API for Contract Class in CSC148 A1 Project

1. Introduction

For each phoneline, a contract is associated with it that holds important information such as the current bill for each month. The specifics of each method in the contracts are determined uniquely by the type of contract they are.

The three types of contracts are coming from the polymorphism of the main class Contract.

2. Term Contract

This type of contract starts off with a initial term deposit added to the bill on the first day of the month. Additionally, term contract has a set number of free minutes given per month.

This method inherits/overrides Contract Methods.

Public Interface:

- enddate: datetime.date
  - The end date for the contract
- currentdate: datetime.date
  - The current date for the contract
- start: datetime.date
  - The starting date for the contract
- bill: Optional[Bill]
  - The bill for the last month of calls

Representation Invariants:

- enddate >= start
  - This is to ensure that the end date is always equal or after the start date.
- The contract should have only one bill per month

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

Creates an object that holds the start attribute of the given contract. Additionally, a bill attribute is instantiated to None. It

then instantiates the enddate attribute based on end and set the currentdate to the start date. This function returns None.

Parameters:

- start: The starting date of the contract.
- end: The ending date of the contract.

Precondition: N/A

A short usage example:

```
>>> import datetime
>>> contract = TermContract(datetime.date(2017, 12, 25),
                    datetime.date(2019, 6, 25))
>>> contract.start
2017-12-25
>>> contract.bill is None
True
```

class TermContract.**cancel_contract**(self) -> float:

Returns the amount owed based on the bill as a float. If the contract is ended before the enddate specified on the contract, then the term deposit is not returned. If it is ended after the enddate, the contract term deposit will be returned.

Parameters: N/A

Precondition: A bill has been created for the month and year and is not None. This means the bill should exists.

A short usage example:

```
>>> import datetime
>>> contract = TermContract(datetime.date(2017, 12, 25),
                    datetime.date(2019, 6, 25))
>>> bill = Bill()
>>> contract.new_month(1, 2018, bill)
>>> contract.cancel_contract()
20.0
```

class TermContract.**new_month**(self, month: int, year: int, bill: Bill) -> None:

Set the rate of 0.1$ per minute and add 20$ of cost into the new bill that replaces the original contract bill. If this is the first element of the contract, add an additional 300$ of term deposit into the bill.

Parameters:

- month: The new month given.

- end: The new year given.
- bill: The new bill for the month

Precondition: 1 <= month <= 12.

A short usage example:

```
>>> import datetime
>>> contract = TermContract(datetime.date(2017, 12, 25),
                            datetime.date(2019, 6, 25))
>>> bill = Bill()
>>> contract.new_month(1, 2018, bill)
>>> contract.bill.get_cost()
20.0
```

class TermContract.**bill_call**(self, call: Call) -> None:

If there are still free minutes remaining within the contract's bill, then those free minutes will be used up first. Afterwards, the remaining duration of the call, if it is still greater than zero, would be added towards the billed mins.

Parameters:

- call: The event call to be added to the contract.

Precondition: A bill has been created for the month and year and is not None. This means the bill should exists.

A short usage example:

```
>>> import datetime
>>> contract = TermContract(datetime.date(2017, 12, 25),
                            datetime.date(2019, 6, 25))
>>> bill = Bill()
>>> contract.new_month(1, 2018, bill)
>>> contract.bill_call(Call("", "", datetime.date.today(),
                            6600, (0, 1), (0, 1)))
>>> contract.bill.get_cost()
21.0
```

3. Month to Month (MTM) Contract

This type of contract contains no term constraints.

This method inherits/overrides Contract Methods.

Public Interface:

- start: datetime.date

    - The starting date for the contract
- bill: Optional[Bill]
    - The bill for the last month of calls

Representation Invariants:

- The contract should have only one bill per month

class MTMContract.**__init__**(self, start: datetime) -> None:

Creates an object that holds the start attribute of the given contract. Additionally, a bill attribute is instantiated to None.

Parameters:

- start: The starting date of the contract.

Precondition: N/A

A short usage example:

```
>>> import datetime
>>> contract = MTMContract(datetime.date(2017, 12, 25))
>>> contract.start
2017-12-25
>>> contract.bill is None
True
```

class MTMContract.**cancel_contract**(self) -> float:

Returns the amount owed based on the bill associated with this contract as a float. This is the inherited method from the class Contract.

Parameters: None

Precondition: A bill has been created for the month and year and is not None. This means the bill should exists.

A short usage example:

```
>>> import datetime
>>> contract = MTMContract(datetime.date(2017, 12, 25))
>>> bill = Bill()
>>> contract.new_month(1, 2018, bill)
>>> contract.cancel_contract()
50.0
```

class MTMContract.**new_month**(self, month: int, year: int, bill: Bill) -> None:

Set the rate of 0.05$ per minute and add the 50$ into the new bill that replaces the original contract bill.

Parameters:

- month: The new month given.
- end: The new year given.
- bill: The new bill for the month

Precondition: 1 <= month <= 12.

A short usage example:

```
>>> import datetime
>>> contract = MTMContract(datetime.date(2017, 12, 25))
>>> bill = Bill()
>>> contract.new_month(1, 2018, bill)
>>> contract.bill.get_cost()
50.0
```

class MTMContract.**bill_call**(self, call: Call) -> None:

Add the duration of the call in minutes into the contract bill's billed_min attribute. This is the inherited method from the class Contract.

Parameters:

- call: The event call to be added to the contract.

Precondition: A bill has been created for the month and year and is not None. This means the bill should exists.

A short usage example:

```
>>> import datetime
>>> contract = MTMContract(datetime.date(2017, 12, 25))
>>> bill = Bill()
>>> contract.new_month(1, 2018, bill)
>>> contract.bill_call(Call("", "", datetime.date.today(),
                           6600, (0, 1), (0, 1)))
>>> contract.cancel_contract()
55.5
```

4. Prepaid Contract

This contract has a balance which is the amount of money the customer owes. When canceling a contract, if the contract still has money in it, then that amount is forfeited.

This method inherits/overrides Contract Methods.

Public Interface:

- start: datetime.date
  - The starting date for the contract
- bill: Optional[Bill]
  - The bill for the last month of calls

Representation Invariants:

- The contract should have only one bill per month

class PrepaidContract.**__init__**(self, start: datetime, balance: float) -> None:

Creates an object that holds the start attribute of the given contract. Additionally, a bill attribute is instantiated to None. Additionally, subtract the current number of prepaid credits to the private balance attribute.

Parameters:

- start: The starting date of the contract.
- Balance: The amount of prepaid credits the user has given.

Precondition: N/A

A short usage example:

```
>>> import datetime
>>> contract = PrepaidContract(datetime.date(2017, 12, 25), 10)
>>> contract.start
2017-12-25
>>> contract.bill is None
True
```

class PrepaidContract.cancel_contract(self) -> float:

Returns the amount owed based on the bill associated with this contract as a float. If this amount is less than zero, then simply return 0.

Parameters: None

Precondition: A bill has been created for the month and year and is not None. This means the bill should exists.

A short usage example:

```
>>> import datetime
>>> contract = PrepaidContract(datetime.date(2017, 12, 25), 10)
>>> bill = Bill()
>>> contract.new_month(1, 2018, bill)
>>> contract.cancel_contract()
0
```

class PrepaidContract.**new_month**(self, month: int, year: int, bill: Bill) -> None:

Set the rate of 0.025$ per minute for the new bill that replaces the original contract bill. Calculate the cost of last month's used billed_min and add it onto the private balance attribute. If the balance is greater than 10$, then subtract a additional 25$. This new balance attribute should now be set as the bill's fixed cost.

Parameters:

- month: The new month given.
- end: The new year given.
- bill: The new bill for the month

Precondition: 1 <= month <= 12.

A short usage example:

```
>>> import datetime
>>> contract = PrepaidContract(datetime.date(2017, 12, 25), 10)
>>> bill = Bill()
>>> contract.new_month(1, 2018, bill)
>>> contract.cancel_contract()
0
```

class PrepaidContract.**bill_call**(self, call: Call) -> None:

Add the duration of the call in minutes into the contract bill's billed_min attribute. This is the inherited method from the class Contract.

Parameters:

- call: The event call to be added to the contract.

Precondition: A bill has been created for the month and year and is not None. This means the bill should exists.

A short usage example:

```
>>> import datetime
>>> contract = PrepaidContract(datetime.date(2017, 12, 25), 10)
>>> bill = Bill()
>>> contract.new_month(1, 2018, bill)
>>> contract.bill_call(Call("", "", datetime.date.today(),
                            60000, (0, 1), (0, 1)))
>>> contract.cancel_contract()
15.0
```

Implementation for Contract Class in CSC148 A1 Project

Constants set:

MTM_MONTHLY_FEE = 50.00

TERM_MONTHLY_FEE = 20.00

TERM_DEPOSIT = 300.00

TERM_MINS = 100

MTM_MINS_COST = 0.05

TERM_MINS_COST = 0.1

PREPAID_MINS_COST = 0.025

1. Term Contract

class TermContract.**__init__**(self, start: datetime, end: datetime) -> None:

This function begins by calling the parent class of Contract's __init__ to creates the attributes that holds the start given contract. Additionally, this call also creates the bill attribute which is instantiated to None. It then instantiates the enddate attribute based on end and set the currentdate to the start date.

```
def __init__(self, start: datetime, end: datetime) -> None:
    Contract.__init__(self, start)
    self.currentdate = start
    self.enddate = end
```

class TermContract.**cancel_contract**(self) -> float:

This function begins by resetting the start date attribute to None to indicate this contract is not valid.

It then checks whether this contract has been cancelled after the end date of the contract. If it has, the deposit can be safely returned. This means, if the term deposit has been added, you should be removing the deposit from the added fixed cost.

```
    if self.currentdate > self.enddate:
        checkval = self.bill.get_summary()['fixed'] - TERM_DEPOSIT
        checkval = checkval % TERM_MONTHLY_FEE
        if checkval == 0:
            self.bill.add_fixed_cost(-TERM_DEPOSIT)
```

If it has been cancelled before the enddate, the term deposit is forfeited and remains a fixed cost. The total cost of the bill is then returned:

```
    return self.bill.get_cost()
```

class TermContract.**new_month**(self, month: int, year: int, bill: Bill) -> None:

For a term contract to be valid, a term deposit must be set. Thus, a check for if the current bill is None and the starting date is the current date of the event, then it must be the first month of the contract. If so, add the TERM_DEPOSIT into the fixed cost of the new bill.

```
def new_month(self, month: int, year: int, bill: Bill) -> None:
    if self.bill is None and (month, year) == (self.start.month,
self.start.year):
        bill.add_fixed_cost(TERM_DEPOSIT)
```

Afterwards, set the new bill to the contract's newest bill and begin configuring it. The cost of the contract's per month must be added as well as assigning the correct rate of TERM_MINS_COST.

```
    self.bill = bill
    self.bill.add_fixed_cost(TERM_MONTHLY_FEE)
    self.bill.set_rates("TERM", TERM_MINS_COST)
```

Additionally, we have to always update the date so that we know whether we have passed the enddate of the term contract.

```
    self.currentdate = datetime.date(year, month, 1)
```

class TermContract.**bill_call**(self, call: Call) -> None:

When billing the call, two factors must be considered. The number of free minutes remaining, and the duration of the call. We can represent these variables like the following:

```
def bill_call(self, call: Call) -> None:
    callminutes = ceil(call.duration / 60.0)
    freeminutes = self.bill.get_summary()["free_mins"]
```

In this case, callminutes is the duration of the call in minutes while the bill's billed_min attribute is a representation of how many minutes have been used up. For example, if the billed_min was equal to 10, then 10 minutes has been used up.

This gives us two main cases to consider. If the amount of TERM_MINS - freeminutes which represents the amount of free minutes remaining is greater than or equal callminutes, then it should always try to use up the remaining free minutes.

```
    if freeminutes + callminutes <= TERM_MINS:
        self.bill.add_free_minutes(callminutes)
        callminutes = 0
```

If there are still some freeminutes remaining but is not enough to cover up the entire duration of the call, then the duration will first add up the remaining free minutes to the bill's free_min attribute before calculating how much of the call must actually be billed for.

```
    elif freeminutes + callminutes > TERM_MINS:
        timechange = TERM_MINS - freeminutes
        self.bill.add_free_minutes(timechange)
        callminutes -= timechange
```

After both cases, notice how callminutes have been modified. Callminutes has been modified in such a way that it maximizes the freeminutes usage meaning the rest can now be billed normally.

```
    self.bill.add_billed_minutes(callminutes)
```

2. Month to Month (MTM) Contract

class MTMContract.**__init__**(self, start: datetime) -> None:

This function is entirely inherited from the Contract.__init__ without any override.

class MTMContract.**cancel_contract**(self) -> float:

This function is entirely inherited from the Contract.cancel_contract without any override.

class MTMContract.**new_month**(self, month: int, year: int, bill: Bill) -> None:

This function sets the new bill to the contract's newest bill and begin configuring it. The cost of the contract's per month must be added as well as assigning the correct rate of MTM_MINS_COST.

```
def new_month(self, month: int, year: int, bill: Bill) -> None:
    self.bill = bill
    self.bill.add_fixed_cost(MTM_MINS_COST)
    self.bill.set_rates("MTM", MTM_MONTHLY_FEE)
```

class MTMContract.**bill_call**(self, call: Call) -> None:

This function is entirely inherited from the Contract.bill_call without any override.

3. Prepaid Contract

class PrepaidContract.**__init__**(self, start: datetime, balance: float) -> None:

This function begins by calling the parent class of Contract's __init__ to creates the attributes that holds the start given contract. Additionally, this call also creates the bill attribute which is instantiated to None. A private balance attribute is then setup to store the amount the user owes. This value at the initialization should be zero or negative as that would indicate the customer has extra credit.

```
def __init__(self, start: datetime, balance: float) -> None:
    Contract.__init__(self, start)
    self._balance = -balance
```

class PrepaidContract.**cancel_contract**(self) -> float:

This first resets the start attribute back to None. Afterwards, it would return the amount owed in relation to this contract. If there is any extra credit still on the bill, that amount is forfeited.

```
    return max(0, self.bill.get_cost())
```

class PrepaidContract.**new_month**(self, month: int, year: int, bill: Bill) -> None:

If this is not the first month of the year, then the new bill will have a new fixed cost based on the previous months bill. This value gets the amount of minutes total used cost last month. When added to the private balance, if it is greater than -10$ meaning they have less than 10 credits, an additional 25 credits should be added. This new balance will be the amount owed at the start of the new month.

```
currentcost = self._balance
if self.bill is not None:
    # Get the amount of minutes used cost last month
    addon = self.bill.get_cost() - self.bill.get_summary()['fixed']
    currentcost += addon
    if -10 < currentcost:
        currentcost -= 2
self._balance = currentcost
```

In addition, the rates should be set to the bill. These values should be based on PREPAID_MINS_COST with a contract type of “PREPAID”. The bill should now add the fixed cost of the new balance.

class PrepaidContract.**bill_call**(self, call: Call) -> None:

This function is entirely inherited from the Contract.bill_call without any override.