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 № 2

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

## тема: «Python и ООП. Сокрытие реализации (инкапсуляция)»

|  |  |
| --- | --- |
| 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 сокрытия реализации в Python, а также научиться отделять внутреннее представление объекта (метода, класса, модуля, пакета и т.д.) от внешнего (интерфейса) на примере проектирования и реализации ООП-программ с использованием языка программирования Python.

#### Основное задание:

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

 объекты системы (бизнес объекты), созданные на базе соответствующих классов предметной области, скрывали свою реализацию и предоставляли только внешний интерфейс для взаимодействия с ними (реализовать инкапсуляцию данных и соответствующего поведения);

 объекты соответствующих классов-сущностей реализовали инкапсулированный доступ к будущему состоянию объектов через соответствующие свойства: только для чтения, для чтения и записи, только для записи – при необходимости (определяется предметной областью);

 классы-сущности моделируемой системы для первоначальной инициализации объектов содержали соответствующие конструкторы \_\_init\_\_() и соответствующие методы \_\_str\_\_(), которые служат для автоматического строкового представления состояния объектов, на которые указывают соответствующие ссылочные переменные;

 классы системы (обычно, функциональные классы) содержали собственные атрибуты и статические методы, которые должны быть обоснованы и актуальны для всего класса в целом, а не отдельного экземпляра.

#### 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.

#### Описание предметной области:

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.

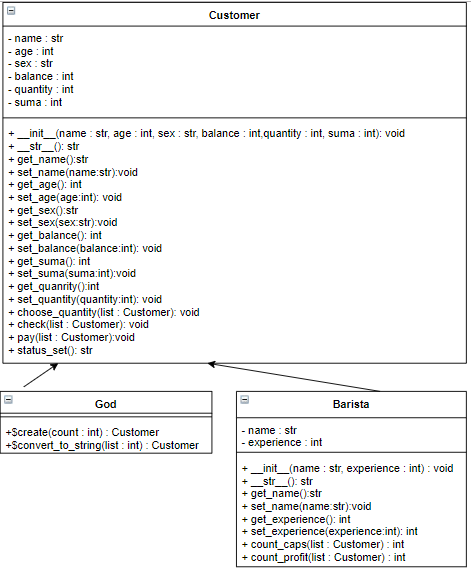


Рисунок 1 – UML-диаграмма предметной области

#### Результаты выполнения основного задания:





#### Ответы на контрольные вопросы:

1. Зачем нужна инкапсуляция? Назовите главный козырь инкапсуляции.

Применяя инкапсуляцию, мы, как бы, возводим крепость, которая защищает данные, принадлежащие объекту, от возможных ошибок. Защита.

1. Где Вы в реальной жизни встречаете инкапсуляцию и на что она влияет?

Черный ящик. Влияет на защиту данных.

1. Является ли инкапсуляция только прерогативой ООП или где-то она уже Вам встречалась при получении опыта программирования?

Инкапсуляция не является только прероготивой ООП, но раньше я с ней не сталкивался.

1. Какими средствами обычно обеспечивается инкапсуляция в ООП мире?

Специальный синтаксис в языках программирования.

1. Как в Python реализуется инкапсуляция на уровне синтаксиса языка и на уровне методологии программирования?

Одиночное подчеркивание

Class Student:

def \_study(self):

print(“Hello”)

вызов

s=Student()

s.\_study()

Двойное подчеркивание

Class Student:

def \_\_study(self):

print(“Hello”)

доступа по имени уже нет, s. \_Student\_\_study()

1. Преимущества и недостатки использования инкапсуляции?

Преимущества: надежность данных, целостность ссылок,сокрытие информации. Недостатки: сложно исправлять ошибки сторонних библиотек, снижается производительность и скорость работы приложения.

1. Что такое свойства в Python? Как реализовать свойства «только для чтения»? А как реализовать свойство «только для записи»?

@property – свойство только для чтения

@(имя поля).setter- только для записи

1. Что такое атрибуты класса?

Параметры этого класса.

1. Зачем нужны статические методы и как их реализовать в Python?

Статический метод никак не связан с состоянием объекта, не использует никакого обращения к полям объекта, следовательно для вызова такого метода не обязательно создавать объект. Достаточно указать имя класса и через точечную нотацию сам метод.

1. В чём концептуальная разница между разными типами методов и функций, которые можно описать внутри класса? Приведите примеры.

* Динамические методы(обычно вызываются на ссылочных переменных то есть на экземплярах класса, но могут быть вызваны и непосредственно на ссылке класса)
* Статические методы(обычно вызываются на ссылке класса, но могут вызываться ичерез ссылочную переменную)
* Обычные функции(вызываются только на ссылке класса)

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

Я зучил механизмы и способы сокрытия реализации в Python, а также научился отделять внутреннее представление объекта (метода, класса, модуля, пакета и т.д.) от внешнего (интерфейса) на примере проектирования и реализации ООП-программ с использованием языка программирования Python.

*ПРИЛОЖЕНИЕ A*

#### Листинг исходных кодов классов и программы в целом 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):*

*return self.\_\_name*

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

*self.\_\_name=name*

*def get\_age(self):*

*return self.\_\_age*

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

*self.\_\_age=age*

*def get\_sex(self):*

*return self.\_\_sex*

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

*self.\_\_sex=sex*

*def get\_balance(self):*

*return self.\_\_balance*

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

*self.\_\_balance=balance*

*def get\_suma(self):*

*return self.\_\_suma*

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

*self.\_\_suma=suma*

*def get\_quantity(self):*

*return self.\_\_quantity*

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

*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)+"$"+")")*

*def choose\_quantity(self, customers\_list):*

*for cus in customers\_list:*

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

*def check(self,customers\_list):*

*for cus in customers\_list:*

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

*def pay(self, customers\_list):*

*for cus in customers\_list:*

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

***Файл barista.py***

*from customer import Customer*

*class Barista:*

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

*self.\_\_name=name*

*self.\_\_experience=experience*

*def get\_name(self):*

*return self.\_\_name*

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

*self.\_\_name=name*

*def get\_experience(self):*

*return self.\_\_experience*

*def set\_experience(self, experience):*

*if experience>=0:*

*self.\_\_experience=experience*

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

*return(self.\_\_name+"( Experience : "+str(self.\_\_experience)+")")*

*def count\_cups(self, customers\_list):*

*c=0*

*for cus in customers\_list:*

*c=c+cus.get\_quantity()*

*return c*

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

*b=0*

*for cus in customers\_list:*

*b=b+cus.get\_suma()*

*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))+"."*

*balance=random.randint(20,1000)*

*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():

##### cus=Customer(1,2,3,4,5,6)

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

##### barista=Barista("Liz", 7)

##### 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:

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

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

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

##### elif ch==4:

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

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

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

##### elif ch==5:

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

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

##### else:

##### break

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

##### main()