Contract Class Code Documentation

Alex Kaminsky
February 16th, 2025

Getting Started

As **Contract** is an abstract class, it cannot be instantiated directly. Developers must instead use its concrete subclasses: **MTMContract** (Month-to-Month), **TermContract**, and **PrepaidContract**. While these subclasses inherit from **Contract**, they implement fundamentally different behaviours - particularly in payment calculation, contract duration, and termination rules. These differences will be explained with nuance in the latter sections.

1. MTMContract (Month-to-month)

1.1 Intro

An **MTMContract** is a **Contract** with no end date and no initial term deposit. This type of contract has higher rates for calls than a term contract, and comes with no free minutes included, but also involves no term commitment.

1.2 Attributes

The **MTMContract** utilizes the attributes **start** (`datetime.date`) , and **bill** (`Bill`) from its parent class **Contract**, meaning it does not have any unique attributes. Thus, making a dedicated `__init__()` method would be redundant.

1.3 Instantiation

The following example illustrates how an **MTMContract** must be instantiated. Note, it is essential to use a valid instance of `datetime.date` as the sole argument.

```
# Instantiate MTMContract with datetime.date parameter <start>

mtm_contract = MTMContract(2017, 12, 25)
```

1.4 Methods

The following defined methods improve the overall usability and functionality of the **MTMContract** class. It is important to note that `new_month()` is the only method that overrides its base method in Contract. All other methods are inherited with no overrides.

1.4.1 `new_month()` → `None`

Void method that advances the contract to a new month. Moves the contract forward to the specified **month** (`int`) and **year** (`int`). After advancing, the method stores the provided **bill** (`Bill`) in the contract and updates the rate per minute (`MTM_MINS_COST`) and the fixed monthly fee (`MTM_MONTHLY_FEE`) accordingly. This method should be used when the contact must be advanced to a new month.

Below is a demonstration of how `new_month()` should be used:

```
# using the mtm_contract declared previously

mtm_contract.new_month(month: int, year: int, bill: Bill)
# Successfully pushed contract to a new month
```

1.4.2 `bill_call()` → `None`

Void method from the base class **Contract** that adds a given **call** (`Call`) to the contract bill. This method should only be used when a call event needs to be billed onto the contract after the appropriate event handling.

Below is a demonstration of how `bill_call()` should be used:

```
# using the mtm_contract declared previously

mtm_contract.bill_call(call: Call)
# Successfully added <call> to the contract
```

1.4.3 `cancel_contract()` → `float`

Method from the base class **Contract** that returns the amount owed in order to close the phone line associated with this contract. As seen in the code, a bill must have already been created for the month and year when this contract is being cancelled. Note, this method does not take any arguments.

Below is a demonstration of how `cancel_contract()` works:

```
# using the mtm_contract declared previously

>>> mtm_contract.cancel_contract()
150.00
# $150 owed in order to cancel the associated contract
```

2. PrepaidContract (Month-to-month)

2.1 Intro

A **PrepaidContract** has a start date but no end date and includes no minutes by default. Its balance represents the amount owed, with a negative balance indicating prepaid credit. Customers must make an initial prepayment, but the amount is flexible.

2.2 Attributes

The **PrepaidContract class** utilizes the attributes **start** (`datetime.date`) , and **bill** (`Bill`) from its parent class **Contract**, however, it has a unique attribute **balance** (`float`) representing the balance owed on a given contract.

2.3 Instantiation

The following example illustrates how an **MTMContract** must be instantiated. Note, it is essential to use a valid instance of `datetime.date` as the sole argument.

```
# Instantiate MTMContract with datetime.date parameter <start>

mtm_contract = MTMContract(2017, 12, 25)
```

1.4 Methods

The following defined methods improve the overall usability and functionality of the **MTMContract** class. It is important to note that `new_month()` is the only method that overrides its base method in Contract. All other methods are inherited with no overrides.

1.4.1 `new_month()` → `None`

Void method that advances the contract to a new month. Moves the contract forward to the specified **month** (`int`) and **year** (int). After advancing, the method stores the provided **bill** (`Bill`) in the contract and updates the rate per minute (`MTM_MINS_COST`) and the fixed monthly fee (`MTM_MONTHLY_FEE`) accordingly. This method should be used when the contact must be advanced to a new month.

Below is a demonstration of how `new_month()` should be used:

```
# using the mtm_contract declared previously

mtm_contract.new_month(month: int, year: int, bill: Bill)
# Successfully pushed contract to a new month
```

1.4.2 `bill_call()` → `None`

Void method from the base class **Contract** that adds a given **call** (`Call`) to the contract bill. This method should only be used when a call event needs to be billed onto the contract after the appropriate event handling.

Below is a demonstration of how `bill_call()` should be used:

```
# using the mtm_contract declared previously

mtm_contract.bill_call(call: Call)
# Successfully added <call> to the contract
```

1.4.3 `cancel_contract()` → `float`

Method from the base class **Contract** that returns the amount owed in order to close the phone line associated with this contract. As seen in the code, a bill must have already been created for the month and year when this contract is being cancelled. Note, this method does not take any arguments.

Below is a demonstration of how `cancel_contract()` works:

```
# using the mtm_contract declared previously

>>> mtm_contract.cancel_contract()
150.00
# $150 owed in order to cancel the associated contract
```