



Programming
Languages

Лабораторная работа #29

**Объектно-Ориентированное Программирование.
Повторное использование кода и
наследование в языке Python**



LEARN. GROW. SUCCEED.

© 2019-2020. Department: <Software of Information Systems and Technologies>
Faculty of Information Technology and Robotics
Belarusian National Technical University
by Viktor Ivanchenko / ivanvikvik@bntu.by / Minsk

ЛАБОРАТОРНАЯ РАБОТА #29

ООП. Повторное использование кода и наследование в языке Python

Цель работы

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.

Общее задание

Необходимо 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); 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.



Требования к выполнению задания

- 1) Необходимо модернизировать спроектированную в предыдущей лабораторной работе UML-диаграмму взаимодействия классов и объектов программной системы исходя из текущих дополнений. На базе данной изменённой 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 и функций.
- 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 снабжена 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***).

Best of LUCK with it, and remember to HAVE FUN while you're learning :)
Victor Ivanchenko



Графическое представление элементов UML-диаграммы классов

UML – унифицированный язык моделирования (***Unified Modeling Language***) – это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для **визуализации, спецификации, конструирования** и **документирования** программных систем. Язык UML применяется не только для проектирования, но и с целью документирования, а также эскизирования проекта.

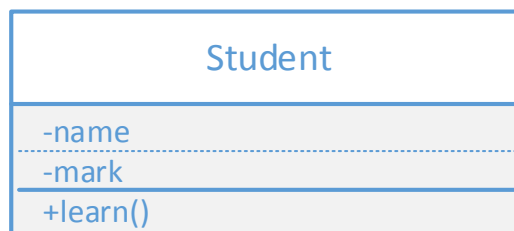
Словарь UML включает три вида строительных блоков: **диаграммы, сущности** и **связи**. **Сущности** – это абстракции, которые являются основными элементами модели, **связи** соединяют их между собой, а **диаграммы** группируют представляющие интерес наборы сущностей.

Диаграмма – это графическое представление набора элементов, чаще всего изображенного в виде связанного графа вершин (сущностей) и путей (связей). Язык UML включает **13** видов диаграмм, среди которых на первом (центральном) месте в списке – диаграмма классов.

UML-диаграмма классов (*Static Structure Diagram*) – диаграмма статического представления системы, демонстрирующая классы (и другие сущности) системы, их атрибуты, методы и взаимосвязи между ними.

Диаграммы классов оперируют тремя видами сущностей UML: структурные, поведенческие и аннотирующие.

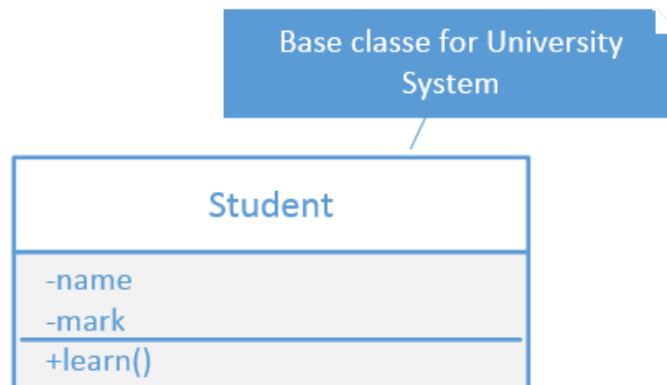
Структурные сущности – это «имена существительные» в модели UML. В основном, статические части модели, представляющие либо концептуальные, либо физические элементы. Основным видом структурной сущности в диаграммах классов является класс. Пример класса Студент (*Student*) с полями и методами:



Поведенческие сущности – динамические части моделей UML. Это «глаголы» моделей, представляющие поведение модели во времени и пространстве. Основной из них является взаимодействие – поведение, которое заключается в обмене сообщениями между наборами объектов или ролей в определенном контексте для достижения некоторой цели. Сообщение изображается в виде линии со стрелкой:



Аннотирующие сущности – это поясняющие части UML-моделей, иными словами, комментарии, которые можно применить для описания, выделения и пояснения любого элемента модели. Главная из аннотирующих сущностей – примечание. Это символ, служащий для описания ограничений и комментариев, относящихся к элементу либо набору элементов. Графически представлен прямоугольником с загнутым углом; внутри помещается текстовый или графический комментарий.



Графически класс изображается в виде прямоугольника, разделенного на 3 блока горизонтальными линиями: имя класса, атрибуты (свойства) класса и операции (методы) класса.

Для атрибутов и операций может быть указан один из нескольких типов видимости:

- + открытый, публичный (*public*)
- закрытый, приватный (*private*)
- # защищённый (*protected*)
- / производный (*derived*) (может быть совмещён с другими)
- ~ пакет (*package*)

Видимость для полей и методов указывается в виде левого символа в строке с именем соответствующего элемента.



Каждый класс должен обладать именем, отличающим его от других классов. **Имя** – это текстовая строка. Имя класса может состоять из любого числа букв, цифр и знаков препинания (за исключением двоеточия и точки) и может записываться в несколько строк. Каждое слово в имени класса традиционно пишут с заглавной буквы (верблюжья нотация), например Датчик (*Sensor*) или ДатчикТемпературы (*TemperatureSensor*).

Атрибут (свойство) – это именованное свойство класса, описывающее диапазон значений, которые может принимать экземпляр атрибута. Класс может иметь любое число атрибутов или не иметь ни одного. В последнем случае блок атрибутов оставляют пустым.

Атрибут представляет некоторое свойство моделируемой сущности, которым обладают все объекты данного класса. Имя атрибута, как и имя класса, может представлять собой текст. Можно уточнить спецификацию атрибута, указав его тип, кратность (если атрибут представляет собой массив некоторых значений) и начальное значение по умолчанию.

Статические атрибуты или просто атрибуты класса обозначаются **подчеркиванием**.

Операция (метод) – это реализация метода класса. Класс может иметь любое число операций либо не иметь ни одной. Часто вызов операции объекта изменяет его атрибуты.

Графически операции представлены в нижнем блоке описания класса.

Допускается указание только имен операций. Имя операции, как и имя класса, должно представлять собой текст. Каждое слово в имени операции пишется с заглавной буквы, за исключением первого, например move (переместить) или isEmpty (проверка на пустоту).

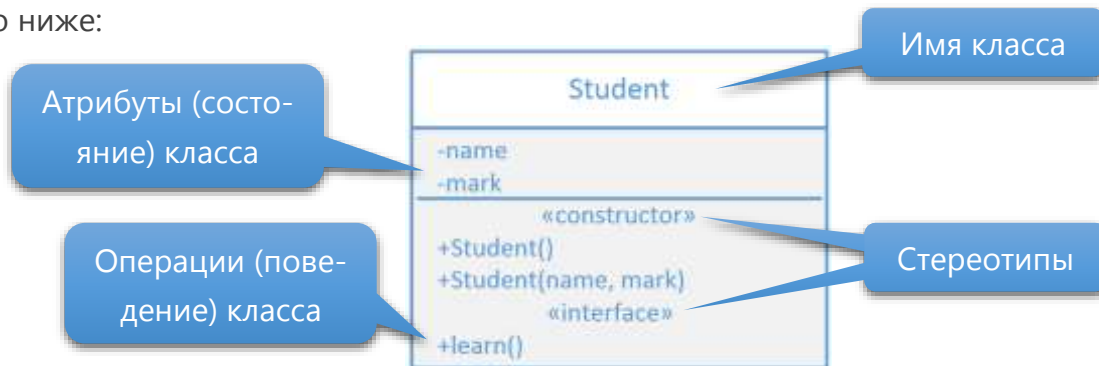
Можно специфицировать операцию, устанавливая ее сигнатуру, включающую имя, тип и значение по умолчанию всех параметров, а применительно к функциям – тип возвращаемого значения.

Статические методы класса обозначаются **подчеркиванием**.

Чтобы легче воспринимать длинные списки атрибутов и операций, желательно снабдить префиксом (именем стереотипа) каждую категорию в них. В данном случае ***стереотип*** – это слово, заключенное в угловые кавычки, которое указывает то, что за ним следует.



Общее описание класса с именем Student и другими атрибутами представлено ниже:



Существуют следующие типы связей в UML: **зависимость**, **ассоциация** (и её разновидности: **агрегация** и **композиция**), **наследование** (обобщение) и **реализация**. Эти связи представляют собой базовые строительные блоки для описания отношений в UML, используемые для разработки хорошо согласованных моделей.

Первая из них – **зависимость** – семантически представляет собой связь между двумя элементами модели, в которой *изменение одного элемента (независимого) может привести к изменению семантики другого элемента (зависимого)*. Графически представлена пунктирной линией, иногда со стрелкой, направленной к той сущности, от которой зависит еще одна; может быть снабжена меткой.



Зависимость – это связь *использования*, указывающая, что изменение спецификаций одной сущности может повлиять на другие сущности, использующие её.

Ассоциация – это структурная связь между элементами модели, которая описывает набор связей, существующих между объектами. Ассоциация показывает, что объекты одной сущности (класса) связаны с объектами другой сущности таким образом, что можно перемещаться от объектов одного класса к другому. Например, класс Человек (*Human*) и класс Школа (*School*) имеют ассоциацию, так как человек может учиться в школе. Ассоциации можно присвоить имя «учится в». В представлении однонаправленной ассоциации добавляется стрелка, указывающая на направление ассоциации.

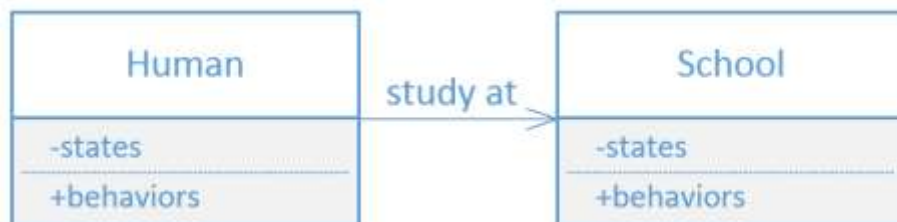
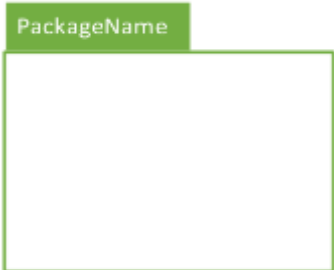
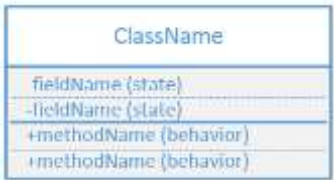
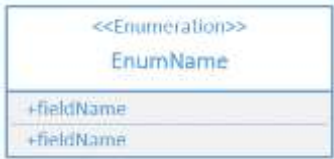
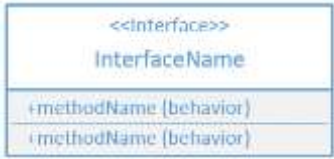








Таблица 1 – Наиболее часто используемые элементы UML-диаграммы

#	Shape (блок)	Description (описание)
1.		Элемент для описания пакета. Пакет логически выделяет группу классов, которые описаны в нём.
2.		Элемент для описания класса. Класс представлен в рамках, содержащих три компонента: имя класса, поля (атрибуты) класса и методы класса.
3.		Элемент для описания перечисления. Перечисление представляется аналогично классу с ключевым словом в самом вверху « <i>Enumeration</i> ».
4.		Элемент для описания интерфейса. Интерфейс представлен в рамках, содержащих два компонента: имя интерфейса с ключевым слово « <i>Interface</i> » и методы интерфейса.
5.		Аннотация (комментарий). Аннотация используется для размещения поясняющего (уточняющего) текста на диаграмме для соответствующих сущностей.
		Зависимость (<i>Dependency</i>)
6.		Ассоциация (<i>Association</i>)
7.		Агрегация (<i>Aggregation</i>)
8.		Композиция (<i>Composition</i>)
9.		Наследование (<i>Inheritance</i>)
10.		Реализация (<i>Implementation or Realization</i>)



Контрольные вопросы

1. Что такое повторное использование кода?
2. Какие подходы и средства предлагает методология ООП для повторного использования кода?
3. На что указывает принцип **DRY (Don't Repeat Yourself)**?
4. Что такое зависимость? Как зависимость реализуется в языке Python?
5. Что такое ассоциация? Как реализована ассоциация в языке Python?
6. Что такое наследование? Как реализовано наследование в языке Python?
7. Какие преимущества и недостатки есть у множественного наследования классов?
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?
9. Как при наследовании вызываются конструкторы?
10. Кто находится во главе всей иерархии в языке Python?
11. Какой именно минимальный функционал получают все экземпляры классов в языке Python? С помощью какой встроенной функции можно посмотреть этот функционал?
12. Что показывает функция **isinstance()** и как ей пользоваться?
13. Что показывает функция **issubclass()** и как ей пользоваться?
14. Что такое агрегация? Как реализована агрегация в языке Python?
15. Что такое композиция? Как реализована композиция в языке Python?
16. Что такое делегирование? Как реализовано делегирование в языке Python?
17. Основное отличие наследования от агрегации/композиции?
18. Преимущества и недостатки композиции и агрегации? Преимущества и недостатки наследования?
19. Как с помощью языка UML отобразить все зависимости между классами на UML-диаграмме классов?
20. Какой должен базовый класс у классов-исключений? Как создавать пользовательские исключения?

