Министерство образования Республики Беларусь Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

### Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

**Отчёт**

### по лабораторной работе № 3

по дисциплине ***«Языки программирования»***

## тема: «Python и ООП. Отношение (взаимосвязь) между классами. Ассоциации и её разновидности: агрегация, композиция и наследование»

|  |  |
| --- | --- |
| Исполнитель: | студент группы 10701118 Воробей Иван Александрович |
| Преподаватель: | Лапанович И.О. |

2018-2019 учебный год

**ЛАБОРАТОРНАЯ РАБОТА № 2**

**Python и ООП. Сокрытие реализации (инкапсуляция)**

#### Цель работы: Изучить механизмы и способы взаимодействия классов и объектов данных классов между собой с использованием разновидностей ассоциации (наследования, агрегации и композиции) и закрепить их на примере проектирования и реализации ООП-программ с использованием языка программирования Python.

#### Основное задание:

#### Спроектировать и реализовать программную систему, которая бы определяла и реализовывала иерархию различных объектов автотранспорта: легковые машины, микроавтобусы, автобусы, грузовые машины, мини-фургоны и т.д., группировала бы все объекты в объекте автопарка и высчитывала бы следующие значения:

####  общее количество пассажиров, которых одновременно может перевезти пассажирский транспорт автопарк;

####  общий вес груза, который могут взять одновременно для перевозки все грузовые машины автопарк;  общее количество топлива, необходимое для заправки всего автопарка машин.

**Индивидуальное задание:**

Произвести рефакторинг программной системы, созданной в предыдущей лабораторной работе, следующим образом:

 классы, описывающие объекты соответствующей предметной области (бизнес объекты), должны быть сведены в иерархическую структуру (произвести, где это необходимо, классификацию типов);

 скрыть наборы объектов, которыми манипулирует логика системы, в соответствующие контейнерные классы; логика системы должны принимать на вход объекты только данных контейнерных классов;

 логика системы должна зависеть только от абстракции, а не от реализации;

 логика системы должна быть реализована внутри соответствующих функциональных классов (к примеру, в виде статических полей), а не в виде отдельных Python-функций.

 добавить в методы бизнес логики проверку входящих объектов на соответствие типа, с которым должна взаимодействовать логика.

**Требования к выполнению:**

1) Программа должна обязательно быть снабжена комментариями на английском языке, в которых необходимо указать краткое предназначение программы, номер лабораторной работы и её название, версию программы, ФИО разработчика, номер группы и дату разработки.

2) Исходный текст классов и демонстрационной программы рекомендуется снабжать комментариями.

3) В отчёте ОБЯЗАТЕЛЬНО привести UML-диаграмму классов, которая демонстрирует классы и объекты приложения, их атрибуты и методы, а также взаимосвязь между ними.

4) Каждый класс должен иметь адекватное осмысленное имя (обычно это имя существительное) и начинаться с заглавной буквы. Имена полей и методов должны начинаться с маленькой буквы и быть также осмысленными (имя метода, который что-то вычисляет, обычно называют глаголом, а поле – именем существительным).

5) Каждый класс необходимо разместить в отдельном модуле, который затем подключается в другом модуле, где происходит создание объекта данного класса и его использование.

6) При проектировании классов необходимо придерживаться принципа единственной ответственности (Single Responsibility Principle), т.е. классы должны проектироваться и реализовываться таким образом, чтобы они были менее завязаны с другими классами при своей работе – они должны быть самодостаточными.

7) Программа для демонстрации работоспособности разработанных классов должна быть снабжена дружелюбным и интуитивно понятным интерфейсом.

8) При разработки кода необходимо придерживаться соответствующего стиля (соглашения по форматированию и именованию), который используется для языка программирования Python.

#### Результаты выполнения основного задания:

#### 

UML-диаграмма для основного задания

#### 

#### Описание предметной области индивидуального задания:

Предметной (проблемной) областью является работа кофейни, подсчет количества заказанного кофе и оплата.

В результате анализа предметной области были выделены следующие классы:

* + покупатель (Customer) – общий класс, описывающий существенные характеристики покупателя и некоторую логику покупки кофе;
  + бариста (Barista) – общий класс, в котором описываются и существенные характеристики бариста, и некоторая логика продажи кофе;
  + Всевышний (God) – утилитный класс для создания списка покупателей, а также представления данного списка в строковом варианте для вывода на консоль.

Ниже на рисунке 1 приведена UML-диаграмма классов заданной предметной (проблемной) области.

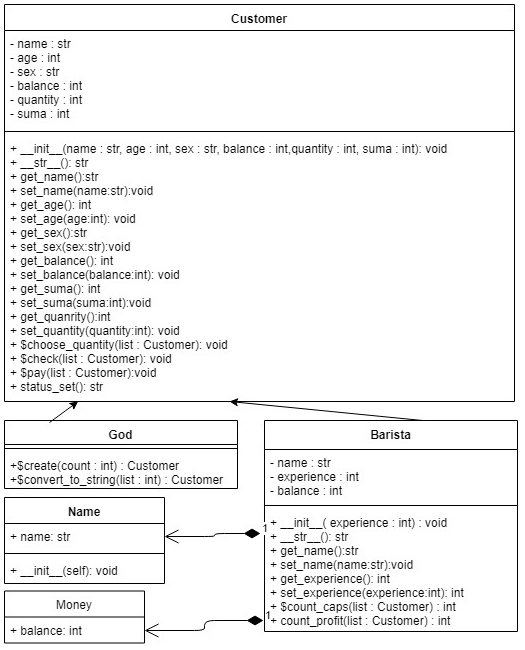
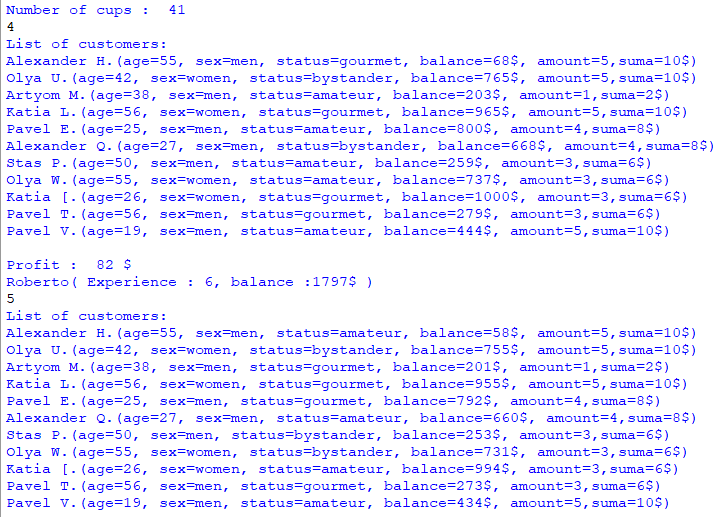


Рисунок 1 – UML-диаграмма предметной области индивидуального задания

#### Результаты выполнения индивидуального задания:





#### Что я освоил в процессе выполнения лабораторной работы (выводы):

#### Я изучил механизмы и способы взаимодействия классов и объектов данных классов между собой с использованием разновидностей ассоциации (наследования, агрегации и композиции) и закрепил их на примере проектирования и реализации ООП-программ с использованием языка программирования Python.

*ПРИЛОЖЕНИЕ A*

#### Листинг исходных кодов классов и программы в целом

#### Файл 3laba.py

#### class Car(object):

#### def \_\_init\_\_(self, passenger, weight, fuel):

#### self.passenger= passenger

#### self.weight=weight

#### self.fuel=fuel

#### class Truck(Car): #inheritance

#### pass

#### class Bus(Car): #inheritance

#### pass

#### class Autopark(object):

#### def \_\_init\_\_(self, car, truck, bus): #aggregation

#### self.car= car

#### self.truck=truck

#### self.bus=bus

#### def sum(self):

#### print("Total number of passengers :{}".format(self.car.passenger+self.truck.passenger+self.bus.passenger))

#### print("Total weight, which all machines can take :{}".format(self.car.weight+self.truck.weight+self.bus.weight))

#### print("Total value of needed fuel :{}".format(self.car.fuel+self.truck.fuel+self.bus.fuel))

#### car=Car(6,20,25)

#### truck=Truck(3,1000,50)

#### bus=Bus(24,70,40)

#### autopark=Autopark(car, truck, bus)

#### autopark.sum()

#### Файл customer.py

#### import random

#### class Customer:

#### def \_\_init\_\_(self, name, age,sex,balance,quantity=0,suma=0):

#### self.\_\_name=name

#### self.\_\_age=age

#### self.\_\_sex=sex

#### self.\_\_balance=balance

#### self.\_\_quantity=quantity

#### self.\_\_suma=suma

#### def get\_name(self):

#### if isinstance(self, Customer):

#### return self.\_\_name

#### def set\_name(self,name):

#### if isinstance(self, Customer):

#### self.\_\_name=name

#### def get\_age(self):

#### if isinstance(self, Customer):

#### return self.\_\_age

#### def set\_age(self,age):

#### if isinstance(self, Customer):

#### self.\_\_age=age

#### def get\_sex(self):

#### if isinstance(self, Customer):

#### return self.\_\_sex

#### def set\_sex(self,sex):

#### if isinstance(self, Customer):

#### self.\_\_sex=sex

#### def get\_balance(self):

#### if isinstance(self, Customer):

#### return self.\_\_balance

#### def set\_balance(self,balance):

#### if isinstance(self, Customer):

#### self.\_\_balance=balance

#### def get\_suma(self):

#### if isinstance(self, Customer):

#### return self.\_\_suma

#### def set\_suma(self, suma):

#### if isinstance(self, Customer):

#### self.\_\_suma=suma

#### def get\_quantity(self):

#### if isinstance(self, Customer):

#### return self.\_\_quantity

#### def set\_quantity(self, quabtity):

#### if isinstance(self, Customer):

#### self.\_\_quantity=quantity

#### @property

#### def status\_set(self):

#### statuses=["amateur","bystander","gourmet"]

#### status=random.choice(statuses)

#### return status

#### def \_\_str\_\_(self):

#### return (self.\_\_name+"(age="+str(self.\_\_age)+

#### ", sex="+self.\_\_sex+ ", status="+

#### self.status\_set+", balance="+str(self.\_\_balance)+"$"+

#### ", amount="+str(self.\_\_quantity)+

#### ",suma="+str(self.\_\_suma)+"$"+")")

#### @staticmethod

#### def choose\_quantity(customers\_list):

#### cus=Customer

#### for cus in customers\_list:

#### if isinstance(cus, Customer):

#### cus.\_\_quantity=random.randint(1,5)

#### @staticmethod

#### def check(customers\_list):

#### cus=Customer

#### for cus in customers\_list:

#### if isinstance(cus, Customer):

#### cus.\_\_suma=2\*cus.\_\_quantity

#### @staticmethod

#### def pay(customers\_list):

#### cus=Customer

#### for cus in customers\_list:

#### if isinstance(cus, Customer):

#### cus.\_\_balance=cus.\_\_balance-cus.\_\_suma

**Файл barista.py**

from customer import Customer

from c\_name import Name

from money import Money

class Barista:

def \_\_init\_\_(self, experience=1):

origin\_name=Name()#composition

self.\_\_name=origin\_name.name#object(barista) has name

self.\_\_experience=experience

self.\_\_balance=Money.balance#composition, object(barista) has money

def get\_name(self):

if isinstance(self, Barista):

return self.\_\_name

def set\_name(self,name):

if isinstance(self, Barista):

self.\_\_name=name

def get\_experience(self):

if isinstance(self, Barista):

return self.\_\_experience

def set\_experience(self, experience):

if experience>=0:

if isinstance(self, Barista):

self.\_\_experience=experience

def \_\_str\_\_(self):

return(self.\_\_name+"( Experience : "+

str(self.\_\_experience)+

", balance :"+str(self.\_\_balance)+"$ )")

@staticmethod

def count\_cups(customers\_list):

c=0

cus=Customer

for cus in customers\_list:

if isinstance(cus, Customer):

c=c+cus.get\_quantity()

return c

def count\_profit(self,customers\_list):

b=0

cus=Customer

for cus in customers\_list:

if isinstance(cus, Customer):

b=b+cus.get\_suma()

self.\_\_balance=self.\_\_balance+b

return b

##### Файл god.py

##### import random

##### from customer import Customer

##### class God:

##### START\_ALPHABET\_WITH\_UPPER\_LETTER=65

##### END\_ALPHABET\_WITH\_UPPER\_LETTER=91

##### @staticmethod

##### def create(count):

##### namesm=["Alexander","Pavel","Artyom","Kirill","Stas"]

##### namesw=["Olya","Liza","Nastya","Katia"]

##### sexes=["men","women"]

##### list\_of\_customers=[]

##### for i in range(count+1):

##### sex=random.choice(sexes)

##### if sex=="men":

##### name=random.choice(namesm)

##### name+=" " + chr(random.randint(

##### God.START\_ALPHABET\_WITH\_UPPER\_LETTER,

##### God.END\_ALPHABET\_WITH\_UPPER\_LETTER))+"."

##### else:

##### name=random.choice(namesw)

##### name+=" " + chr(random.randint(

##### God.START\_ALPHABET\_WITH\_UPPER\_LETTER,

##### God.END\_ALPHABET\_WITH\_UPPER\_LETTER))+"."

##### age=random.randint(16,70)

##### customer=Customer(name,age,sex,balance)

##### list\_of\_customers.append(customer)

##### return list\_of\_customers

##### @staticmethod

##### def convert\_to\_string(list\_of\_customers):

##### string="List of customers:\n"

##### for customer in list\_of\_customers:

##### if isinstance(customer, Customer):

##### string+=str(customer)+"\n"

##### return string

##### Файл main.py

##### from god import God

##### from barista import Barista

##### from customer import Customer

##### def main():

##### customers\_list=God.create(10)

##### barista=Barista(6)

##### print(barista)

##### print('''1)To show cost of a cup and barista info.

##### 2)To show list of customers befor the ordering.

##### 3)To show list of customers with number of caps. Customers make the orders.

##### 4)To show check, and total barista's profit. Customers calculate chekc. Barista calculate profit.

##### 5)To show ramaining balance of customers.

##### 6)To stop the programm.''')

##### while True:

##### ch=int(input())

##### if ch==1:

##### print("Cost of one cap is 2$")

##### print(barista)

##### elif ch==2:

##### print(God.convert\_to\_string(customers\_list))

##### elif ch==3:

##### Customer.choose\_quantity(customers\_list)

##### print(God.convert\_to\_string(customers\_list))

##### print("Number of cups : ",Barista.count\_cups(customers\_list))

##### elif ch==4:

##### Customer.check(customers\_list)

##### print(God.convert\_to\_string(customers\_list))

##### print("Profit : ",barista.count\_profit(customers\_list),"$")

##### print(barista)

##### elif ch==5:

##### Customer.pay(customers\_list)

##### print(God.convert\_to\_string(customers\_list))

##### else:

##### break

##### if \_\_name\_\_=="\_\_main\_\_":

##### main()

##### Файл c\_name.py

##### class Name:

##### def \_\_init\_\_(self):

##### self.name="Roberto"

##### Файл money.py

##### import random

##### class Money: #Abstract class

##### balance=random.randint(1000,2000)