create a new account. It takes one argument `owner` (name of account owner). If an account with the given owner name already exists, it raises a `ValueError`. Otherwise, it creates a new `Account` object with the given owner name and adds it to the `accounts` dictionary.

3. `get_account`: This method is used to get the `Account` object under the given owner name. It takes one argument `owner` (name of account owner) and returns the `Account` object if it exists, otherwise, it returns `None`.

4. `transfer`: This method is used to transfer money from one account to another. It takes three arguments: `from_owner` (name of the account owner from where money will be withdrawn), `to_owner` (name of the account owner to whom money will be deposited), and `amount` (amount of money to be transferred). If either of the accounts doesn't exist, it raises a `ValueError`. Otherwise, it performs a withdrawal from the `from_owner` account and a deposit to the `to_owner` account.

The `Account` class, defined in `account.py` (imported at the top), is expected to have at least two methods: `withdraw` and `deposit`, which take the amount of money as an argument and perform appropriate transactions.

Please note, the code assumes no two owners have the same name, as the `owner` string value is used as a unique identifier (key) in the `accounts` dictionary.

Summary for `transaction.py`

- This Python code is a simple implementation of a class named `Transaction`. The purpose of this class is to represent a monetary transaction and it includes traits like amount, transaction type, and the timestamp at which the transaction was conducted.

The class definition begins with the `__init__` method. This is known as the constructor method. It is automatically called when a new object of this class is created. It has four parameters `self`, `amount`, `transaction_type`, and `timestamp`. The `self` parameter is a reference to the current instance of the class and is used to access variables that belongs to the class.

The parameters, `amount` and `transaction_type`, are used to initialize the respective instance variables `self.amount` and `self.transaction_type`. The `datetime.now()` is used to initialize the `self.timestamp` instance variable with the current timestamp.

The method `to_dict` inside the `Transaction` class is to convert the instance variables to a dictionary. This method might be useful if you need to serialize this object, say to JSON format, before storing or transmitting it. It includes the `amount`, `type` (which captures the transaction_type), and `timestamp`.

The `isoformat()` method is used to present the timestamp in the ISO 8601 format, which represents date and time in a way that avoids ambiguity and confusion.

Summary for `__init__.py`