The MewbileTech Phone Company

Introduction

MewbileTech software is designed to track customer data and information associated with each customer, including customer id, phone lines, call history and the contracts associated with each phone line. The following documentation will describe the public interface and implementation of three different classes in the MewbileTech company code; Term Contract, Month - to - Month Contract and Prepaid contract. The public interface and implementation of all methods will be described for each contract.

Term Contract

Public Interface

One of the possible contracts a phone line of a customer can have is a term contract. This contract has a start date and end date, plus an initial deposit amount that the customer must pay. If the contract is fulfilled and carried until or after the end date, the customer will receive their initial deposit back. Additionally, a term contract includes 100 free minutes at the start of each month which the customer can use without charge. The start date must be before the end date in the term contract.

TermContract Methods

TermContract().__init__(self, start: datetime.date, end: datetime.date) → None

Create a new term contract with <start> and <end> date

TermContract().new_month(self, month: int, year: int, bill: Bill) → None

Advance the contract to a new month corresponding to the given month and year in the parameters. If the month and year parameters passed through the new_month method correspond to the first month of the contract, the <bill> will include a term deposit of $300. Additionally, the <bill> parameter will be charged a monthly fee of $20. The term contract will store the most recent bill, given through the new_month method. *Precondition*: The <month> and <year>

parameters should always succeed the date parameters this method was previously called with and the start date of the Term Contract.

TermContract().bill_call(self, call: Call) → None

The given call in the parameter will be added to the most recent bill stored in the term contract. The customer will be billed $0.10/minute for the duration of the call once their free minutes are used up, if not, the free minutes will be used first for the call. *Precondition:* the bill must be advanced to the correct month and year

TermContract().cancel_contract(self) → float

Returns the amount a customer owes on their most recent bill in order to cancel the contract and deactivate the phone line associated with the term contract of the customer. Thus, returns the amount owed in the bill, minus term deposit if the contract is canceled on or after the designated end date of the term. If a customer cancels before the end date, this method returns only the amount the customer owes on their recent bill.

Method Implementations

TermContract().__init__(self, start: datetime.date, end: datetime.date) → None

This method assigns the end date to self.end and calls the superclass: Contract, to initialize the start parameter. Additionally, a private attribute self._date_tracker is assigned the <start> date parameter to track the current date of the Term Contract and self._free_min_track is assigned to 100 to track the free minutes used in the Term Contract.

TermContract().new_month(self, month: int, year: int, bill: Bill) → None

This method configures the given bill to have the appropriate charge per minute, and fixed cost on the bill. The method calls Bill.set_rates to assign the charge per minute of this term contract bill with the term contract cost per minute of $0.10, and also calls Bill.add_fixed_cost to charge the term monthly fee of $20.00 to the bill. Additionally, the method checks if the <month> and <year> parameters are equal to the start date of the contract. If the month and year parameters of this method are equal to the start date of the contract, the method calls

Bill.add_fixed_cost to add an initial term deposit of $300.00 to the bill. Moreover, the newly configured bill is reassigned to be the most recent bill attribute of the term contract. Lastly, the private attribute _date_tracker is updated with the month and year of the parameters given in this method, and the private attribute self._free_min_track is updated to be 100 minutes again.

TermContract().bill_call(self, call: Call) → None

For each call that the user passes through this method parameter, the duration of the call is calculated and converted to minutes. Then, it is checked if the free minutes can be used to accommodate the entire duration of the call. If the accommodation is possible, the recent bill that is stored in the term contract is updated with the duration of the call, since that is the amount of free minutes that is used in order to make the current call free. The _free_min_track is updated to be reduced by the duration of the call. If the accommodation is not possible, then part of the duration of the call is billed through the Bill.add_billed_minutes method, while the rest is free, depending on the amount of free minutes left. Lastly, since all the free minutes are used to accommodate part of the call duration, bill reflects that all 100 free minutes are used, and the private attribute _free_min_track is reassigned as 0.

TermContract().cancel_contract(self) → float

The private attribute self._date_tracker is compared with the end date of the contract. If the date tracker, which reflects the current month and year, is after or on the end date, then the customer received their deposit back. This is done through calling Bill.add_fixed_cost to the current bill of the term contract, and passing a negative value of the term deposit, which will reduce the customer's bill by $300.00. If the current date is before the end date, then the method simply calls Bill.get_cost, and returns the amount the customer owes on the current bill of term contract.

Month - to - Month Contract

Public Interface

Month to month contract is a type of contract without any end date or term deposit. This contract simply charges customers a monthly fee to use, without any other perks such as free minutes

MTMContract Methods

MTMContract().new_month(self, month: int, year: int, bill: Bill) → None

This method configures the bill given in the parameters to have a charge of $0.05 per minute of a call, and charges a monthly fee of $50.00 to the bill. Lastly, the method stores the recent bill, given through the parameter <bill> as the bill attribute of MTMContract.
Precondition: The <month> and <year> parameters of the new_month method must be the most recent date that this method is called on.

Method Implementations

MTMContract().new_month(self, month: int, year: int, bill: Bill) → None

This method calls the methods Bill.set_rates and Bill.add_fixed_cost on the bill parameter given in the method in order to configure the bill. The method calls Bill.set_rates to assign the charge per minute of this MTM contract bill with the MTM contract cost per minute of $0.05, and also calls Bill.add_fixed_cost to charge the MTM monthly fee of $50.00 to the bill, using the global variables MTM_MINS_COST and MTM_MONTHLY_FEE. Lastly, the newly configured bill is reassigned to be the bill attribute of the MTM contract.

Prepaid Contract

Public Interface

Prepaid contract is the last type of contract a phone line of a customer can have. This contract is initialized with an initial balance that the customer inputs.

Prepaid Methods

PrepaidContract().__init__(self, start: datetime.date, balance: float) → None

Create a new contract with <start> date and <balance> as its initial amount.

PrepaidContract().new_month(self, month: int, year: int, bill: Bill) → None

Advance the contract to a new month by storing the new bill, given through the <bill> parameter, as the bill attribute of the contract. If the credit balance of the contract is less than $10, an additional top-up of $25 is given to the customer. *Precondition:* The <month> and <year> parameters of the new_month method must be the most recent date that this method is called on and succeed the start date of the Prepaid contract.

PrepaidContract().bill_call(self, call: Call) → None

The given parameter of call will be billed to the bill attribute of the contract. *Precondition:* The bill must already be advanced to the most recent month and year

PrepaidContract().cancel_contract(self) → float

Return the amount the customer owes in their most recent bill in order to close out the phone line. If the customer has any credit on their balance, it will not be returned.

Method Implementations

PrepaidContract().__init__(self, start: datetime.date, balance: float) → None

This method calls parent class Contract to initialize self and <start>. Then, it assigns the public attribute, balance, to a negative value of the input <balance> parameter, since it is the credit the customer added to their phone line contract.

PrepaidContract().new_month(self, month: int, year: int, bill: Bill) → None

First, the method reassigns the <bill> parameter given in the method as the most recent bill for the bill attribute of the prepaid contract. The bill is configured to have the proper rates of $0.025/minute through the Bill.set_rates method and the balance is added as a fixed cost to the bill through the Bill.add_fixed_cost method. The method then checks if the prepaid contract attribute is greater than -10, in which case the balance is updated to add an additional -25, which is the top-up received. Also, when giving the top-up, the method will call Bill.add_fixed_cost to add $(-25) to the customer bill in order to accommodate for the update in balance. Lastly, the

balance is updated and reassigned to the value received when calling the Bill.get_cost method in order to accommodate for the carry over into next month.

PrepaidContract.bill_call(self, call: Call) → None

The method determines the cost of the call by converting the call duration into minutes and multiplying by the given rate per minute cost of prepaid contract; $0.025. Then, the method updates the contract attribute, balance, to add the call cost to the balance. Lastly, the current bill attribute of the contract is updated to add the billed minutes of the call duration using the Bill.add_billed_minutes.

PrepaidContract.cancel_contract(self) → float

This method calls Bill.get_cost and determines whether the cost is positive or negative. If the cost is negative, then the method returns 0, since no amount is owed to the company. If the cost is positive, then the method simply returns that cost.