Министерство образования Республики Беларусь

Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

Кафедра «Программное обеспечение информационных систем и технологий»

**Отчёт**

***по Лабораторным работам №27,28,29,30***

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

Исполнитель: Колосов Артём Александрович

Группа: 10701219

Преподаватель: Лапанович Ирина Олеговна

2020

**Цели работы:**

1. Научиться грамотно анализировать предметную область и с помощью абстракции (абстрагирования) выделять существенные детали, на базе которых в дальнейшем проектируются классы и создаются объекты будущей программной системы согласно методологии ООП.
2. Изучить механизмы и способы сокрытия реализации в языке Python, а также научиться отделять внутреннее представление объекта (метода, класса, модуля, пакета и т.д.) от внешнего на примере проектирования и реализации ООП-программ с использованием языка Python.
3. Изучить механизмы и способы повторного использования кода в языке Python, а также организацию способов взаимодействия объектов (классов) между собой посредством связей (ассоциация, наследование, агрегация, композиция и делегирование); закрепить всё вышеизложенное на примере проектирования и реализации ООП-программ с использованием языка Python.
4. 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:

Необходимо произвести небольшую модернизацию предметной области и программы, созданных в предыдущей лабораторной работе, следующим образом:

− расширить набор атрибутов основной сущности предметной области (их может быть и больше); − перегрузить соответствующие конструкторы и добавить деструктор в соответствующие классы;

− проанализировать доступные способы и средства первоначальной инициализации состояния объектов в языке Python;

− на уровне всей программы добавить глобальный механизм отслеживания количества создаваемых объектов предметной области, которыми манипулирует система при выполнении основных расчётов программной системы.

Объекты системы (бизнес-объекты), созданные на базе соответствующих классов предметной области, должны скрывать свою реализацию и предоставлять только внешний интерфейс для взаимодействия (т.е. необходимо реализовать инкапсуляцию данных и поведения);

− ввести, где это необходимо, контейнерный класс для хранения набора бизнес-объектов, т.е. инкапсулировать объекты в более высокоуровневый объект на уровне программной системы; логика системы должны принимать на вход объекты только данных контейнерных классов;

− объекты соответствующих классов-сущностей должны реализовать 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 обязательно должны содержать соответствующий метод \_\_str\_\_(), который служит для автоматического строкового представления состояния объекта, на который указывает соответствующая ссылочная переменная;

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

− классы, описывающие объекты соответствующей предметной области (бизнес-объекты), должны быть сведены в иерархическую структуру (произвести, где это необходимо, классификацию типов); к примеру, в предыдущей лабораторной работе была только сущность автомобиль/автотранспорт, а теперь должна быть иерархия автотранспорта с соответствующими характеристиками: легковой автомобиль (седан, универсал, хэтчбэк, …), грузовой автомобиль (фура, самосвал, бетономешалка ,…), пассажирский автомобиль (автобус, микроавтобус, минивэн, …) и т.д., никто никого не ограничивает в фантазиях;

− логика системы должна быть реализована внутри соответствующих функциональных классов; в программе не должно быть отдельных Python-функций кроме главной функции main();

− логика системы и большинство других компонентов должны зависеть преимущественно только от абстракции, а не от реализации;

− необходимо для безопасности выполнения кода добавить по возможности в методы бизнес логики проверку входящих объектов на соответствие типа, с которым должна взаимодействовать данная логика.

− 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

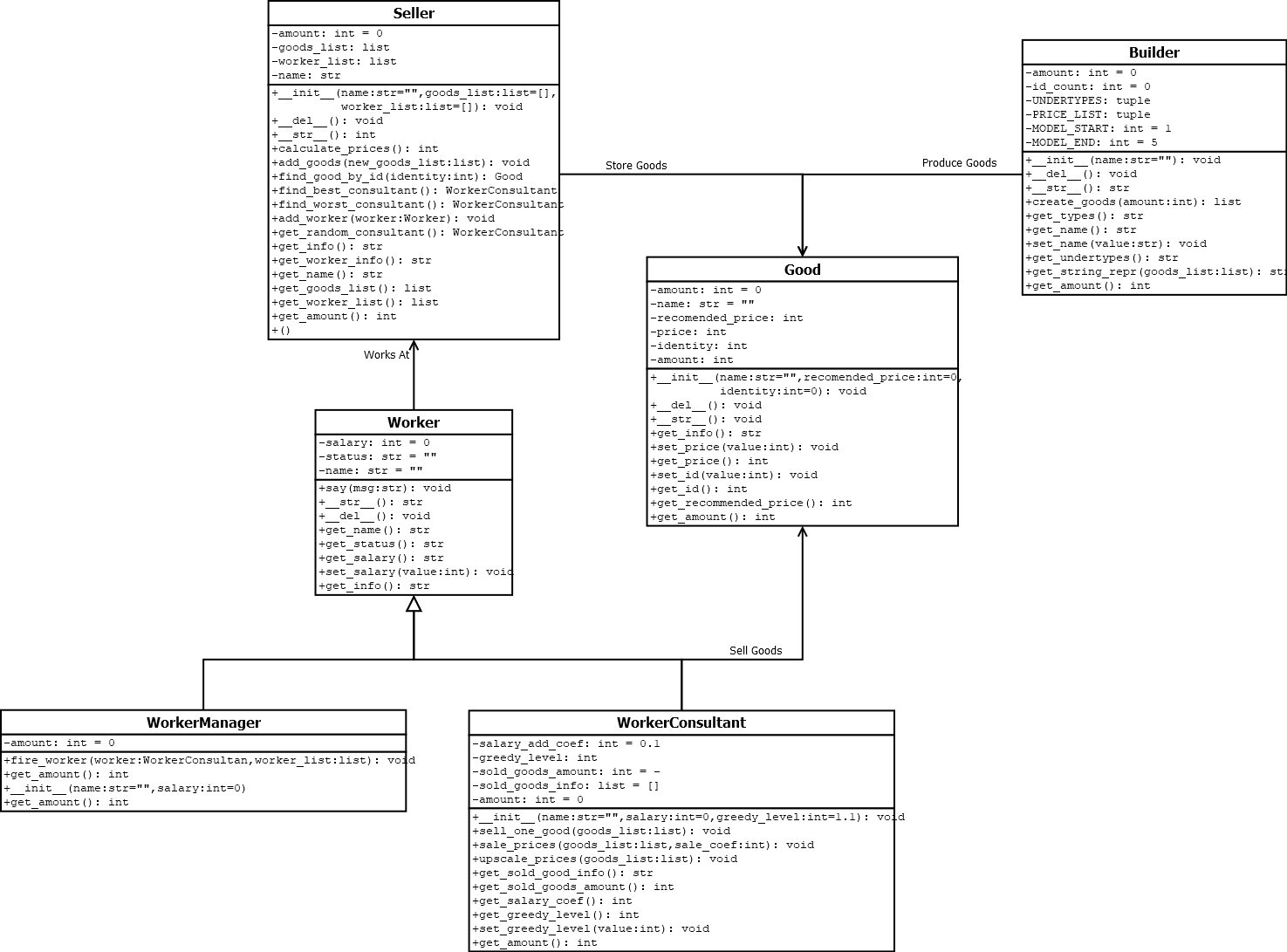
работы и её название, версию программы, ФИО разработчика, номер группы и дату разработки. Исходный текст программного кода и демонстрационной программы рекомендуется также снабжать поясняющими краткими комментариями.

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:

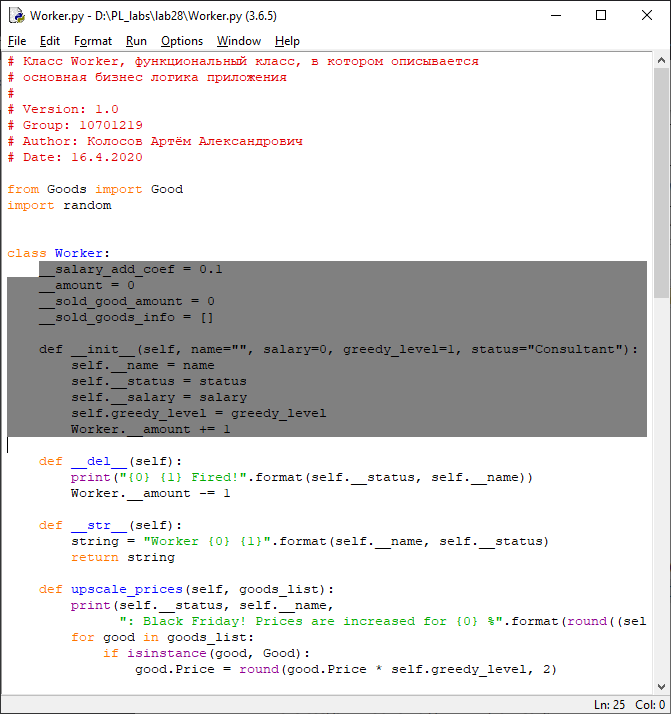


Рис 30.2 Инкапсуляция в исходном коде программы



Рис 30.3 Наследование в исходном коде программы

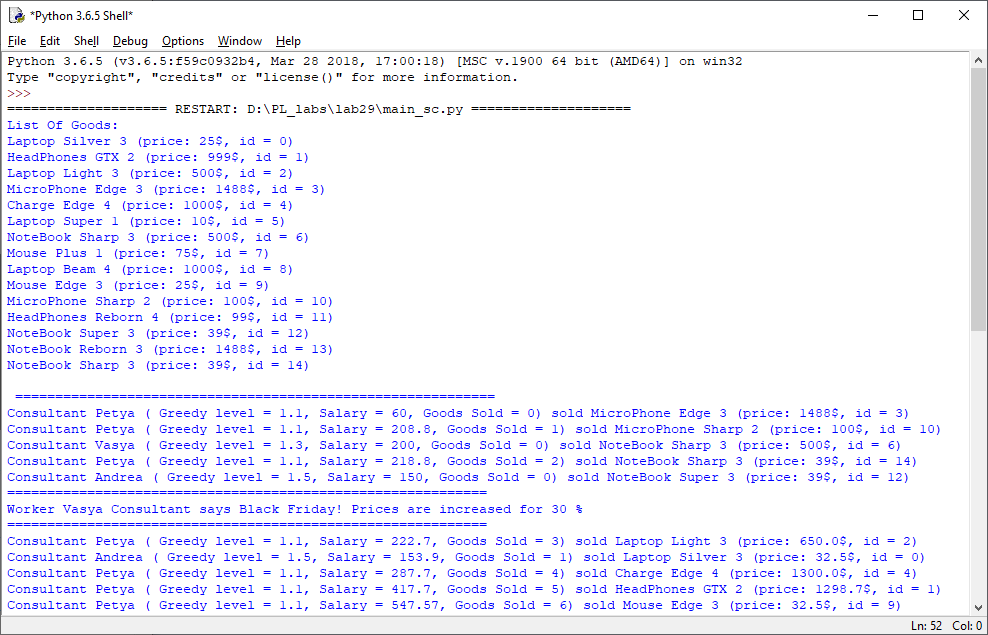


Рис 30.4 Результат работы основного задания

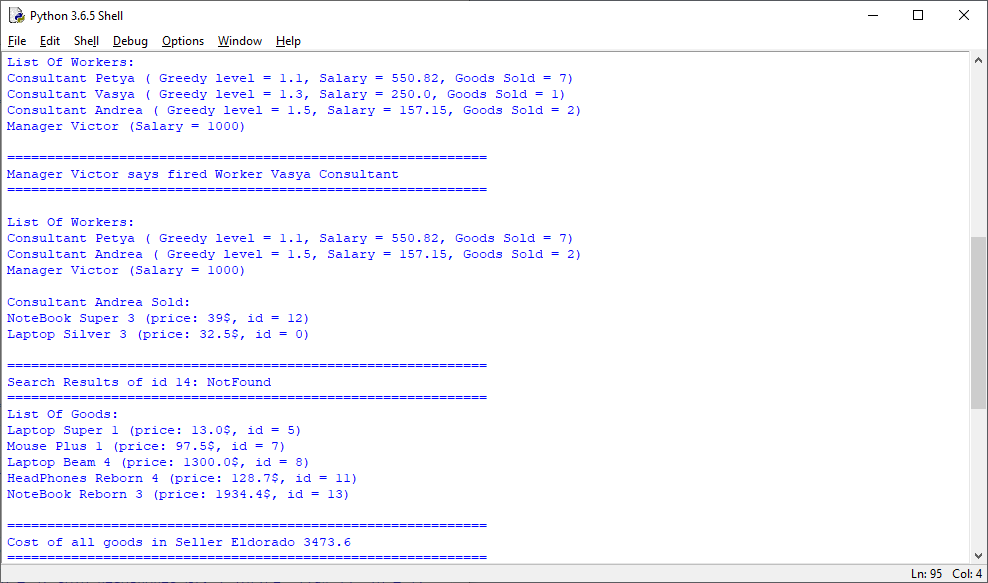


Рис 30.5 Результат работы основного задания

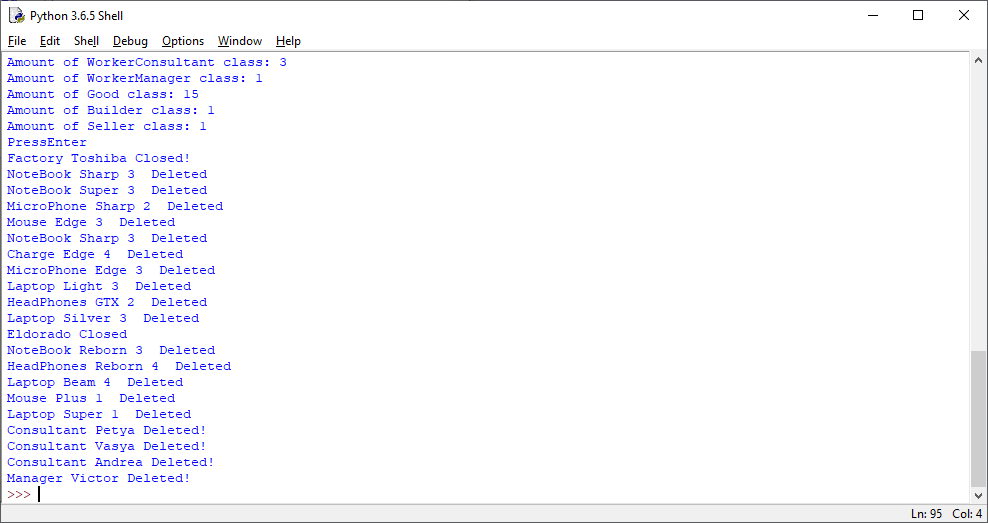


Рис 30.6 Результат работы основного задания (Механизм отслеживания кол-ва созданных объектов и перегрузка метода(деструктора) \_\_del\_\_)

Ответы на контрольные вопросы:

1. Что такое инстанциирование?

*Создание экземпляра (объекта класса)*

2. Кто и как в языке Python создаёт экземпляр (объект) класса?

*Чтобы создать экземпляры классов, нужно вызвать класс с использованием его имени и передать аргументы, которые принимает метод \_\_init\_\_.*

3. Как происходит выделения памяти под экземпляр класса? Приведите полную картину с учётом использования стека (stack) и хипа (heap)?

*Сначала в heap’e выделяется память под экземпляр класса и там же он сохраняется.*

*В stack’e находится ссылка на экземпляр класса, который уже находится в heap.*

4. Как в терминах ООП называется метод, который вызывается сразу после создания объекта для инициализации его состояния?

*Конструктор*

5. Как его описать в классе в языке Python?

*В определении класса при помощи метода \_\_init\_\_*

6. Когда именно вызывается конструктор?

*При создании экземпляра(объекта)*

7. Можно ли перегружать конструкторы? Если да, то зачем это нужно?

*Да, в некоторых языках программирования можно перегружать конструктор(C#, C++, Java), но в Python этого механизма нет. Это нужно для того, чтобы изменять поведение класса в зависимости от переданных ему аргументов*

8. Кто занимается удаление объектов из памяти, если они уже не нужны системе?

*Этим занимается Garbage Collector*

9. Как в терминах ООП называется метод, который вызывается перед удалением объекта из памяти?

*Деструктор*

10. Как происходит удаление памяти объекта? Приведите полную картину с учётом использования стека (stack) и хипа (heap)?

*Изначально удаляется ссылка на объект из стека, когда область видимости , в которой находилась ссылка на объект, прекращает свое существование. После завершения программы Garbage Collector отчищает память в heap. Также удаление объекта может произойти тогда, когда все ссылка на объект прекратят своё существование*.

11. Какие существуют основные столпы методологии ООП?

*1) Абстракция*

*2) Инкапсуляция*

*3) Наследование*

*4) Полиморфизм*

*5) Посылка сообщений*

*6) Повторное использование кода*

12. Что такое абстракция (абстрагирование) в ООП?

*Когда при описании атрибутов объекта мы используем только основные характеристики объекта, которые нужны нам для решения нашей проблемы. А от остальных характеристик объекта абстрагируемся.*

13. Каким образом можно сохранить данные на уровне системы не прибегая к глобальным переменным?

*Можно создать статическую переменную в каком либо классе.*

14. Чем отличаются атрибуты класса от атрибутов экземпляров класса?

*Атрибуты класса будут всегда доступны в независимости от того, как был инициализирован класс. А атрибуты экземпляра всегда могут быть разными*

15. Каким образом можно получить доступ к атрибутам класса? А как можно получить доступ к состоянию и поведению экземпляра класса?

*Если атрибут не является защищённым(private) то его можно получить зная название класса и имя атрибута ClassName.attr\_name. Также можно создать свойства класса, которые предоставляют доступ к защищённым атрибутам класса*

1.Что такое инкапсуляция и зачем она нужна?

*Это сокрытие реализации класса и отделение его внутреннего представления от внешнего. В большинстве случаев при использовании ООП и других методологий не рекомендуется использовать прямой доступ к данным и реализации моделируемых объектов соответствующих классов, а общаться (иметь доступ) только через общедоступный интерфейс.*

2. Каков главный козырь инкапсуляции?

*Можно легко вносить изменения в реализацию программного компонента без риска изменения других компонентов.*

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

*Например, моё имя в паспорте. Другие люди не могут его изменить, сделать могу это только я, для них оно доступно только для чтения. Но если бы менять имя смог бы не только я лично, но и другие люди, то я уверен, что меня бы уже не звали Артёмом)*

4. Является ли инкапсуляция только прерогативой ООП или гдето она уже Вам встречалась при получении опыта программирования?

*При использовании различных программных модулей у меня не было доступа к их исходному коду. Я мог использовать функции, написанные разработчиками этих модулей, и не мог их менять.*

5. Какими средствами обычно обеспечивается инкапсуляция в ООП?

*Например модификаторами доступа в различных языках програмиирования*

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

*Инкапсуляция в языке программирования Python работает лишь на уровне соглашения между разработчиками. О том, какие данные должны быть общедоступными, а какие – внутренними (инкапсулированными), решает сам программист.*

7. Как получить доступ к инкапсулированным данным в языке Python?

*Двойное подчёркивание в коде полностью не защищает доступ к имени идентификатора (поля или метода), т.к. к нему можно всегда обратиться с использованием специального имени через ссылку на соответствующий класс – \_ClassName\_\_ IdentificatorName:*

8. Что такое свойства в языке Python и как их реализовать?

*Свойство - способ доступа к внутреннему состоянию объекта, имитирующий переменную некоторого типа. Их можно реализовать при помощи декораторов класса*

*@property, @property\_name.setter, @property\_name.getter или же при помощи команды property(getter, setter, delleter)*

9. Как реализовать свойства «только для чтения» или свойство «только для записи»?

*@property, @property\_name.setter,*

*def get\_attr(self):*

*return attr*

*def set\_attr(self, value):*

*attr = value*

*Attr = property(get\_attr, set\_attr)*

10. Зачем нужны динамические методы и как их реализовать в языке Python?

*Динамические методы доступны только в экземпляре класса.*

*Используется в тех случаях, если его реализация на прямую или косвенно связано с состоянием объекта. Согласно ООП парадигме правильный вызов динамических методов в Попытка обратиться к методу экземпляра класса через специальное имя класса коде должен происходить на ссылочной переменной, т.е. на самом экземпляре. Это наиболее часто используемый вид методов.*

11. Зачем нужны методы класса и как их реализовать в языке Python?

Они могут менять состояние класса, что отразится на всех объектах этого класса, но не могут менять конкретный объект. Методы класса принимают класс в качестве параметра, который принято обозначать как cls.Реализовать их можно при помощи декоратора @classmethod

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

*Статические методы в коде помечаются декораторами @staticmethod и описывают общий функционал, который связан с каким-либо программируемым процессом, а не поведением какого-нибудь отдельного объекта или объекта-класса.*

13. Каким образом можно вызвать в языке Python динамические, статические методы и методы класса?

*Статические методы работают как обычные функции, но принадлежат области имён класса. Статический метод не принимает неявных аргументов, к примеру, self или cls, а следовательно, никак не может поменять состояние объекта или атрибуты класса, в котором он описан. Они работают только с теми данными, я передаются явно в качестве аргументов.*

*Правильный вызов динамических методов в Попытка обратиться к методу экземпляра класса через специальное имя класса коде должен происходить на ссылочной переменной, т.е. на самом экземпляре.*

*Они могут менять состояние класса, что отразится на всех объектах этого класса, но не могут менять конкретный объект. Методы класса принимают класс в качестве параметра, который принято обозначать как cls.*

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

*Они принимают различные аргументы и по разному работают с объектом класса.*

15. Преимущества и недостатки инкапсуляции?

*Плюсы инкапсуляции:*

*- обеспечивает согласованность, целостность и непротиворечивость данных внутреннего состояния компонента (объекта или класса в целом);*

*- гораздо проще контролировать корректные значения полей путём полного контроля над входящими и исходящими данными;*

*- не составит труда изменить способ хранения состояния (данных) компонента (если информация станет храниться не в памяти, а в долговременном хранилище, такой как файловая система или база данных, то потребуется изменить внутреннюю реализацию ряда методов одного класса, а не вносить изменения во все части системы, где напрямую использовались данные компонента);*

*- облегчает поиск ошибок, а также программный код легче отлаживать*

*Недостатки инкапсуляции:*

*-необходимо больше писать кода;*

*-сложность исправлять ошибки сторонних библиотек;*

*-снижается общая производительность и скорость работы приложения.*

1. Что такое повторное использование кода?

*Методология проектирования компьютерных и других систем, заключающаяся в том, что система (компьютерная программа, программный модуль) частично либо полностью должна составляться из частей, написанных ранее компонентов и/или частей другой системы. Повторное использование — основная методология, которая применяется для сокращения трудозатрат при разработке сложных систем.*

2. Какие подходы и средства предлагает методология ООП для повторного использования кода?

*ООП предполагает механизм наследования*

3. На что указывает принцип DRY (Don't Repeat Yourself)?

*На то что самое простое решение по уменьшению сложности – разделить систему на мелкие, независимые модули, которыми проще управлять.*

4. Что такое зависимость? Как зависимость реализуется в языке Python?

*Между объектами можно устанавливать зависимости по данным. Эти зависимости выражают связи или отношения между классами указанных объектов. Например при импорта модуля мы также импортируем другие модули, которые были использованы для написания этого модуля*

5. Что такое ассоциация? Как реализована ассоциация в языке Python?

*В объектно-ориентированном программировании ассоциация означает отношение между классами объектов, которое позволяет одному экземпляру объекта вызвать другой, чтобы выполнить действие от его имени.*

6. Что такое наследование? Как реализовано наследование в языке Python?

*Это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым или родительским. Новый класс – потомком, наследником или производным классом.*

*Наследуемый класс в Python указывается внутри круглых скобок в определении производного класса*

7. Какие преимущества и недостатки есть у множественного наследования классов?

*Можно использовать множество различных классов для создания нового объекта.*

*Сложно отслеживать ошибки, если они могут возникнуть.*

8. Разрешено ли в языке Python множественное наследование? Если разрешено множественное наследование в языке Python, то как наследуются дочерним классом одинаковые характеристики из разных базовых классов?

*Да, можно. Если имя характеристик совпадает, то будет использована характеристика, которая была объявлена самой последней*

9. Как при наследовании вызываются конструкторы?

*Если в производном классе конструктор переопределён, то он будет вызван именно переопределённый метод*

10. Кто находится во главе всей иерархии в языке Python?

*Класс object*

11. Какой именно минимальный функционал получают все экземпляры классов в языке Python? С помощью какой встроенной функции можно посмотреть этот функционал?

*'\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_'*

*При помощи функции dir()*

12. Что показывает функция isinstance() и как ей пользоваться?

*Она показывает принадлежность к какому либо классу.*

*isinstance(object, ClassInfo)*

13. Что показывает функция issubclass() и как ей пользоваться?

*Возвращает флаг, указывающий на то, является ли указанный класс подклассом указанного класса (классов).*

*issubclass(cls, classinfo) -> bool*

*cls : Класс, требующий проверки.*

*classinfo : Класс, либо кортеж с классами. Если аргумент не является классом, либо кортежем с классами, возбуждается TypeError*

14. Что такое агрегация? Как реализована агрегация в языке Python?

*Когда объект хранит в себе другой объект. Например при создании объекта ему можно передать ссылку на другой класс.*

15. Что такое композиция? Как реализована композиция в языке Python?

*Композиция – более строгая реализация агрегации. Добавление объекта в контейнер*

16. Что такое делегирование? Как реализовано делегирование в языке Python?

*В реализации части своего функционала, объект полагается на другой объект, в Python это можно организовать путём создания атрибута хранящего класс помощник*.

17. Основное отличие наследования от агрегации/композиции?

*Наследование реализует полиморфизм.*

18. Преимущества и недостатки композиции и агрегации? Преимущества и недостатки наследования?

*Недостатки и плюсы композиции и агрегации:*

*-Большое кол-во зависимого кода*

*- Невозможно реализовать полиморфизм*

*+Нет конфликтов при наследовании*

*+Возможность смены объекта во время выполнения*

*+Возможность замены агр. объектов во время выполенния*

*Недостатки и плюсы наследования:*

*-Зависимость от базовых объектов*

*-Конфликты имён*

*+Реализация полиморфного поведения*

19. Как с помощью языка UML отобразить все зависимости между классами на UML-диаграмме классов?

*Зависимости отображаются с помощью линий и стрелок. В разные типы зависимостей обозначаются разными стрелками*.

20. Какой должен базовый класс у классов-исключений? Как создавать пользовательские исключения?

*У классов исключений должен быть базовый класс BaseException. Для создания пользовательских исключений нужно наследовать класс Exception*

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("=" \* 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\_info(self):

return self.\_\_name

def get\_workers\_info(self):

result = "List Of Workers:\n"

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

if isinstance(worker, Worker):

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

return result

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:

\_\_salary = 0

\_\_status = ""

\_\_name = ""

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

\_\_status = "Manager"

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

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

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

self.\_\_name = name

self.\_\_salary = salary

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

\_\_status = "Consultant"

\_\_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!")

self.\_\_name = name

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

self.\_\_salary = salary

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)

*ПРИЛОЖЕНИЕ Б*

Листинг исходных кодов программ лабораторной работы №27

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

TYPES = ("Computer", "NoteBook", "Laptop",

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

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

UNDERTYPES = ("Reborn", "Edge", "Plus",

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

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

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

"Premium")

PRICE\_LIST = (100, 500, 50,

10000, 1000, 75,

25, 999, 99, 10, 39)

MODEL\_START = 1

MODEL\_END = 5

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

Builder.amount += 1

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

print("Factory Closed!")

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, i))

return goods\_list

def get\_string\_repr(self, 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

def get\_types(self):

info = ""

for typ in self.TYPES:

info += typ + "\n"

return info

def get\_undertypes(self):

info = ""

for undertype in self.UNDERTYPES:

info += undertype + "\n"

return info

Файл Client.py

class Client:

number\_of\_clients = 0

def \_\_init\_\_(self, name, money, id):

self.name = name

self.money = money

self.id = id

Client.number\_of\_clients += 1

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

print(self.name, " - deleted")

Client.number\_of\_clients -= 1

def \_\_repr\_\_(self):

return self.name

def main():

lst = []

for i in range(20):

lst.append(Client("A"+str(i), 100\*i, i))

print(lst)

main()

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, identity):

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 get\_info(self):

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

self.price,

self.identity)

def get\_recomended\_price(self):

return self.recomended\_price

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 Worker

from Goods import Good

def main():

build = Builder()

go = build.create\_goods(10)

eldorado = Seller(go, "Eldorado")

eldorado.add\_worker(Worker("Vasya", salary=100, greedy\_level=1.3))

eldorado.add\_worker(Worker("Petya", salary=200, greedy\_level=1.8))

eldorado.add\_worker(Worker("Ivan", salary=150, greedy\_level=1.1))

eldorado.add\_worker(Worker("Vasya", salary=560, greedy\_level=1.4))

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

print(build.get\_string\_repr(eldorado.goods\_list))

print("Amount of Worker class:", Worker.amount\_of\_workers)

print("Amount of Good class:", Good.amount)

print("Amount of Builder class:", Builder.amount)

print("Amount of Seller class:", Builder.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 Worker

import random

class Seller:

worker\_list = []

amount = 0

def \_\_init\_\_(self, goods\_list, name):

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

goods\_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 calculate\_prices(self):

all\_price = 0

for good in self.goods\_list:

if isinstance(good, Good):

all\_price += good.price

return all\_price

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.get\_info()

break

return result

def add\_worker(self, worker):

if isinstance(worker, Worker):

self.worker\_list.append(worker)

def get\_random\_worker(self):

return random.choice(self.worker\_list)

def get\_info(self):

return self.name

def get\_workers\_info(self):

result = "List Of Workers:\n"

for worker in self.worker\_list:

if isinstance(worker, Worker):

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

return result

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:

add\_to\_salary = 0.1

amount\_of\_workers = 0

amount\_of\_sold\_goods = 0

list\_of\_sold\_goods = []

def \_\_init\_\_(self, name, salary, greedy\_level=1, status="Consultant"):

self.name = name

self.status = status

self.salary = salary

self.greedy\_level = greedy\_level

Worker.amount\_of\_workers += 1

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

print("{0} {1} Fired!".format(self.status, self.name ))

Worker.amount\_of\_workers -= 1

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

print(self.status, self.name,

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

if 0 < sale\_coef < 1:

print(self.status, self.name, ": 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:

print("What?")

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())

self.salary += self.add\_to\_salary \* good.price

self.amount\_of\_sold\_goods += 1

self.list\_of\_sold\_goods.append(good.get\_info())

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

result = "{0} {1} Sold Nothing".format(self.status, self.name)

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

result = "{0} {1} Sold:".format(self.status, self.name)

for info in self.list\_of\_sold\_goods:

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.amount\_of\_sold\_goods)

*ПРИЛОЖЕНИЕ В*

Листинг исходных кодов программ лабораторной работы №28

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

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

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

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

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

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

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

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

"Premium")

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

10000, 1000, 75,

25, 999, 99, 10, 39)

\_\_MODEL\_START = 1

\_\_MODEL\_END = 5

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

self.\_\_name = name

Builder.\_\_amount += 1

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

Builder.\_\_amount -= 1

print("Factory Closed!")

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

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

return "Factory"

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, i))

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):

self.\_\_name = value

def get\_undertypes(self):

info = ""

for undertype in self.\_\_UNDERTYPES:

info += undertype + "\n"

return info

@staticmethod

def get\_amount():

return Builder.\_\_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):

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

@staticmethod

def get\_amount():

return Good.\_\_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 Worker

from Goods import Good

def main():

build = Builder("Toshiba")

go = build.create\_goods(10)

eldorado = Seller("ELDORADO", go)

eldorado.add\_worker(Worker("Egor", salary=100, greedy\_level=1.3))

eldorado.add\_worker(Worker("Petya", salary=200, greedy\_level=1.8))

eldorado.add\_worker(Worker("Ivan", salary=150, greedy\_level=1.1))

eldorado.add\_worker(Worker("Vasya", salary=560, greedy\_level=1.4))

print(eldorado.find\_good\_by\_id(2))

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

print("BestWorker", eldorado.find\_best\_worker())

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

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

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

print("Amount of Worker class:", Worker.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 Worker

import random

class Seller:

\_\_amount = 0

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

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

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

goods\_list.clear()

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

@staticmethod

def get\_amount():

return Seller.\_\_amount

def calculate\_prices(self):

all\_price = 0

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

if isinstance(good, Good):

all\_price += good.Price

return all\_price

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.get\_info()

break

return result

def find\_best\_worker(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], Worker):

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

if 0 <= maximum < new\_max:

maximum = new\_max

max\_index = i

return self.\_\_worker\_list[max\_index].get\_info()

def add\_worker(self, worker):

if isinstance(worker, Worker):

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

def get\_random\_worker(self):

return random.choice(self.\_\_worker\_list)

def get\_info(self):

return self.\_\_name

def get\_workers\_info(self):

result = "List Of Workers:\n"

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

if isinstance(worker, Worker):

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

return result

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

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:

\_\_salary\_add\_coef = 0.1

\_\_amount = 0

\_\_sold\_good\_amount = 0

\_\_sold\_goods\_info = []

def \_\_init\_\_(self, name="", salary=0, greedy\_level=1, status="Consultant"):

self.\_\_name = name

self.\_\_status = status

self.\_\_salary = salary

self.greedy\_level = greedy\_level

Worker.\_\_amount += 1

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

print("{0} {1} Fired!".format(self.\_\_status, self.\_\_name))

Worker.\_\_amount -= 1

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

string = "Worker {0} {1}".format(self.\_\_name, self.\_\_status)

return string

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

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

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

if 0 < sale\_coef < 1:

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

": 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:

print("What?")

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())

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

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

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

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:".format(self.\_\_status, self.\_\_name)

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

if isinstance(info, str):

result += info

return result

def get\_name(self):

return self.\_\_name

def get\_status(self):

return self.\_\_status

def set\_status(self, value):

self.\_\_status = value

def get\_salary(self):

return self.\_\_salary

def set\_salary(self, value):

self.\_\_salary = value

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

return self.\_\_sold\_good\_amount

def get\_salary\_coef(self):

return self.\_\_salary\_add\_coef

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

if 0 < value <= 1:

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

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)

@staticmethod

def get\_amount():

return Worker.\_\_amount

Name = property(get\_name)

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

Status = property(get\_status, set\_status)

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

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