Министерство образования Республики Беларусь

Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

Кафедра «Программное обеспечение информационных систем и технологий»

**Отчёт**

***по Лабораторной работе №30***

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

Исполнитель: Колосов Артём Александрович

Группа: 10701219

Преподаватель: Лапанович Ирина Олеговна

2020

**Цели работы:**

Изучить механизмы виртуализации и полиморфного поведения в языке Python, а также практически закрепить всё на примере проектирования и реализации ООП-программ с использованием языка Python.

Основное задание:

− наделить сущности (бизнес-объекты), которыми манипулирует бизнеслогика, соответствующим полиморфным поведением таким образом, чтобы не пришлось переписывать саму бизнес-логику.

Требования к выполнению:

1) Необходимо спроектировать UML-диаграмму взаимодействия классов и объектов создаваемой программной системы. На базе данной UML-диаграммы реализовать рабочее приложение с использованием архитектурного шаблона проектирования MVC.

2) Каждый класс разрабатываемого приложения должен иметь адекватное осмысленное

имя (обычно это имя существительное). Имена полей и методов должны нести также

логический смысл (имя метода, который что-то вычисляет, обычно называют глаголом,

а поле – именем существительным). Имя класса пишется с большой (заглавной) буквы,

а имена методов и переменных – с маленькой (строчной).

3) Соответствующие классы должны группироваться по модулям, которые затем подключаются там, где происходит создание объектов классов и их использование.

4) При проектировании классов необходимо придерживаться принципа единственной ответственности (Single Responsibility Principle), т.е. классы должны проектироваться и

реализовываться таким образом, чтобы они были слабо завязаны с другими классами

при своей работе – они должны быть самодостаточными.

5) При выполнении заданий необходимо по максимуму пытаться разрабатывать универсальный, масштабируемый, легко поддерживаемый и читаемый код.

6) В соответствующих компонентах (классах, функциях) бизнес-логики необходимо

предусмотреть «защиту от дурака».

7) Рекомендуется избегать использования глобальных переменных.

8) Там, где это необходимо, необходимо обеспечить грамотную обработку исключительных ситуаций, которые могут произойти при выполнении разработанной программы.

9) Все действия, связанные с демонстрацией работы приложения, должны быть размещены в главном модуле программы в функции main. При проверки работоспособности

приложения необходимо проверить все тестовые случаи.

10) Программы должны обязательно быть снабжены комментариями на английском языке,

в которых необходимо указать краткое предназначение программы, номер лабораторной

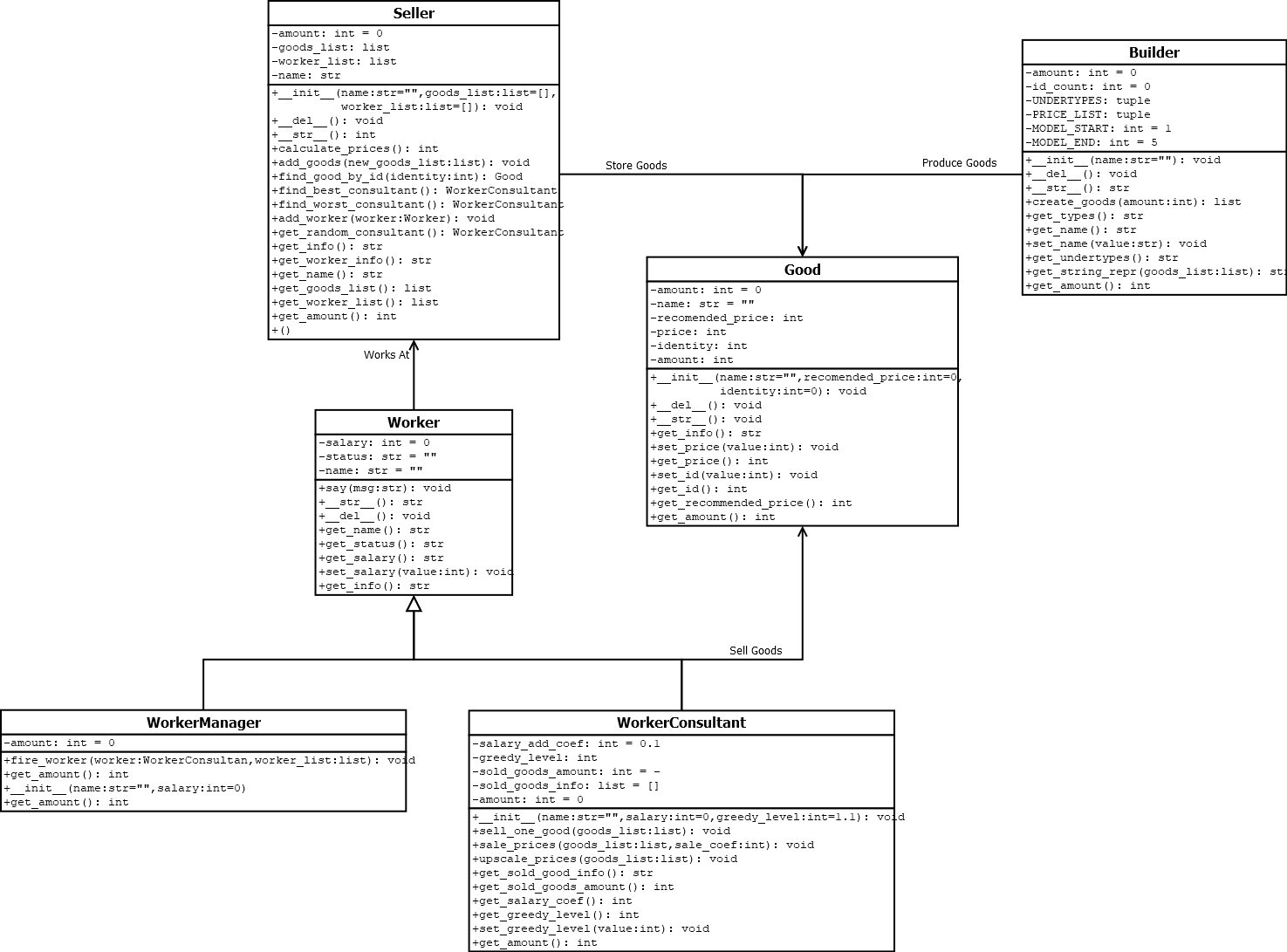
работы и её название, версию программы, ФИО разработчика, номер группы и дату разработки. Исходный текст программного кода и демонстрационной программы рекомендуется также снабжать поясняющими краткими комментариями.

11) Программа должна быть снабжены дружелюбным и интуитивно понятным интерфейсом для взаимодействия с пользователем. Интерфейс программы должен быть на английском языке.

12) При разработке программы придерживайтесь соглашений по написанию кода на Python

(Python Code Convention)

UML Диаграмма программы:



**Рис 30.1 UML-диаграмма программы**

Результаты выполнения основного задания:

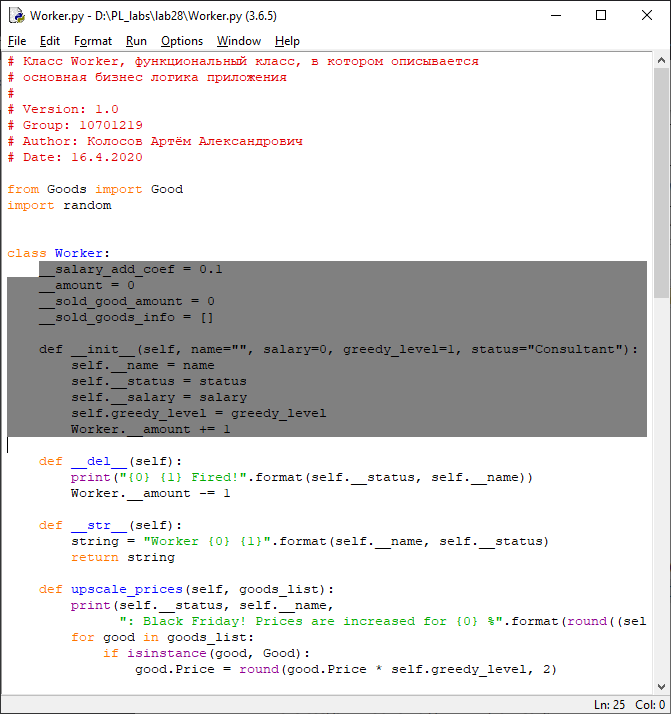


Рис 30.2 Инкапсуляция в исходном коде программы



Рис 30.3 Наследование в исходном коде программы

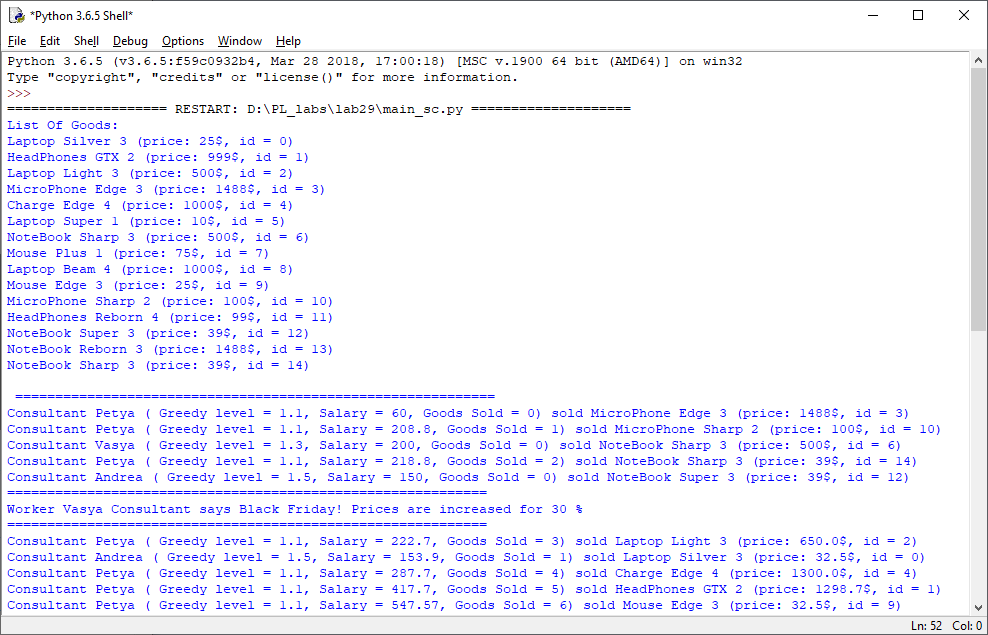


Рис 30.4 Результат работы основного задания

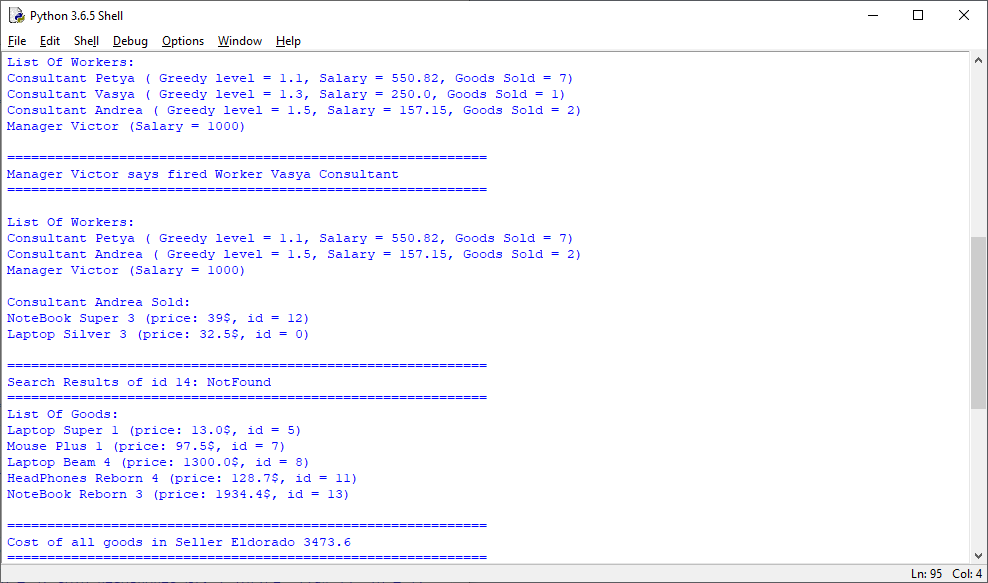


Рис 30.5 Результат работы основного задания

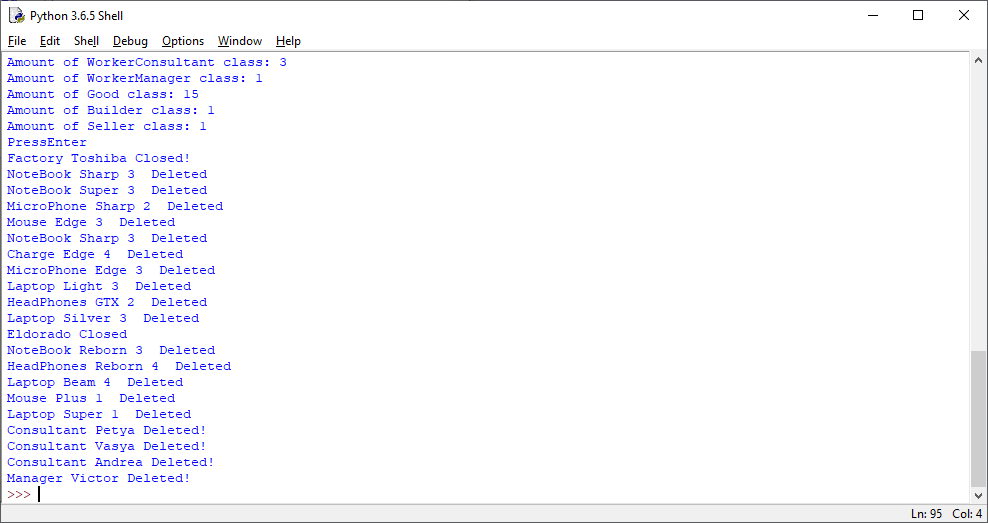


Рис 30.6 Результат работы основного задания (Механизм отслеживания кол-ва созданных объектов и перегрузка метода(деструктора) \_\_del\_\_)

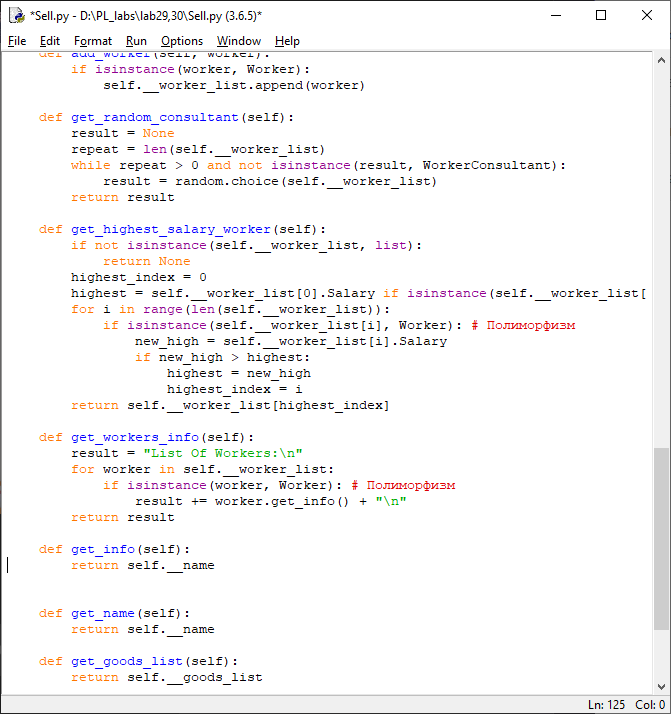


Рис 30.7 Результат работы основного задания (Полиморфизм в коде программы)

Ответы на контрольные вопросы:

1. Что такое полиморфизм?

*Способность логики программы работать с разными типами данных*

2. Какие разновидности полиморфизма бывают в языках программирования?

*Статический*

*Динамический*

3. Как реализуется статический полиморфизм в языке Python?

*Перегрузкой методов*

4. Что такое перегрузка методов (функций или операторов) в языке Python?

*Изменение поведения некоторый операторов, которые взаимодействуют с экземплярами классов.*

5. Как динамическая типизация влияет на реализацию динамического полиморфизма в языке Python?

*Так-как тип данных не известен до выполнения каких либо команд, то поведение может изменится во время выполнения программы. Например: строка кода*

*a\*b даёт разный результат в зависимости от типа a и b.*

*a = 5; b = 5*

*a\*b = 25,*

*a = ‘+’; b = 3*

*a\*b=”+++”*

6. Чем полиморфизм языка Python отличается от многих других языков программирования?

*В Python динамический полиморфизм по сути присутствует в большей части операций*

7. Что такое виртуальный метод? Что такое переопределение методов в языке Python?

*Метод, который можно переопределить в объектах наследниках. Все методы в Python являются виртуальными*

8. Что такое абстрактный класс? А что такое абстрактный метод?

*Это класс, который не предполагает создания своего экземпляра. Абстрактный метод – это метод, который нужно переопределить в наследуемых классах*

9. Можно ли реализовать абстрактный класс (метод) в языке Python? Если да, то как и какой модуль при этом нужно подключать?

*Да, можно. Для этого нужно подключить модуль abc*

10. Что должен всегда делать абстрактный метод в языке Python при его объявлении в базовом классе?

*Возбуждать исключение, которое говорит, что нужно переопределить метод.*

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

Основные столпы объектно-ориентированного программирования. А именно мной были закреплены основы инкапсуляции, абстракции, наследования, полиморфизма. Была создана программа, которая создавалась с учётом всех правил ООП. Также я закрепил своё умение мыслить объектно и переносить эти объекты в программный код, что, несомненно, пригодится мне в будущем в моей профессиональной деятельности.

*ПРИЛОЖЕНИЕ А*

Листинг исходных кодов программ лабораторной работы №29,30

# **Файл Builder.py**

# Класс Builder, утилитный класс для

# создания списка товаров, а также

# представления данного списка в

# строковом варианте для вывода на консоль.

#

# Version: 1.0

# Group: 10701219

# Author: Колосов Артём Александрович

# Date: 16.4.2020

import random

from Goods import Good

class Builder:

\_\_amount = 0

\_\_id\_count = 0

\_\_TYPES = ("Computer", "NoteBook", "Laptop",

"Phone", "Mouse", "Camera",

"MicroPhone", "HeadPhones", "Charge")

\_\_UNDERTYPES = ("Reborn", "Edge", "Plus",

"Extra", "Razr", "Sharp",

"Light", "Image", "GTX",

"Ti", "Gold", "Silver",

"Premium", "Super", "Beam")

\_\_PRICE\_LIST = (100, 500, 50, 699, 1488,

10000, 1000, 75,

25, 999, 99, 10, 39)

\_\_MODEL\_START = 1

\_\_MODEL\_END = 5

def \_\_init\_\_(self, name=""):

if not isinstance(name, str):

raise TypeError("Types of arguments are wrong!")

self.\_\_name = name

Builder.\_\_amount += 1

def \_\_del\_\_(self):

Builder.\_\_amount -= 1

print(self, "Closed!")

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

string = "Factory {0}".format(self.\_\_name)

return string

def create\_goods(self, amount):

goods\_list = []

for i in range(amount):

name = random.choice(self.\_\_TYPES) + " " + random.choice(self.\_\_UNDERTYPES) + " " + \

str(random.randrange(self.\_\_MODEL\_START, self.\_\_MODEL\_END))

price = random.choice(self.\_\_PRICE\_LIST)

goods\_list.append(Good(name, price, Builder.\_\_id\_count))

Builder.\_\_id\_count += 1

return goods\_list

def get\_types(self):

info = ""

for typ in self.\_\_TYPES:

info += typ + "\n"

return info

def get\_name(self):

return self.\_\_name

def set\_name(self, value):

if isinstance(value, str):

self.\_\_name = value

def get\_undertypes(self):

info = ""

for undertype in self.\_\_UNDERTYPES:

info += undertype + "\n"

return info

@classmethod

def get\_amount(cls):

return cls.\_\_amount

@staticmethod

def get\_string\_repr(goods\_list):

string = "List Of Goods:\nNothing!"

if isinstance(goods\_list, list) and goods\_list:

string = "List Of Goods:\n"

for good in goods\_list:

if isinstance(good, Good):

string += good.get\_info()

return string

Name = property(get\_name, set\_name)

# **Файл Goods.py**

# Класс Good, класс-сущность, описывающий существенные характеристики

# товара

#

# Version: 1.0

# Group: 10701219

# Author: Колосов Артём Александрович

# Date: 16.4.2020

class Good:

\_\_amount = 0

def \_\_init\_\_(self, name="", recomended\_price=0, identity=0):

if not (isinstance(name, str) and isinstance(recomended\_price, int) and isinstance(identity, int)):

raise TypeError("Types of arguments are wrong!")

self.\_\_name = name

self.\_\_recomended\_price = recomended\_price

self.\_\_price = recomended\_price

self.\_\_identity = identity

Good.\_\_amount += 1

def \_\_del\_\_(self):

Good.\_\_amount -= 1

print(self.\_\_name, " Deleted")

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

return "{0} (price: {1}$, id = {2})\n".format(self.\_\_name,

self.\_\_price,

self.\_\_identity)

def get\_info(self):

return "{0} (price: {1}$, id = {2})\n".format(self.\_\_name,

self.\_\_price,

self.\_\_identity)

def set\_price(self, value):

self.\_\_price = value

def get\_price(self):

return self.\_\_price

def set\_id(self, value):

self.\_\_identity = value

def get\_id(self):

return self.\_\_identity

def get\_recomended\_price(self):

return self.\_\_recomended\_price

@classmethod

def get\_amount(cls):

return cls.\_\_amount

Price = property(get\_price, set\_price)

Identity = property(get\_id, set\_id)

# **Файл main\_sc.py**

# Главный Файл, который собирает

# все классы и позволяет с ними работать

#

# Version: 1.0

# Group: 10701219

# Author: Колосов Артём Александрович

# Date: 16.4.2020

from Builder import Builder

from Sell import Seller

from Worker import WorkerConsultant, WorkerManager

from Goods import Good

from random import randint

def main():

build = Builder("Toshiba")

amount\_of\_goods = 15

splitter = 60

goods = build.create\_goods(amount\_of\_goods)

eldorado = Seller("Eldorado")

eldorado.add\_goods(goods)

print(build.get\_string\_repr(eldorado.GoodsList), "\n", "=" \* splitter)

cons\_petya = WorkerConsultant("Petya", salary=splitter, greedy\_level=1.1)

cons\_vasya = WorkerConsultant("Vasya", salary=200, greedy\_level=1.3)

cons\_andrea = WorkerConsultant("Andrea", salary=150, greedy\_level=1.5)

man\_victor = WorkerManager("Victor", salary=1000)

eldorado.add\_worker(cons\_petya)

eldorado.add\_worker(cons\_vasya)

eldorado.add\_worker(cons\_andrea)

eldorado.add\_worker(man\_victor)

for i in range(len(eldorado.GoodsList) - 10):

eldorado.get\_random\_consultant().sell\_one\_good(eldorado.GoodsList)

print("=" \* splitter)

eldorado.get\_random\_consultant().upscale\_prices(eldorado.GoodsList)

print("=" \* splitter)

for i in range(len(eldorado.GoodsList) - 5):

eldorado.get\_random\_consultant().sell\_one\_good(eldorado.GoodsList)

print(eldorado.get\_workers\_info())

print("=" \* splitter)

man\_victor.fire\_worker(eldorado.find\_worst\_consultant(), eldorado.WorkerList)

print("=" \* splitter)

print("\n", eldorado.get\_workers\_info(), sep="")

print(eldorado.get\_random\_consultant().get\_sold\_goods\_info())

print("Highest salary", eldorado.get\_highest\_salary\_worker().get\_info())

print("=" \* splitter)

search\_id = randint(0, amount\_of\_goods-1)

print("Search Results of id {0}: {1}".format(search\_id, eldorado.find\_good\_by\_id(search\_id)))

print("=" \* splitter)

print(build.get\_string\_repr(eldorado.GoodsList))

print("=" \* splitter)

print("Cost of all goods in", eldorado, eldorado.calculate\_prices())

print("=" \* splitter)

print("Amount of WorkerConsultant class:", WorkerConsultant.get\_amount())

print("Amount of WorkerManager class:", WorkerManager.get\_amount())

print("Amount of Good class:", Good.get\_amount())

print("Amount of Builder class:", Builder.get\_amount())

print("Amount of Seller class:", Seller.get\_amount())

input("PressEnter")

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

main()

# **Файл Sell.py**

# Класс Seller, который работает с классами Worker, Good

# Класс Seller, общий класс, в котором описываются и существенные

# характеристики работника, товара

# Version: 1.0

# Group: 10701219

# Author: Колосов Артём Александрович

# Date: 16.4.2020

from Goods import Good

from Worker import WorkerConsultant, Worker

import random

class Seller:

\_\_amount = 0

def \_\_init\_\_(self, name="", goods\_list=[], worker\_list=[]):

if not (isinstance(goods\_list, list) and isinstance(name, str) and isinstance(worker\_list, list)):

raise TypeError("Types of arguments are wrong!")

self.\_\_goods\_list = goods\_list[:]

goods\_list.clear()

self.\_\_worker\_list = worker\_list[:]

worker\_list.clear()

self.\_\_name = name

Seller.\_\_amount += 1

def \_\_del\_\_(self):

for worker in self.\_\_worker\_list:

if isinstance(worker, Worker):

del worker

for good in self.\_\_goods\_list:

if isinstance(good, Good):

del good

print(self.\_\_name, "Closed")

Seller.\_\_amount -= 1

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

string = "Seller {0}".format(self.\_\_name)

return string

def calculate\_prices(self):

all\_price = 0

for good in self.\_\_goods\_list:

if isinstance(good, Good):

all\_price += good.Price

return round(all\_price, 2)

def add\_goods(self, new\_goods\_list):

if isinstance(new\_goods\_list, list):

for new\_good in new\_goods\_list:

if isinstance(new\_good, Good):

self.\_\_goods\_list.append(new\_good)

def find\_good\_by\_id(self, identity):

if not (isinstance(identity, int) and self.\_\_goods\_list):

return None

result = "NotFound"

for good in self.\_\_goods\_list:

if isinstance(good, Good):

if good.Identity == identity:

result = good

break

return result

def find\_best\_consultant(self):

if not isinstance(self.\_\_worker\_list, list):

return None

max\_index = 0

maximum = self.\_\_worker\_list[max\_index].SoldGoodsAmount

for i in range(0, len(self.\_\_worker\_list)):

if isinstance(self.\_\_worker\_list[i], WorkerConsultant):

new\_max = self.\_\_worker\_list[i].SoldGoodsAmount

if 0 <= maximum < new\_max:

maximum = new\_max

max\_index = i

return self.\_\_worker\_list[max\_index]

def find\_worst\_consultant(self):

if not isinstance(self.\_\_worker\_list, list):

return None

min\_index = 0

minimum = self.\_\_worker\_list[min\_index].SoldGoodsAmount

for i in range(0, len(self.\_\_worker\_list)):

if isinstance(self.\_\_worker\_list[i], WorkerConsultant):

new\_min = self.\_\_worker\_list[i].SoldGoodsAmount

if 0 <= new\_min < minimum:

minimum = new\_min

min\_index = i

return self.\_\_worker\_list[min\_index]

def add\_worker(self, worker):

if isinstance(worker, Worker):

self.\_\_worker\_list.append(worker)

def get\_random\_consultant(self):

result = None

repeat = len(self.\_\_worker\_list)

while repeat > 0 and not isinstance(result, WorkerConsultant):

result = random.choice(self.\_\_worker\_list)

return result

def get\_highest\_salary\_worker(self):

if not isinstance(self.\_\_worker\_list, list):

return None

highest\_index = 0

highest = self.\_\_worker\_list[0].Salary if isinstance(self.\_\_worker\_list[0], Worker) else None

for i in range(len(self.\_\_worker\_list)):

if isinstance(self.\_\_worker\_list[i], Worker): # Полиморфизм

new\_high = self.\_\_worker\_list[i].Salary

if new\_high > highest:

highest = new\_high

highest\_index = i

return self.\_\_worker\_list[highest\_index]

def get\_workers\_info(self):

result = "List Of Workers:\n"

for worker in self.\_\_worker\_list:

if isinstance(worker, Worker): # Полиморфизм

result += worker.get\_info() + "\n"

return result

def get\_info(self):

return self.\_\_name

def get\_name(self):

return self.\_\_name

def get\_goods\_list(self):

return self.\_\_goods\_list

def get\_worker\_list(self):

return self.\_\_worker\_list

@classmethod

def get\_amount(cls):

return cls.\_\_amount

GoodsList = property(get\_goods\_list)

WorkerList = property(get\_worker\_list)

Name = property(get\_name)

# **Файл Worker.py**

# Класс Worker, функциональный класс, в котором описывается

# основная бизнес логика приложения

#

# Version: 1.0

# Group: 10701219

# Author: Колосов Артём Александрович

# Date: 16.4.2020

from Goods import Good

import random

class Worker:

def \_\_init\_\_(self, name="", salary=0, status=""):

self.\_\_name = name

self.\_\_salary = salary

self.\_\_status = status

def say(self, msg):

print(self.\_\_status, self.\_\_name, msg)

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

return self.\_\_status + " " + self.\_\_name

def \_\_del\_\_(self):

print(self.\_\_name, self.\_\_status, "Fired")

def get\_name(self):

return self.\_\_name

def get\_status(self):

return self.\_\_status

def get\_salary(self):

return self.\_\_salary

def set\_salary(self, value):

self.\_\_salary = value

def get\_info(self):

return self.\_\_name + self.\_\_status

Name = property(get\_name)

Status = property(get\_status)

Salary = property(get\_salary, set\_salary)

class WorkerManager(Worker):

\_\_amount = 0

def \_\_init\_\_(self, name="", salary=0):

if not (isinstance(name, str) and isinstance(salary, int)):

raise TypeError("Types of arguments are wrong!")

super().\_\_init\_\_(name, salary, "Manager")

WorkerManager.\_\_amount += 1

def \_\_del\_\_(self):

print("{0} {1} Deleted!".format(self.Status, self.Name))

WorkerManager.\_\_amount -= 1

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

string = "{0} {1}".format(self.Status, self.Name)

return string

def say(self, msg):

print(self, "says", msg)

def get\_info(self):

return "{0} {1} (Salary = {2})".format(self.Status,

self.Name,

round(self.Salary, 2))

def fire\_worker(self, worker, worker\_list):

if isinstance(worker\_list, list):

if isinstance(worker, WorkerConsultant):

if worker in worker\_list:

worker\_list.remove(worker)

self.say("fired {0}".format(worker))

@classmethod

def get\_amount(cls):

return cls.\_\_amount

class WorkerConsultant(Worker):

\_\_amount = 0

\_\_salary\_add\_coef = 0.1

def \_\_init\_\_(self, name="", salary=0, greedy\_level=1.1):

if not (isinstance(name, str) and isinstance(salary, int) and isinstance(greedy\_level, float)):

raise TypeError("Types of arguments are wrong!")

super().\_\_init\_\_(name, salary, "Consultant")

self.\_\_greedy\_level = greedy\_level

self.\_\_sold\_goods\_info = []

self.\_\_sold\_good\_amount = 0

WorkerConsultant.\_\_amount += 1

def \_\_del\_\_(self):

print("{0} {1} Deleted!".format(self.Status, self.Name))

WorkerConsultant.\_\_amount -= 1

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

string = "Worker {0} {1}".format(self.Name, self.Status)

return string

def say(self, msg):

print(self, "says", msg)

def sell\_one\_good(self, goods\_list):

if goods\_list:

delete\_index = random.randint(0, len(goods\_list) - 1)

if isinstance(goods\_list[delete\_index], Good):

good = goods\_list.pop(delete\_index)

print(self.get\_info(), "sold", good.get\_info(), end='')

self.Salary += self.\_\_salary\_add\_coef \* good.Price

self.\_\_sold\_good\_amount += 1

self.\_\_sold\_goods\_info.append(good.get\_info())

def sale\_prices(self, goods\_list, sale\_coef=0.5):

if 0 < sale\_coef < 1:

self.say("Sales!!! Decreasing prices for {0} %".format(round((1 - sale\_coef) \* 100)))

for good in goods\_list:

if isinstance(good, Good):

good.Price = round(good.Price \* sale\_coef, 2)

else:

self.say("What?")

def upscale\_prices(self, goods\_list):

self.say(msg="Black Friday! Prices are increased for {0} %".format(round((self.\_\_greedy\_level - 1) \* 100)))

for good in goods\_list:

if isinstance(good, Good):

good.Price = round(good.Price \* self.\_\_greedy\_level, 2)

def get\_sold\_goods\_info(self):

result = "{0} {1} Sold Nothing".format(self.Status, self.Name)

if isinstance(self.\_\_sold\_goods\_info, list) and self.\_\_sold\_goods\_info:

result = "{0} {1} Sold:\n".format(self.Status, self.Name)

for info in self.\_\_sold\_goods\_info:

if isinstance(info, str):

result += info

return result

def get\_info(self):

return "{0} {1} ( Greedy level = {2}, Salary = {3}, Goods Sold = {4})".format(self.Status,

self.Name,

self.\_\_greedy\_level,

round(self.Salary, 2),

self.\_\_sold\_good\_amount)

def get\_sold\_goods\_amount(self):

return self.\_\_sold\_good\_amount

def get\_salary\_coef(self):

return self.\_\_salary\_add\_coef

def get\_greedy\_level(self):

return self.\_\_greedy\_level

def set\_greedy\_level(self, value):

if 0 < value < 3:

self.\_\_greedy\_level = value

def set\_salary\_coef(self, value):

if 0 < value <= 1:

self.\_\_salary\_add\_coef = value

@classmethod

def get\_amount(cls):

return cls.\_\_amount

SoldGoodsAmount = property(get\_sold\_goods\_amount)

SalaryCoef = property(get\_salary\_coef, set\_salary\_coef)

GreedyLevel = property(get\_greedy\_level, set\_greedy\_level)