# Codebase Tutorial Summary

Table of Contents

# Codebase Tutorial Summary

## Summary for `account.py`

- This Python program defines a basic `Account` class. This class is a model for a banking account with features like deposit, withdraw, and track transactions.

Here's a breakdown of the components:

1. `from .transaction import Transaction`: This line imports the `Transaction` class from the `transaction` module within the same directory.

2. `def __init__(self, owner, balance=0)`: This is the constructor method for the `Account` class. It is automatically called when a new object of the `Account` type is created. It takes three parameters: `self` (which represents the instance of the class), `owner` (the person who owns the account), and `balance` (the initial balance in the account).

3. `self.transactions = []`: This initializes an empty list of transactions. It will be used to store all transaction details.

4. `deposit(self, amount)` and `withdraw(self, amount)` are instance methods. You use `deposit` to add money to the account balance and log the transaction. You use `withdraw` to remove money from the account balance and also log the transaction. If you try to withdraw more than you have, a `ValueError` with the message `"Insufficient funds"` is raised.

5. `get_balance(self)` is another instance method, which returns the current account balance.

6. `get_statement(self)` is an instance method that returns a list of all transactions made on the account, converting each transaction to a dictionary for readability.

In conclusion, this `Account` class provides simple functionalities of a banking account. It also keeps track of the transactions made on the account, which the account owner can retrieve at any time.

## Summary for `bank.py`

- This code contains a class named 'Bank' that has methods (functions) to manage bank accounts. Let's go through it step-by-step:

- Firstly, an import statement is made to make the 'Account' class from the 'account' module available in this script.

- Inside the 'Bank' class:

- The `__init__` method is the first method that runs when you instantiate (create) a new Bank object. In this method, a dictionary called `accounts` is created, which will store all the accounts with their owners as keys.

- The `create_account` method is designed to create a new bank account. It accepts an argument 'owner', and creates an 'Account' object in the accounts dictionary. If an account for the given owner already exists, it raises a ValueError saying "Account already exists".

- The `get_account` method takes one argument, 'owner', and returns the account for that owner from the accounts dictionary. If there's no account for that owner, Python will return 'None' because `.get()` is used, which unlike direct indexing doesn't throw an error if the key is not found.

- The `transfer` method is designed to transfer the given amount of money from one account to another. This method accepts three arguments: 'from_owner' (source account), 'to_owner' (target account), and 'amount' (the money to be transferred). The checking for existence of both accounts is done. If one or both the accounts do not exist, the method raises a ValueError saying "Account not found". If they exist, it withdraws the amount from the source account and deposits it into the target account.

In summary, this python script uses object-oriented programming techniques to manage bank accounts. The Bank class can add new accounts, retrieve existing ones and perform transfers between accounts.

## Summary for `transaction.py`

- This code defines a simple Python class named 'Transaction'. This class is meant to represent a monetary transaction. It includes characteristics of any common transaction: an amount, a transaction type, and a timestamp representing the time the transaction was created.

First, from the `datetime` module, we import the `datetime` class. This class has several useful methods for dealing with dates and times.

Then, we define the `Transaction` class. Each instance of `Transaction` class will have three attributes: `amount`, `transaction_type`, and `timestamp`.

- `amount`: the amount of money that was transacted

- `transaction_type`: the type of transaction (it could be "debit", "credit", "transfer", etc.)

- `timestamp`: the exact date and time the transaction instance was created

The method `__init__` is a special method for the initialization of an object (instance) after it's been created. When creating a new `Transaction` instance, we need to provide `amount` and `transaction_type` as parameters, while `timestamp` is automatically set to the current date and time by calling `datetime.now()`.

Lastly, the `to_dict` method is used to convert the `Transaction` instance into a dictionary, making it easier accessible or usable in other parts of code, for instance when storing it in a database or sending over a network. Here, the `isoformat()` method converts `datetime` object into an ISO 8601 formatted string (a standard way of representing date and time information).

Thus, the main focus of this code is to model a simple transaction handling mechanism where one can easily create a new transaction and convert the transaction details into a dictionary.

## Summary for `__init__.py`