Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
Факультет Безопасности Информационных Технологий

Дисциплина
«Управление мобильными устройствами»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
«Обработка и тарификация CDR (Call Detail Record)»
Вариант 2

Выполнила:
Студентка группы N3350
Шкарева Алена Дмитриевна

Проверил:
доцент ФБИТ,
Федоров Иван Романович

Санкт-Петербург
2020

Цель работы:

Изучение биллинговой системы и тарификации звонков.

Задачи:

Реализация простейшего правила тарификации для услуг типа “Телефония” по длительности разговора и “СМС” по общему количеству.

Реализация:

Для реализации программы был выбран язык Python.
Классы ‘Coefficient’ были реализованы, чтобы для каждой подуслуги использования телефонии (исходящие и входящие звонки, СМС) можно было задать собственные правила расчета.
Классы типа ‘Rule’ – правила расчета также имеют возможные типы – по количеству, по периоду, а также базовый для расчета записей, не входящих в дополнительные правила.
Класс ‘Tariff’ производит подготовку записей для расчетов каждой из подуслуг.
Класс ‘BaseBilling’ нужен для хранения информации об абоненте, его номер, записи и используемый тариф.
Класс ‘DataManager’ используется для перевода данных об использовании услугой разных абонентов к тому виду, который используется программой.

Из сторонних библиотек используется библиотека pandas.

Выполнение программы:

```
C:\Users\Gammilen\projects\labs\умy\1>python -m program
Результат:  340.64
```

Выводы

Во время выполнения данной лабораторной работы были реализованы простейшие правила тарификации.

Исходный код

https://github.com/gammilen/mobile_device_management

program.py

```
from billing import BaseBilling
from tariffs import Tariff
```

```
from coefficients import IncomingCoefficient, OutgoingCoefficient,
SMSCoefficient
from rules import BasicRule
from data_manager import DataManager

def calculate_tel():
    i = IncomingCoefficient()
    i.set_base_rule(BasicRule(1))
    o = OutgoingCoefficient()
    o.set_base_rule(BasicRule(3))
    s = SMSCoefficient()
    s.set_base_rule(BasicRule(1))
    t = Tariff(i, o, s)
    b = BaseBilling("968247916")
    b.load_tariff(t)
    b.load_records(DataManager.get_data())
    return b.count()

def main():
    print("Результат: ", calculate_tel())

if __name__ == "__main__":
    main()
```

data_manager.py

```
import pandas as pd
import os

file_name = os.path.join(os.path.dirname(os.path.abspath(__file__)),
"data.csv")

class DataManager:
    @classmethod
    def get_data(cls, f_name=file_name):
        data = pd.read_csv(f_name)
        data['msisdn_origin'] = data['msisdn_origin'].astype(str)
        data['msisdn_dest'] = data['msisdn_dest'].astype(str)
        return data
```

coefficients.py

```
from abc import ABC, abstractmethod

class Restriction:
    def __init__(self, *args, **kwargs):
        self.count = {
            "free finite": True,
            "free amount": 1,
        }
        self.period = {
            "free finite": False,
            "free amount": None,
        }
        self.validation = {
            "free-count": self.free_count,
            "free-period": self.free_period,
            "non-free-count": self.non_free_count,
            "non-free-period": self.non_free_period,
        }

    #вызывается при добавлении non-free-count правила
    @staticmethod
    def non_free_count(period_num, free_period_num):
        if period_num != free_period_num:
```

```
            raise ValueError("Нарушен порядок сочетания правил")

    #вызывается при добавлении non-free-period правила
    @staticmethod
    def non_free_period(count_num, free_count_num):
        if count_num != free_count_num:
            raise ValueError("Нарушен порядок сочетания правил")

    def free_count(self, free_count_num):
        if self.count['free finite']:
            if free_count_num != self.count['free amount']:
                raise ValueError(
                    "Нарушен порядок использования"
                    "правил типа 'количество' с отсутствием платы")

    def free_period(self, free_period_num):
        if self.period['free finite']:
            if free_period_num != self.count['free amount']:
                raise ValueError(
                    "Нарушен порядок использования"
                    "правил типа 'количество' с отсутствием платы")

    #TODO Проверка на пересечение
    # периоды в формате (начало, конец)
    def period(self, periods):
        pass


class Coefficient(ABC):
    def __init__(self, *args, **kwargs):
        self.restriction = Restriction()
        self.free_period_num = 0
        self.free_count_num = 0
        self.period = dict()
        self.count = dict()

    def set_base_rule(self, rule):
        self.base = rule


class CallingCoefficient(Coefficient):
    def __init__(self, *args, **kwargs):
        super(CallingCoefficient, self).__init__(*args, **kwargs)
        self.free_period_num = 0
        self.free_count_num = 0
        self.period = dict()
        self.count = dict()

    def add_period_rule(self, rule):
        if rule.k == 0:
            self.restriction.validation['free-
period'](self.free_period_num+1)
            self.free_period_num += 1
            self.update_rules(rule, "period", True)
        else:
            self.restriction.validation['non-free-period'](
                len(self.count), self.free_count_num)
            self.update_rules(rule, "period", False)

    def add_count_rule(self, rule):
        if rule.k == 0:
            self.restriction.validation['free-count'](self.free_count_num+1)
            self.free_count_num += 1
            self.update_rules(rule, "count", True)
```

```
            else:
                self.restriction.validation['non-free-count'](
                    len(self.period), self.free_period_num)
                self.update_rules(rule, "count", False)

    def has_free(self):
        return self.free_count_num > 0 or self.free_period_num > 0

    def update_rules(self, _rule, r_type, free):
        if r_type == "period":
            if free is True:
                self.period["free"].append(_rule)
            else:
                self.period["non-free"].append(_rule)
        elif r_type == "count":
            if free is True:
                self.count["free"].append(_rule)
            else:
                self.count["non-free"].append(_rule)

    #TODO Учет прочих правил при расчете
    def calculate(self, recordset):
        # return self.base * (sum of out calling)
        return self.base.k * recordset["call_duration"].sum()


class IncomingCoefficient(CallingCoefficient):
    pass


class OutgoingCoefficient(CallingCoefficient):
    pass


class SMSCoefficient(Coefficient):
    def __init__(self, *args, **kwargs):
        super(SMSCoefficient, self).__init__(*args, **kwargs)
        self.free_count_num = 0
        self.count = dict()

    def add_count_rule(self, rule):
        if rule.k == 0:
            self.restriction.validation['free-count'](self.free_count_num+1)
            self.free_count_num += 1
            self.update_rules(rule, "count", True)
        else:
            self.restriction.validation['non-free-count'](
                len(self.period), self.free_period_num)
            self.update_rules(rule, "count", False)

    def has_free(self):
        return self.free_count_num > 0

    def update_rules(self, _rule, r_type, free):
        if r_type == "count":
            if free is True:
                self.count["free"].append(_rule)
            else:
                self.count["non-free"].append(_rule)

    #TODO Учет прочих правил при расчете
    def calculate(self, recordset):
        # return self.base * (sum of sms)
        return self.base.k * recordset["sms_number"].sum()
```

billing.py

```
class BaseBilling:
    def __init__(self, subscriber, *args, **kwargs):
        self.records = None
        self.tariff = None
        self.subscriber = subscriber

    def load_records(self, recordset):
        self.records = recordset

    def load_tariff(self, tariff):
        tariff.load_subscriber(self.subscriber)
        self.tariff = tariff

    def count(self):
        if not self.tariff:
            raise AttributeError("Отсутствует тариф для расчета")
        return self.tariff.calculate(self.records)
```

tariffs.py

```
class Tariff:
    def __init__(self, incoming, outgoing, sms):
        self.coeffs = {
            'incoming': incoming,
            'outgoing': outgoing,
            'sms': sms
        }

    def load_subscriber(self, sub):
        self.subscriber = sub

    def _filter_incoming(self, recordset):
        return recordset[recordset["msisdn_dest"] ==
self.subscriber][["timestamp","call_duration"]]

    def _filter_outgoing(self,recordset):
        return recordset[recordset["msisdn_origin"] ==
self.subscriber][["timestamp","call_duration"]]

    def _filter_sms(self, recordset):
        return recordset[recordset["msisdn_origin"] ==
self.subscriber][["timestamp","sms_number"]]

    def _calculate_incoming(self, recordset):
        return self.coeffs["incoming"].calculate(recordset)

    def _calculate_outgoing(self, recordset):
        return self.coeffs["outgoing"].calculate(recordset)

    def _calculate_sms(self, recordset):
        return self.coeffs["sms"].calculate(recordset)

    def calculate(self, recordset):
        result = 0
        result += self._calculate_incoming(self._filter_incoming(recordset))
        result += self._calculate_outgoing(self._filter_outgoing(recordset))
        result += self._calculate_sms(self._filter_sms(recordset))
        return result
```

rules.py

```
from abc import ABC
```

```python
class Rule(ABC):
    def __init__(self, k, *args, **kwargs):
        self.k = k

class BasicRule(Rule):
    #Просто стабильный множитель
    pass

class PeriodRule(Rule):
    #Ограничения по времени
    #Если не указано то 'до' с наименьшего времени, если 'после', то до
наибольшего времени
    pass

class CountRule(Rule):
    #Множитель зависит от количества
    #Если не указано 'от', то с 0 , если не 'до', то бесконечности
    pass
```