The Documentation of Contract Subclasses For MewBile and Implementation Details For Their Methods

Contents

Term Contracts

A contract that incentivises the customer to keep their contract by holding onto a deposit in return for lower monthly rates and calling rates.

Public Interface

Attributes

start:

The starting date for the contract

type: datetime.date

Representation Invariant:

This date must be before <self.end>

bill:

The bill for this contract for the last month of call records loaded from the input dataset.

type: Optional[Bill] (An in-house class stored in bill.py)

term_fulfilled:

A boolean that tells whether the contract has been kept to the end date.

type: bool

end:

The end date for the contract that customer is incentivised to keep their contract till

type: datetime.date

Representation Invariant:

This date must be after <start.start>

Methods

- **def __init__(self, start: datetime.date, end: datetime.date) -> None:**

Creates a new contract that begins at <start> and with the end date for the deposit at <end>.

- **def new_month(self, month: int, year: int, bill: Bill) -> None:**

Advance to a new month in the contract, corresponding to <month> and <year>, updating <term_fulfilled> or charging a deposit as needed. This may be the first month of the contract. Store the <bill> argument in this contract and set the appropriate rate per minute and fixed cost.

- **def bill_call(self, call: Call) -> None:**

Precondition: the new_month() must have already been called to ensure that the contract contains a bill of the same month and year as the call occurred.

This method adds the duration of the call rounded up to the bill, accounting for the number of free minutes as needed.

- **def cancel_contract(self) -> Optional[float]:**

Precondition: the new_month() must have already been called to ensure that the contract contains a bill of the same month and year as when this method is called.

This function cancels the contract and returns a float that if positive is the amount the customer owes to MewBile, or is owed otherwise.

Implementation Details

- **def __init__(self, start: datetime.date, end: datetime.date) -> None:**

  This method creates four instance attributes. It first calls Contract.__init__(self, start) to create <self.bill> and <self.start> before assigning instance attributes <self.end> to <end> and <self.term_fulfilled> which is initially set to False.

- **def new_month(self, month: int, year: int, bill: Bill) -> None:**

  This function first sets the calling rate for <bill> by using the public method of the Bill class Bill.set_rates() with constants <TERM_MINS>. Then the method has an if statement that checks whether the tuple of <year> and <month> match the tuple of <self.start.month> and <self.start.year>, if it does then the function uses the public Bill method *Bill.add_fixed_cost()* with the <TERM_DEPOSIT> constant passed. The method uses *Bill.add_fixed_cost()* again *with the argument* <TERM_MONTHLY_FEE> to charge the monthly fee, before assigning <bill> to <self.bill>. Finally, this method checks the tuple of <year> and <month> to <self.end.month> and <self.end.year>, and if they match will update <self.term_fulfilled> to <True>.

- **def bill_call(self, call: Call) -> None:**

This function assigns a <duration> variable to be the duration attribute of the call <call> divided by 60 and rounded up to the nearest integer. Then to determine whether the customer has enough free minutes left for this call, this function checks whether adding <duration> to <self.bill.free_mins> would result in an integer strictly greater than <TERM_MINS>. If the sum is greater than <TERM_MINS>, then this function will charge the difference of the sum and <TERM_MINS> as billed minutes using Bill.add_billed_minutes() before setting the contract's bill's free minutes attribute to <TERM_MINS>. If the sum is not greater, then the function will simply add the call's duration to the <self.bill> with Bill.add_free_mins().

- **def cancel_contract(self) -> Optional[float]:**
  This function checks whether <self.term_fulfilled> is true in order to see if the contract has been carried to the end of its term. If it has then it sets <self.start> to <None> before returning current month’s bill total obtained by calling self.bill.get_cost() minus <TERM_DEPOSIT>. If <self.term_fulfilled> is False, then this method calls Term Contract’s parent class Contract.cancel_contract which reassigns <self.start> to <None> before returning the month’s bill obtained with self.bill.get_cost()

Month to Month Contracts

A contract with no end date or deposit in exchange for higher monthly fees and calling rates.

Public Interface

Attributes

start:

type: datetime.date
The starting date for the contract

bill:

type: Optional[Bill] (An in-house class stored in bill.py)
The bill for this contract for the last month of call records loaded from the input dataset. This attribute is <None> when this contract is inactive.

Methods

- **def __init__(self, start: datetime.date) -> None:**

Creates a new contract that begins at <start> with no bill.

- **def new_month(self, month: int, year: int, bill: Bill) -> None:**

This method replaces the bill stored by this contract regardless of if there is one and sets up, and charges the monthly fee of MTM contracts.

- **def bill_call(self, call: Call) -> None:**

Precondition: the new_month() must have already been called to ensure that the contract contains a bill of the same month and year as the call occurred.

Charges the call duration rounded up to the nearest minute to the bill stored by the contract.

- **def cancel_contract(self) -> float:**

Precondition: the new_month() must have already been called to ensure that the contract contains a bill of the same month and year as when this function is called.

This method cancels the contract and returns the current total cost of this month's expenditure

Implementation Details

- **def __init__(self, start: datetime.date) -> None:**

This method creates two instance attributes by calling the parent class's __init__ function Contract.__init__(self, start) to create <self.bill> which is assigned to None and <self.start> which is assigned to <start>.

- **def new_month(self, month: int, year: int, bill: Bill) -> None:**

This method first calls Bill.set_rates() to set the rate of the bill to constants MTM_MINS_COST before charging the monthly fee <MTM_MONTHLY_FEE> using the method Bill.add_fixed_cost. Then this function assigns <bill> to <self.bill>.

- **def bill_call(self, call: Call) -> None:**

Since this function does not have any included free minutes for the customer. This method is inherited from the parent class <Contract>. Which uses the Bill method Bill.add_billed_minutes() with the duration attribute from <call> divided by 60 and rounded up to the closest minute.

- **def cancel_contract(self) -> None:**

This method is inherited from <Contract>. This method reassigns the attribute <self.start> to <None> to indicate that this contract has been cancelled and returns the value obtained by Bill.get_total_cost() so the program can charge the customer their remaining balance.

Prepaid Contracts

A contract where the customer must top up a balance in order to pay the bill

Public Interface

Attributes

start:

type: datetime.date
The starting date for the contract

bill:

type: Optional[Bill] (An in-house class stored in bill.py)
The bill for this contract for the last month of call records loaded from the input dataset.

balance:

type: float

When balance > 0, this number indicates the amount of money the customer owes. When balance < 0, the absolute value of this number indicates the amount of money the customer has as credit that can be used to pay for calls.

Methods

- **def __init__(self, start: datetime.date, balance: float) -> None:**

Creates a new contract that begins at <start> with no bill, and has <balance> amount of credit.

- **def new_month(self, month: int, year: int, bill: Bill) -> None:**

This method updates the bill stored by this contract to <bill> with the proper rates and charges a $25 top up fee if the contract's credit is less than $10.

- **def bill_call(self, call: Call) -> None:**

Precondition: the new_month() must have already been called to ensure that the contract contains a bill of the same month and year as the call occurred.

Charges the call duration rounded up to the nearest minute to the bill stored by the contract.

- **def cancel_contract(self) -> Optional[float]:**

This method cancels the contract and returns the remaining amount owed by the customer, this method returns None customer does not owe any money.

Implementation Details

- **def __init__(self, start: datetime.date, balance: float) -> None:**

This method creates three instance attributes <self.start>, <self.balance> and <self.bill>. These attributes are initially assigned the values <start>, <0 - balance> and <None> respectively. <balance> is added as a negative in order to indicate the amount of credit the customer has started with.

- **def new_month(self, month: int, year: int, bill: Bill) -> None:**

This method first sets the rates of <bill> using Bill.set_rates() with the <PREPAID_MINS_COST>. Then this method checks whether there is already a Bill object stored in <self.bill>, if there is then the function updates <self.balance> to the value obtained by running <self.bill.get_cost()>. This function then has an if statement that checks whether there is less than $10 of credit, if there isn't then <self.balance> is reduced by $25 to reflect the top up then the contract's current balance is added to <bill> by using Bill.add_fixed_cost().

- **def bill_call(self, call: Call) -> None:**

Since this function does not have any included free minutes for the customer. This method is inherited from the parent class <Contract>. Which uses the Bill method Bill.add_billed_minutes() with the duration attribute from <call> divided by 60 and rounded up to the closest minute.

- **def cancel_contract(self) -> Optional[float]:**

This method updates <self.balance> to <self.bill.get_cost()>, before reassigning <self.start> to None. If <self.balance> is greater than 0, then this function returns <self.balance>, None otherwise.