Contract Class

A contract for a phone line. This class is not to be changed or instantiated. It is an Abstract Class.

Public Attributes

- `start: datetime.date`
  - Starting date for the contract.
- `bill: Optional[Bill]`
  - Bill for this contract for the last month of call records loaded from the input dataset

Public Methods

- `__init__(self, start: datetime.date) -> None`
  - **Description:** Create a new Contract with the `start` date, starts as inactive
- `new_month(self, month: int, year: int, bill: Bill) -> None`
  - **Description:** A new month has begun corresponding to `month` and `year`. 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.
- `bill_call(self, call: Call) -> None`
  - **Description:** Add the `call` to the bill.
  - **Precondition:** A bill has already been created for the month+year when the call was made. In other words, you can safely assume that `self.bill` has been already advanced to the right month+year.
- `cancel_contract(self) -> float`
  - **Description:** Return the amount owed in order to close the phone line associated with this contract.
  - **Precondition:** A bill has already been created for the month+year when this contract is being canceled. In other words, you can safely assume that `self.bill` exists for the right month+year when the cancellation is requested.

Implementation

The `Contract` class serves as an abstract base class and should not be instantiated directly.

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

The initialization method for the `Contract` class accepts a parameter `start` represent the start date for this contract, which is then assigned to `self.start` using `self.start = start`.

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

The `new_month()` method is marked as abstract and should be implemented in subclasses derived from the `Contract` class.

`bill_call(self, call: Call) -> None`

The `bill_call()` method begins by computing the ceiling value of `call.duration / 60.0` to avoid decimal minutes. The resulting amount is then added to the current month bill's billed minutes using `self.bill.add_billed_minutes(ceil(call.duration / 60.0))`.

`cancel_contract(self) -> float`

This method sets `self.start` to `None` and returns the cost of the current month's bill by using `return self.bill.get_cost()`.

TermContract Class

A term contract for a phone line.

Public Attributes

- `end: datetime.datetime`
  - Stores the end date for this contract.
- `current_month: int`
  - Stores the current billing month.
- `current_year: int`
  - Stores the current billing year.

Public Methods

- `__init__(self, start: datetime.date, end: datetime.date) -> None`
  - **Description:** Create a new TermContract with the specified `start` date and `end` date. The contract starts as inactive.
- `new_month(self, month: int, year: int, bill: Bill) -> None`
  - **Description:** A new month has begun corresponding to `month` and `year`. 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. Also store the `month` and `year` to keep track of the bill current date
- `bill_call(self, call: Call) -> None`
  - **Description:** Add the `call` to the bill.
  - **Precondition:** A bill has already been created for the month+year when the `call` was made. In other words, you can safely assume that `self.bill` has been already advanced to the right month+year.
- `cancel_contract(self) -> float`
  - **Description:** Return the amount owed in order to close the phone line associated with this term contract. If the customer cancels the contract early, the deposit is forfeited. If the contract is carried to term, the customer gets back the term deposit minus that month's cost.
  - **Precondition:** A bill has already been created for the month+year when this contract is being canceled. In other words, you can safely assume that `self.bill` exists for the right month+year when the cancellation is requested.

Implementation

The `TermContract` class extends from the abstract `Contract` class.

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

This initialization method for the `TermContract` class takes the start and end dates as parameters, inherits from the parent class `Contract`, and stores the end date in `self.end` by `self.end = end`.

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

This method set a new `bill` for the `month` and `year` to the `TermContract` class. The `new_month()` method first uses `bill.set_rates("TERM", TERM_MINS_COST)` to set the billing rates to the constant `TERM_MINS_COST` and the billing type to `TERM`, representing the term contract bill. It then adds a fixed cost to the bill as `bill.add_fixed_cost(TERM_MONTHLY_FEE)`, representing the monthly payment fee for the term contract. The method checks if the month and year passed in as parameters are the start month and year for this contract. If so, it adds an additional `TERM_DEPOSIT` constant (which represent the term deposit fee) to the bill's fixed cost as `bill.add_fixed_cost(TERM_DEPOSIT)`. After setting the bill rates and fixed costs, `new_month()` takes the passed-in bill object as a parameter and stores it in `self.bill`, inheriting attributes from the `Contract` class. This method also keeps track of the current month and year with `self.current_month = month` and `self.current_year = year`.

`bill_call(self, call: Call) -> None`

This method begins by creating a local variable `duration` and storing `ceil(call.duration / 60.0)` to it. The use of `ceil(call.duration / 60.0)` is necessary because `call.duration` is in seconds, while the `Bill` class calculates the cost using minutes. The method then checks if `self.bill.free_min` is less than `TERM_MINS`. If `self.bill.free_min < TERM_MINS`, the `bill_call` method continues to check if `self.bill.free_min + duration <= TERM_MINS`. If true, this call is covered by the monthly free minutes, so the method adds the `duration` to free minutes in the bill by `self.bill.add_free_minutes(duration)`. If `self.bill.free_min + duration <= TERM_MINS` is false, the free minutes will be added first, and the remaining duration will be added to billed minutes by `self.bill.add_billed_minutes(charge)`, where `charge = duration - (TERM_MINS - self.bill.free_min)` represents the remaining duration for this call that will be charged.

`cancel_contract(self) -> float`

This method calculates the owed amount. `cancel_contract()` checks whether the customer canceled before or after the end date for the contract, using `self.current_month` and `self.current_year`. If the customer cancels before the end date, the term deposit is forfeited, and `cancel_contract()` returns `self.bill.get_cost()`. If the customer cancels on the end date, the term deposit is refunded to the customer, and `cancel_contract()` returns the current month's bill costs minus term deposit as `self.bill.get_cost() - deposit`.

MTMContract Class

A month to month contract for a phone line

Public Methods

- `new_month(self, month: int, year: int, bill: Bill) -> None`
  - **Description:** A new month has begun corresponding to `month` and `year`. 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.

Implementation

The `MTMContract` class extends from the abstract `Contract` class.

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

This method set a new `bill` for the `month` and `year` to the the `MTMContract` class. This `new_month()` method first uses `bill.set_rates("MTM", MTM_MINS_COST)` to set the billing rates to the constant `MTM_MINS_COST` and the billing type to `MTM`, representing the month-to-month contract bill. It then adds a fixed cost `MTM_MONTHLY_FEE` to the bill as `bill.add_fixed_cost(MTM_MONTHLY_FEE)`, representing the monthly payment fee for the month-to-month contract. After setting the bill rates and fixed costs, the method takes the passed-in bill object and stores it in `self.bill`, inheriting attributes from the `Contract` class.

PrepaidContract Class

A prepaid contract for a phone line

Public Attributes

- `balance: float`
  - Track the balance for this contract, where a negative balance means the customer has this much credit.

Public Methods

- `__init__(self, start: datetime.date, balance: float) -> None`
  - **Description:** Create a new PrepaidContract with the `start` date and `balance` to tack the credit, starts as inactive
  - **Precondition:** `balance` should be a positive float.
- `new_month(self, month: int, year: int, bill: Bill) -> None`
  - **Description:** A new month has begun corresponding to `month` and `year`. 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. Also keep track of balance if balance is less then `$10` add `$25` to it
- `bill_call(self, call: Call) -> None`
  - **Description:** Add the `call` to the bill. Also change the balance according to bill.
  - **Precondition:** A bill has already been created for the month+year when the `call` was made. In other words, you can safely assume that `self.bill` has been already advanced to the right month+year.
- `cancel_contract(self) -> float`
  - **Description:** Return the amount owed in order to close the phone line associated with this prepaid contract. If the contract still has some credit on it (a negative balance), then the amount left is forfeited and returned, otherwise, return the balance.
  - **Precondition:** A bill has already been created for the month+year when this contract is being canceled. In other words, you can safely assume that `self.bill` exists for the right month+year when the cancellation is requested.

Implementation

The `PrepaidContract` class extends the abstract `Contract` class.

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

This initialization method for the `PrepaidContract` class takes the start date and balance as parameters. It inherits from the parent class `Contract` and stores the negative balance in `self.balance`, where the negative balance stands for the credit the customer has when making calls.

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

This method set a new `bill` for the `month` and `year` to the the `PrepaidContract` class. `new_month()` method start from checks if `self.balance` is greater than -10. If true, it adds a `$25` credit to the balance (`self.balance -= 25`). After checking the balance, `new_month()` uses `bill.set_rates("PREPAID", PREPAID_MINS_COST)` to set the cost per minute for calls as `PREPAID_MINS_COST`, and billing type as `PREPAID`, representing the prepaid contract bill. The passed-in bill object is then stored in `self.bill`, inheriting attributes from the `Contract` class.

`bill_call(self, call: Call) -> None`

The `bill_call()` method first takes the ceiling of `call.duration / 60.0` to avoid decimal minutes. This amount is then added to the bill for the current month under `billed_minutes` by `self.bill.add_billed_minutes(ceil(call.duration / 60.0))`. After that, `bill_call()` method updates the balance accordingly with `self.balance = self.bill.get_cost()`.

`cancel_contract(self) -> float`

The `cancel_contract()` method starts by updating `self.balance` one last time by `self.balance = self.bill.get_cost()` to ensure the balance is up to date. It then checks if `self.balance` is negative or not. If it is negative, indicating the customer owes no money, the method forfeits the remaining credit by setting `self.balance = 0` and returns it. If `self.balance` is positive, the method returns `self.balance` since it represents the amount that the customer owes.