



Laboratory Work #21

Java Object-Oriented Programming. Abstraction



LEARN. GROW. SUCCEED.

© 2020-2021. 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 #21

Объектно-Ориентированное Программирование на языке Java.

Абстракция

Цель работы

Научиться грамотно анализировать предметную область и с помощью абстракции выделять существенные детали, на базе которых в дальнейшем проектируются классы и объекты будущей программной системы согласно методологии ООП, а также практически закрепить данные навыки при решении соответствующих задач (бизнес проблем).

Требования

- 1) Необходимо спроектировать и реализовать UML-диаграмму взаимодействия классов и объектов разрабатываемой программной системы с отображением всех связей (отношений) между классами и объектами.
- 2) При проектировании и разработке системы необходимо полностью использовать своё объектно-ориентированное воображение и по максимум использовать возможности, которые предоставляет язык программирования Java для реализации ООП-методологии.
- 3) Основные классы системы должны быть самодостаточными, т.е. не зависеть, к примеру, от консоли! Любые типы отношений между классами должны применяться обосновано и лишь тогда, когда это имеет смысл.
- 4) При выполнении задания необходимо по максимуму пытаться разрабатывать универсальный, масштабируемый, легко поддерживаемый и читаемый код.
- 5) Также рекомендуется придерживаться **Single Responsibility Principle, SRP** (принципа единственной ответственности): у каждого пакета, класса или метода должна быть только одна ответственность (цель), т.е. должна быть только одна причина изменить в дальнейшем соответствующий блок кода.

- 6) Создаваемые классы необходимо грамотно разложить по соответствующим пакетам, которые должны иметь «адекватные» названия и быть вложены в указанные стартовые пакеты: ***by.bntu.fitr.poisit.nameofstudent.nameofproject***.
- 7) В соответствующих компонентах бизнес-логики необходимо предусмотреть «защиту от дурака».
- 8) На базе спроектированной программной системы реализуйте простейшее интерактивное консольное приложение. Используйте при реализации архитектурный шаблон проектирования ***Model-View-Controller, MVC***.
- 9) Программа должна обязательно быть снабжена комментариями, в которых необходимо указать краткое предназначение программы, номер лабораторной работы и её название, версию программы, ФИО разработчиков, название бригады (если есть), номер группы и дату разработки. Исходный текст классов и демонстрационной программы рекомендуется также снабжать поясняющими краткими комментариями.
- 10) Программа должна быть снабжена дружелюбным и интуитивно понятным интерфейсом для взаимодействия с пользователем.
- 11) Интерфейс программы и комментарии должны быть на английском языке.
- 12) Необходимо проверить все тестовые случаи работы основной бизнес-логики программы.
- 13) При разработке программ придерживайтесь соглашений по написанию кода на *Java* (***Java Code-Convention***) !!!

Основное задание



Необходимо решить задачу с использованием методологии ООП. Для чего необходимо подобрать самостоятельно соответствующую проблемную (предметную/доменную) область, которая базируется на объектах и событиях реального мира (примеры соответствующих предметных областей приведены ниже). Спроектировать классы (собственные пользовательские типы данных) в языке Java для программного представления данных объектов и основной логики будущей программной системы.

Система должна решать, как минимум, два полезных действия и иметь дополнительно следующие опции:

- не менее 3 разнообразных классов предметной области;
- не менее 5 атрибутов (состояния) и методов (поведения) в классе-сущности;
- не менее 3 методов, которые реализуют бизнес-логику программы, в соответствующих функциональных классах;
- хранить глобальные характеристики системы или характеристики уровня отдельных классов.

На базе спроектированной программной системы реализовать программу и продемонстрировать её работоспособность.

Примеры предметных областей

1. **Цветочница или магазин цветов (*Flower Shop*)**. В магазине цветов можно собрать букет из соответствующих цветов. Необходимо определить вес букета, его стоимость и самый дорогой/недорогой цветок (или цветы, если они одинаковы по стоимости).
2. **Новогодняя ёлка (*Christmas tree*)**. Есть новогодняя ёлка, которую можно украсить соответствующими новогодними игрушками. Необходимо подсчитать вес всей ёлки вместе с игрушками, её общую стоимость, а также найти самую дорогую игрушку.
3. **Кредиты (*Credits*)**. В банке можно сформировать набор предложений клиенту по соответствующим целевым кредитам. Необходимо подсчитать общую сумму по всем кредитам клиента и найти максимальную ставку по кредиту и самый дорогой кредит клиента.
4. **Жилищно-Коммунальное Хозяйство, ЖКХ (*Housing and Communal Services*)**. В ЖКХ предлагают набор соответствующих услуг по обслуживанию и эксплуатации жилищного хозяйства клиента. Необходимо подсчитать общую стоимость услуг, которые были оказаны клиенту ЖКХ и найти самую дорогую (недорогую) услугу.
5. **Турагентство и Туристические путевки (*Tourist trips*)**. В турфирме можно сформировать набор предложений клиенту по выбору туристической путевки различного типа. Необходимо подсчитать общую сумму, которую клиент должен заплатить за выбранные им путёвки и найти самую выгодную по стоимости путёвки, исходя из расчёта стоимости за один день.
6. **Шеф-повар (*Chef*)**. Необходимо приготовить овощной салат и определить вес салата и его калорийность, а также найти самый калорийных овощ в салате.
7. **Шеф-повар (*Chef*)**. Необходимо приготовить фруктовый салат и определить вес салата и его калорийность, а также найти самый калорийных фрукт в салате.
8. **Налоги (*Taxes*)**. Определить множество и сумму налоговых выплат физического лица за год с учетом доходов с основного и дополнительного мест работы, авторских вознаграждений, продажи имущества, получения в подарок

денежных сумм и имущества, переводов из-за границы, льгот на детей и материальную помощь.

9. **Шеф-повар (Chef)**. Необходимо приготовить основное блюдо (или десерт) определить его вес и калорийность, а также найти самый калорийных ингредиент в блюде (десерте).
10. **Игровая комната (Game room)**. Есть игровая комната с соответствующими игрушками. Необходимо провести подсчёт стоимости всех игрушек, их общий вес и найти самую дорогую игрушку.
11. **Таксопарк (Taxi Station)**. Есть таксопарк, который состоит из соответствующего автотранспорта. Необходимо подсчитать стоимость автопарка и найти самый дорогой автотранспорт по тарифам поездки.
12. **Авиакомпания (Airline)**. Есть авиакомпания, которая состоит из соответствующих самолётов. Необходимо подсчитать общее количество мест для пассажиров и грузоподъемность всех самолётов авиакомпании, а также найти самый вместительный самолёт авиакомпании.
13. **Камни (Stones)**. В ювелирном магазине можно отобрать соответствующие драгоценные камни для ожерелья. Необходимо подсчитать общий вес (в каратах) и стоимость изделия. Определить самый дорогой камень в ожерелье.
14. **Звукозапись (Sound Recording)**. В звукозаписывающей студии можно записать на диск сборку музыки. Необходимо подсчитать продолжительность всего диска и найти самую короткую (длинную) песню.
15. **Компьютерный герой (Game Hero)**. Определить сильные стороны героя (его способности: ум, сила, ловкость и т.д.) на базе собранных им артефактов. Подсчитать общую стоимость артефактов, а также величину его соответствующих способностей.
16. **Новогодний подарок (New Year gift)**. Есть новогодний сладкий подарок, который состоит из конфет и прочих сладостей. Необходимо подсчитать общий вес и стоимость подарка, а также самую дорогую (недорогую) сладость.
17. **Железнодорожный транспорт (Railway Transport)**. Имеются поезда, которые состоят из соответствующих вагонов. Необходимо подсчитать общую длину конкретного поезда, а также найти самый длинный (короткий) вагон.

18. **Автосалон (Car Center).** Есть автосалон, который состоит из соответствующих машин. Необходимо подсчитать общую стоимость машин автосалона и найти самую дорогую (недорогую) машину.
19. **Страховое агентство (Insurance Company).** Есть страховая фирма, которая предлагает страховые услуги и обязательства своим клиентам. Необходимо подсчитать общую стоимость страховых услуг, оказываемых конкретному клиенту, а также найти самую дорогую (недорогую) услугу.
20. **Грузоперевозки (Cargo Transportation).** Есть транспортная компания, которая занимается грузовыми перевозками. Необходимо подсчитать максимальное количество грузов, которое за один раз может осуществить компания, исходя из имеющего собственного грузового транспорта, а также найти транспорт, который перевозит максимальное (минимальное) количество груза.
21. **Пассажирские перевозки (Passenger Operations).** Есть транспортная компания, которая занимается пассажирскими перевозками. Необходимо подсчитать максимальное количество пассажиров, которых за один раз может перевезти компания, исходя из имеющего собственного пассажирского транспорта, а также найти транспорт, который перевозит максимальное (минимальное) количество пассажиров.
22. **Фургон кофе (Coffee Car).** Есть фургон определенного объема, в который можно загрузить на определенную сумму соответствующие упаковки кофе. Необходимо определить, сколько упаковок может влезть в фургон заданного объема и на какую общую сумму.
23. **Рыцарь (Knight).** Есть рыцаря, который экипирован соответствующей амуницией. Необходимо определить общую стоимость всей амуниции, а также найти самую дорогую (недорогую) амуницию рыцаря.
24. **Мобильная связь (Mobile Communication).** Компания-оператор мобильной связи предлагает своим клиентам различные тарифные планы использования мобильной связи. Необходимо определить общую сумму дохода компании в месяц, которую она получает от клиентов за использования соответствующих тарифных планов, а также найти клиента, который заплатил больше (меньше) других.
25. **Футбольный Менеджер (Football Manager System).** Есть футбольные клубы, у которых есть соответствующие характеристики, влияющие на исход встречи

с противником. Необходимо определить шансы футбольной команды на победу против команды-соперника, а также общие шансы на победу в кубке и чемпионате соответствующей страны.

26. **Вклады или кредиты (*Deposits or Credits*)**. Банки предлагают своим клиентам соответствующие вклады или кредиты с различными процентами. Необходимо определить общую сумму дохода соответствующего банка в месяц, которую он получает согласно соответствующей марже. Найти самые дорогие (недорогие) кредиты с учётом всех платежей или самые выгодные (невыгодные) вклады с учётом процентной ставки и доходности по ним.
27. **Зоопарк (*Zoo*)**. Есть зоопарк животных, которые **ОЧЕНЬ** любят кушать. Необходимо определить общий суточный запас продуктов, необходимых зоопарку, чтобы прокормить всех своих животных. Также необходимо найти самых (мнее) прожорливых животных зоопарка.
28. **Спортивная рыбалка (*Sports Fishing*)**. В соревнованиях по спортивной рыбалке собираются команды из нескольких рыбаков. В процессе соревнований каждая из команд за определённое время должна наловить максимальное количество рыбы. Необходимо определить общий улов всех команд, а также найти команду победителя (команда, которая имеет наибольший улов рыбы) и аутсайдера (команда, которая имеет самые скромные результаты по улову).
29. **Компания по разработки программного обеспечения, ИТ-компания (*Software Company*)**. Есть ИТ-компании, которые состоят из соответствующих сотрудников. Каждый сотрудник за свою работу получает соответствующую зарплату. Необходимо найти общий фонд заработной платы компании на месяц, а также определить сотрудников с максимальной (минимальной) заработной платой.
30. **Домашние электроприборы (*House Equipments*)**. Подсчитать потребляемую мощность всех бытовых приборов или только тех, которые включенный в данный момент. Найти также бытовой прибор, который наиболее (наименее) энергозатратный.
31. **Пчелиная пасека (*Bee Apiary*)**. На пасеке есть несколько домиков с ульями пчёл. Каждая пчела собирает за день определённое количество мёда и приносит его в свой улей. Необходимо подсчитать общее количество мёда в день,

которая даёт вся пасека, а также определить улей, которые даёт больше (меньше) всего мёда.

32. **Поликлиника (*Clinic/Hospital*)**. В городе есть несколько поликлиник (больниц), куда обращаются пациенты, у которых есть проблемы со здоровьем. Необходимо подсчитать общее количество обращений во все поликлиники (больницы) города, а также найти самую (менее) загруженную по посещению поликлинику (больницу).
33. **Библиотека (*Library*)**. Библиотека имеет собственный книжный фонд и предоставляет своим читателям книги (и другую литературу), которые можно почитать как в самой библиотеке, так и взять с собой на определённый период. Необходимо подсчитать, сколько сейчас книг на руках у читателей, а также определить самую (менее) популярную книгу у читателей.
34. **Тюрьма (*Jail/Prison*)**. Есть тюрьма, в которой содержатся заключённые, осуждённые по разным статьям и на различный срок. Необходимо подсчитать суммарное количество дней (или месяцев, или лет, ...) всех заключённых, а также найти самую (менее) популярную уголовную статью, за которую отбывают наказание осуждённые, или найти самый большой (короткий) срок заключённого.
35. **Морской порт (*Sea Port*)**. Есть морской порт с пирсами (причалами), на которых каждый день происходит выгрузка и загрузка соответствующих грузов. Необходимо подсчитать общий грузопоток, который проходит через данный порт за сутки (или любой другой период), а также определить максимально (минимально) загруженный пирс (причал).
36. **Суд или судебное делопроизводство (*Court*)**. В городе есть суд, который каждый день рассматривает уголовные дела или другие правонарушения, касающиеся нарушения закона. Необходимо подсчитать: общее количество дел, которые были рассмотрены судом за отчётный период; сколько из общих дел были оправдательными, а сколько – с доказанной виной и повлёкшим к заключению под стражу виновных. Можно дополнительно подсчитать общее число лиц, которые были осуждены на соответствующие сроки и сейчас находятся в местах отбывания наказания, а также найти осуждённых, которым судья установил максимальный (минимальный) срок.

37. **Видео/Компьютерная игра (Video/Computer Game).** Есть крутая игра, направленная на зарабатывание игроками в процессе игры определённого количества «качества» – это может быть соответствующие баллы, время, уровни, виртуальные деньги, алмазы, брильянты, золото, другие виртуальные ресурсы и т.д. Необходимо определить максимальное количество «качества», заработанное всеми игроками в данной видео игре, а также выявить победителя (– игрок с максимальным количеством «качества») и аутсайдера (– игрок с минимальным количеством «качества») на данный момент.
38. **Продуктовый магазин или любой другой магазин (Grocery Store).** Есть магазин, в который каждый день приходят покупатели за продуктами. Необходимо подсчитать среднюю выручку магазина за сутки (или за месяц, или год и т.д.), а также найти покупателя, который оставил в магазине само больше (меньше) денег за отчётный период.
39. **Ресторан (Restaurant).** Есть ресторан соответствующей кухни, который своим посетителям предоставляет разнообразное меню блюд. Необходимо подсчитать общую сумму заказа соответствующего столика или общее количества денег, который ресторан получил за отчётный период (день, месяц и т.д.). Также необходимо найти чек (столлик) с максимальной (минимальной) суммой заказа.
40. **Автостоянка (Parking).** В городе есть автостоянка (или несколько автостоянок), стоимость машиномест на которой тарифицируется по часам. Необходимо подсчитать общую сумму, которую получает автостоянка за сутки (или любой другой отчётный период), а также автовладельца, который суммарно заплатил больше (меньше) всего денег за парковку.

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



Что нужно запомнить (краткие тезисы)

1. Процедурное программирование описывает структуры данных и алгоритмы.
2. Наш мозг оперирует образами и объектами – доказанный факт с 1974 года.
3. **ООП** использует в качестве базовых элементов **объекты**, а не алгоритмы.
4. Использование объектного подхода – это естественный путь решать задачи реального мира с помощью компьютера. Это просто, понятно и проверено временем.
5. Методология ООП переводит программирование в **моделирование**. Другими словами, ООП позволяет упрощать сложные вещи через моделирование.
6. **Объект** – это понятие, абстракция или любой предмет с чётко очерченными границами, имеющий смысл в контексте рассматриваемой прикладной проблемы. Введение объекта преследует две цели: *понимание прикладной задачи (проблемы)* и *введение основы для реализации на компьютере*.
7. **Главная идея ООП – всё состоит из объектов!** Программа, написанная с использованием ООП, состоит из множества объектов, и все эти объекты взаимодействуют между собой посредством посылке (передачи) сообщений друг другу. ООП и реальный мир не могут существовать раздельно!
8. **Программные объекты** могут представлять собой объекты реального мира или быть полностью абстрактными объектами, которые могут существовать только в рамках программы. **Гради Буч** (создатель унифицированного языка моделирования UML) даёт следующее определение объекта: «**Объект – это мыслимая или реальная сущность, обладающая характерным поведением, отличительными характеристиками и являющая важной в предметной области**».
9. **Каждый объект** имеет состояние, обладает некоторым хорошо определённым поведением и уникальной идентичностью.
10. **Состояние (state)** (синонимы: параметры, аспекты, характеристики, свойства, атрибуты, ...) – совокупный результат поведения объекта: одно из стабильных условий, в которых объект может существовать, охарактеризованных количественно; в любой конкретный момент времени состояние объекта включает в себя перечень параметров объекта и текущее значение этих параметров.

11. **Поведение (*behavior*)** – действия и реакции объекта, выраженные в терминах передачи сообщений и изменения состояния; видимая извне и воспроизводимая активность объекта.
12. **Уникальность (*identity*)** – эта природа объекта; то, что отличает один объект от других. В машинном представлении уникальность объекта – это адрес размещения объекта в памяти. Следовательно, уникальность объекта состоит в том, что всегда можно определить, указывают две ссылки на один и тот же объект, или на разные объекты.
13. **Эквивалентность (*equivalence*)** объектов – это когда состояние сравниваемых объектов совпадает, т.е. соответствующие атрибуты состояния объектов равны.
14. Чтобы создать программный объект или группу объектов необходимо вначале описать где-то его(их) характеристики и шаблон поведения. Для этих целей в ООП существуют классы (***classes***). Формально, **класс** – это шаблон поведения объектов определённого типа с определёнными параметрами, которые описывают состояние объекта.
15. **Все экземпляры** (объекты) одного и того же класса имеют один и тот же набор характеристик (значение которых может быть различным у разных объектов) и общее поведение (все объекты одинаково реагируют на одинаковые сообщения).
16. В упрощённом виде, класс – это кусок кода, у которого есть имя. Обычно чтобы воспользоваться функциональностью данного кода, нужно создать объект (**экземпляр класса**).
17. **Инстанцирование** – процесс создания и инициализации объекта (экземпляра класса).
18. Абстракция в философии – это мысленное отвлечение от ряда свойств предметов и отношений между ними; понятие, образуемое в результате отвлечения.
19. **Абстракция (абстрагирование)** в программировании – процесс выделения (отделения от всего остального) из предметной (проблемной) области **существенных деталей** и их реализация в программной среде.
20. Объект – простейший вид абстракции в ООП.
21. Абстракция в ООП пытается решить сложность, скрывая ненужные детали от программиста.

22. Во многих объектно-ориентированных языках программирования класс также является объектом. В таком случае состояние и поведение такого объекта (объекта класса **Class**) выражается в виде статических полей и методов.
23. В основном статическая функциональность класса существует в единственном экземпляре для каждого класса в системе и является логическим способом организации функциональности, общей для любого объекта заданного класса. Т.е. это возможность создать дополнительный набор самодостаточного кода (функционала), который будет работать на уровне всей системы для заданного типа объектов.
24. Ещё один взгляд на классы и объекты: **класс – это набор функций, объект – это набор данных.**

25. Пример: класс - это Бог, который есть всегда. И Бог по образу и подобию своему создаёт экземпляры – людей. Каждый человек наделён своим состоянием – динамические поля (у него есть рост, вес, ...) и поведением – динамические методы (он может плавать, ходить, летать, учиться ...). Бог в свою очередь имеет свои глобальные характеристики (статические поля) и наделён общими для всех своих объектов функциями (статические методы): принимать мольбы, карать, посылать манну небесную и сотворить чудо. Это поведение может быть приватным, т.е. Бог, например, может принимать мольбы только людей, а не марсиан. А вот сотворить чудо может как для людей, так и для марсиан – ему же не жалко ☺.



26. Использование парадигмы ООП в программировании помогает:
- ✓ уменьшить **сложность** программного обеспечения (ПО);
 - ✓ увеличить **производительность** труда программистов и **скорость** разработки ПО;
 - ✓ повысить **надёжность** ПО;
 - ✓ обеспечить возможность лёгкой **модификации** отдельных компонентов ПО без изменения остальных его частей;

- ✓ обеспечить возможность **повторного использования** отдельных компонентов ПО;
- ✓ ... и многое другое.

27. У парадигмы ООП также есть некоторые минусы:

- ✓ много времени и сил тратится на размышления об абстракциях и шаблонах проектирования вместо решения реальных проблем;
- ✓ на чтение объектно-ориентированного кода тратится больше времени, чем на его написание;
- ✓ увеличивается объём кода;
- ✓ увеличиваются накладные расходы на ресурсы;
- ✓ снижается скорость выполнения программ;
- ✓ требует много труда, которое окупается на больших проектах.

28. ООП – это инструмент, подходящий для решения определенных задач, а не всего подряд. Нет ничего идеального и совершенного, ООП не панацея!

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.

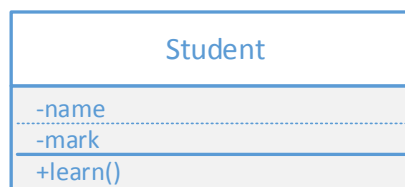
Словарь UML включает три вида строительных блоков: **диаграммы, сущности** и **связи**. **Сущности** – это абстракции, которые являются основными элементами модели, **связи** соединяют их между собой, а **диаграммы** группируют представляющие интерес наборы сущностей.

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.

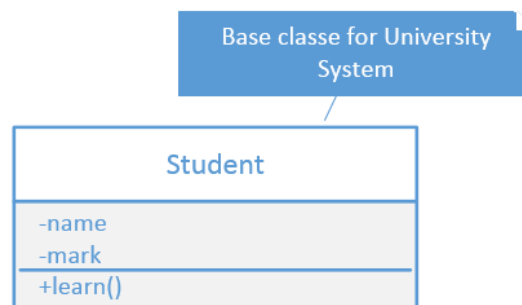
Структурные сущности – это «имена существительные» в модели UML. В основном, статические части модели, представляющие либо концептуальные, либо физические элементы. Основным видом структурной сущности в диаграммах классов является класс. Пример класса Студент (*Student*) с полями и методами:



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*).

Для **абстрактного класса** имя класса записывается **курсивом**.

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 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).

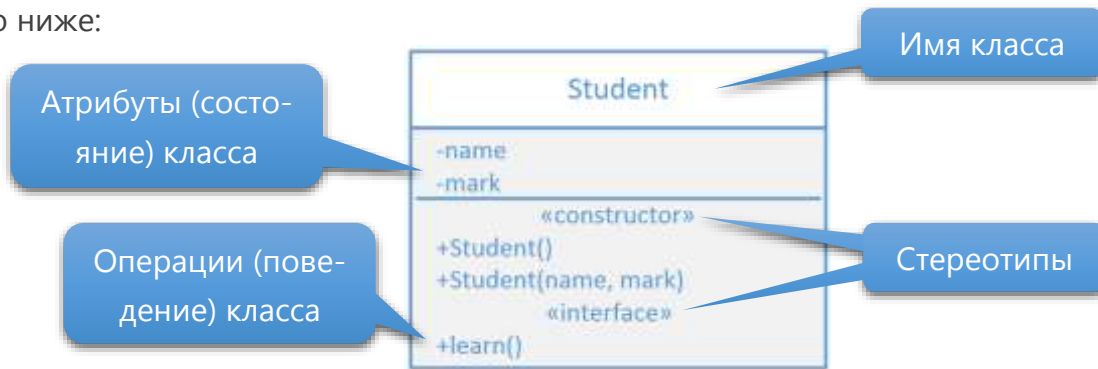
Можно специфицировать операцию, устанавливая ее сигнатуру, включающую имя, тип и значение по умолчанию всех параметров, а применительно к функциям – тип возвращаемого значения.

Абстрактные методы класса обозначаются **курсивным** шрифтом.

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.

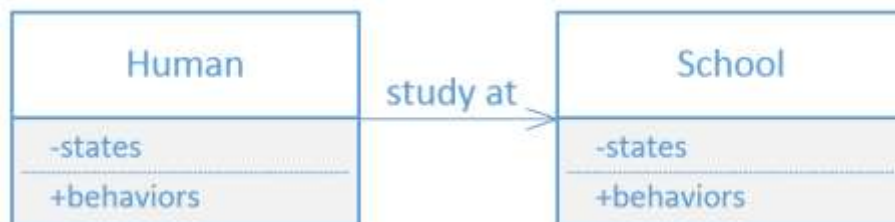
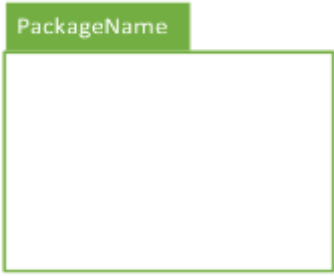
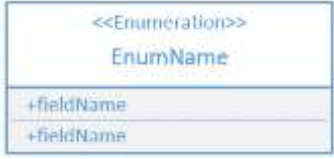




Таблица 1 – Наиболее часто используемые элементы UML-диаграммы

#	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>)

Пример выполнения практического задания с использованием Java-классов и объектов, а также архитектурного шаблона проектирования MVC

Задание

В группе есть три студента: Иванов, Петров и Сидоров. Спроектируйте и реализуйте программную систему, которая бы хранила информацию о данных студентах (к примеру, имя и оценку) и подсчитывала среднюю успеваемость студентов группы.

Решение

- 1) Выделим основные сущности нашей предметной области: класс-сущность *Студент* (***Student***), класс бизнес логики *Заведующий кафедрой* (***Manager***), утилитный класс для вывода данных *Принтер* (***Printer***) и класс-контроллер ***Main***. Согласно предметной области спроектируем UML-диаграмму взаимодействия классов и объектов будущего приложения:

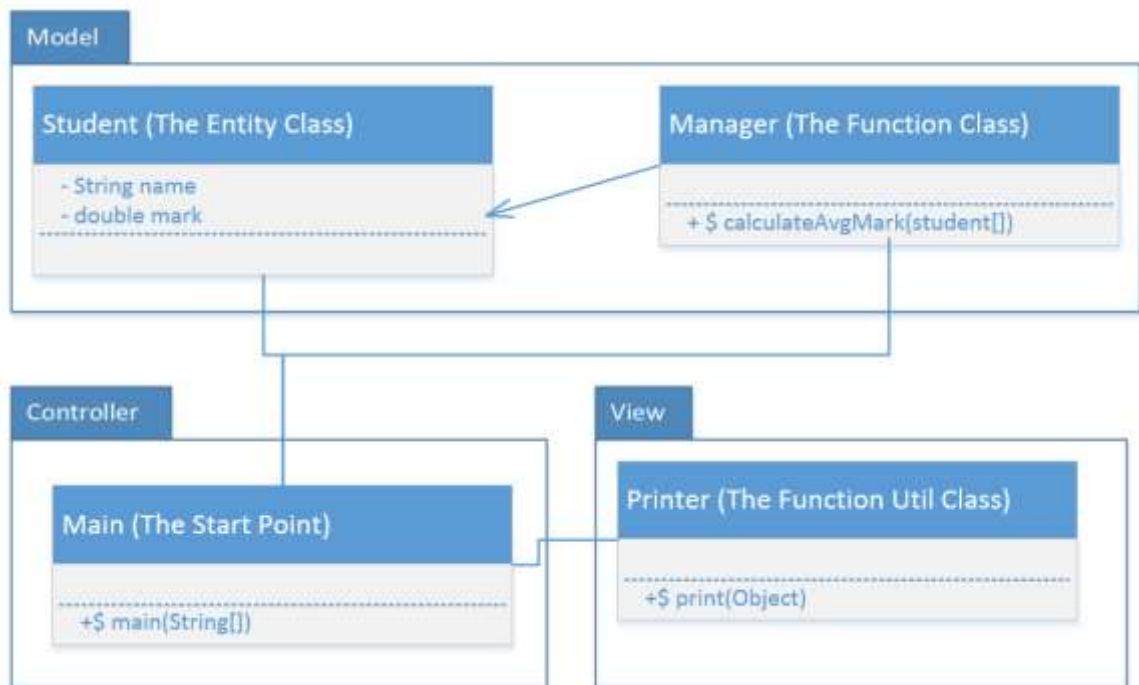


Рисунок 1 – UML-диаграмма предметной области University

- 2) Опишем сущность **Student**. Данная сущность ответственна за хранение соответствующей информации о студентах. Класс данной сущности содержит в себе Ф.И.О. студента (**name**), оценку (**mark**) и переопределённый метод `toString()`:

```
package by.bntu.fitr.poisit.vikvik.university.model.entity;
```

```
public class Student {
```

```
    public String name;  
    public int mark;
```

```
    @Override
```

```
    public String toString() {  
        return name + " (" + mark + ")";  
    }
```

```
}
```

Имя класса должно быть именем существительным, заданным в единственном числе

- 3) Разберём более детально сущность **Student**:

```
package by.bntu.fitr.poisit.vikvik.university.model.entity;
```

```
public class Student {
```

```
    public String name;  
    public int mark;
```

```
    @Override
```

```
    public String toString() {  
        return name + " (" + mark + ")";  
    }
```

```
}
```

Поля уровня объекта (экземпляра класса) для хранения его состояния

Переопределение метода, который обычно автоматический вызывается там, где требуется строковое представление состояния объекта

- 4) Теперь реализуем бизнес-логику программы (составная часть модели согласно паттерну MVC) в соответствующем статическом методе класса **Manager** – `calculateAvgMark(Student[] students)`. Данный метод предназначен для вычисления средней успеваемости группы студентов. Он принимает на вход один параметр – ссылочную переменную, которая ссылается на объект группы, и возвращает среднеарифметическое трёх оценок соответствующих студентов группы:

```
package by.bntu.fitr.poisit.vikvik.university.model.Logic;
```

```
import by.bntu.fitr.poisit.vikvik.university.model.entity.Student;
```

```

public class Manager {
    public static double calculateAvgMark(Student[] students) {
        int total = 0;

        for (int i = 0; i < students.Length; i++) {
            total += students[i].mark;
        }

        return total / students.Length;
    }
}

```

Находим сумму оценок всех студентов

Находим среднюю успеваемость студентов

- 5) Один из последних классов, который необходимо реализовать, это класс отображения данных с использованием системной консоли **Printer** – компонент View согласно архитектурному шаблону MVC:

```
package by.bntu.fitr.poisit.vikvik.university.view;
```

```

public class Printer {
    public static void print(Object msg) {
        System.out.print(msg);
    }
}

```

Вывод данных с использованием системной консоли

- 6) На заключительном этапе соберём из разработанных компонентов (классов) готовую программу. Для этого опишем класс **Main**, который будет выполнять роль контроллера согласно архитектурному шаблону MVC. В нём будет описан стартовый статический метод **main(...)**:

```
package by.bntu.fitr.poisit.vikvik.university.controller;
```

```
import java.util.Arrays;
```

```

import by.bntu.fitr.poisit.vikvik.university.model.entity.Student;
import by.bntu.fitr.poisit.vikvik.university.model.logic.Manager;
import by.bntu.fitr.poisit.vikvik.university.view.Printer;

```

```

public class Main {
    public static void main(String[] args) {

        Student ivanov = new Student();
        ivanov.name = "Ivanov";
        ivanov.mark = 10;

        Student petrov = new Student();
        petrov.name = "Petrov";
        petrov.mark = 5;
    }
}

```

Класс-контроллер для связывания всех разработанных компонентов в единую программу для получения результата

```

Student sidorov = new Student();
sidorov.name = "Sidorov";
sidorov.mark = 9;

Student[] group = { ivanov, petrov, sidorov };

double avgGroupMark = Manager.calculateAvgMark(group);

Printer.print(Arrays.toString(group));
Printer.print("\nAvg group mark = " + avgGroupMark);
}
}

```



Обратите внимание, как **приятно читать** и **сопровождать** описанный код. Рекомендуется следовать такому же стилю написания программного кода на Java

7) В общем виде архитектура приложения представлена ниже на рисунке:

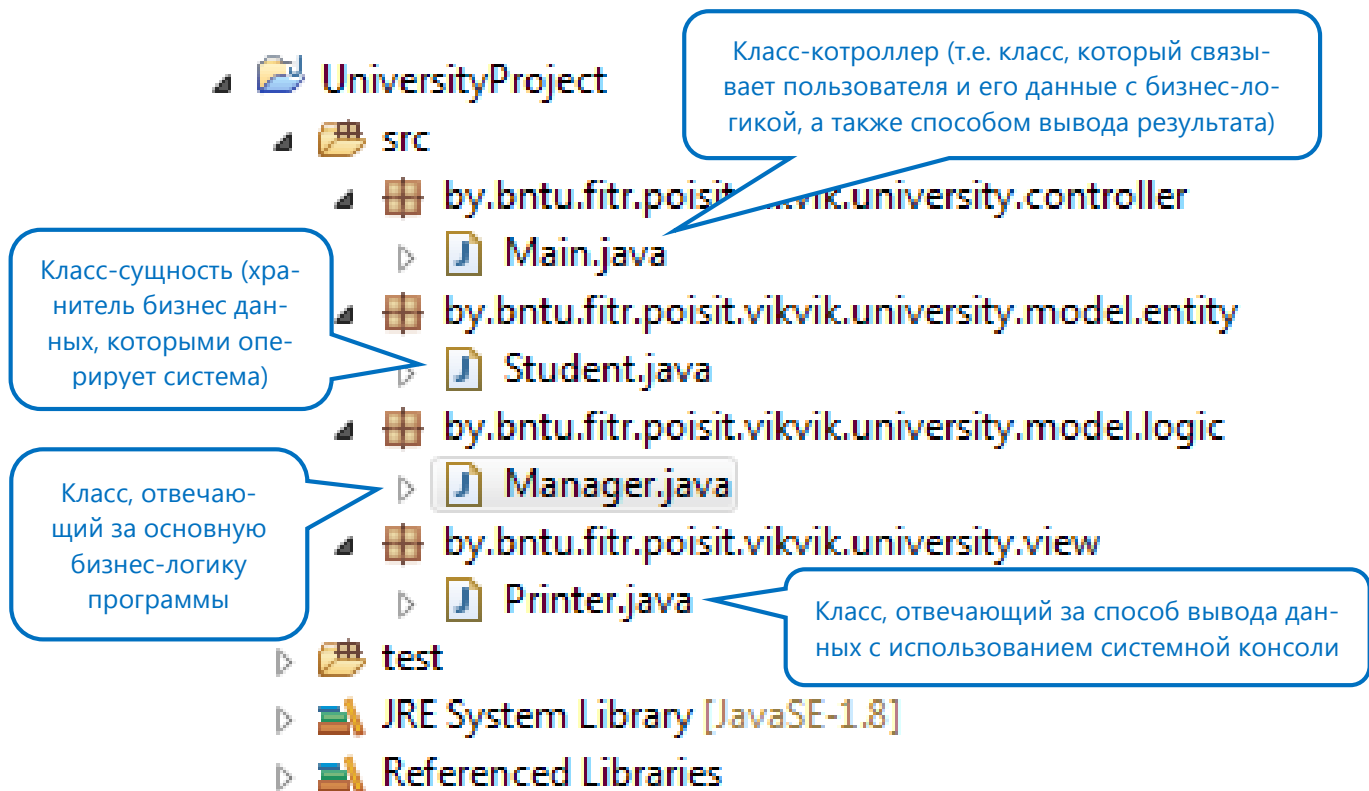
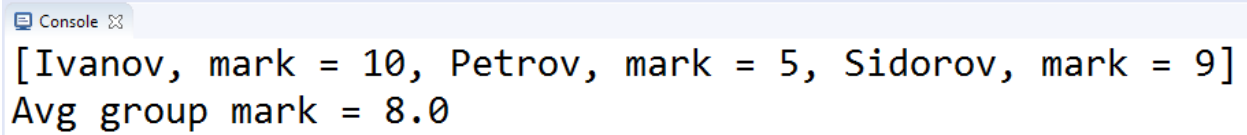


Рисунок 2 – Архитектура разработанного приложения

- 8) Для демонстрации работы программы перекомпилируем разработанный проект и запустим стартовый класс *Main* на выполнение:

A screenshot of a console window with a light blue header bar containing the word "Console" and a small icon. The console displays two lines of text: "[Ivanov, mark = 10, Petrov, mark = 5, Sidorov, mark = 9]" and "Avg group mark = 8.0".

```
[Ivanov, mark = 10, Petrov, mark = 5, Sidorov, mark = 9]
Avg group mark = 8.0
```

Рисунок 3 – Результат работы программы

Как можно улучшить вышеописанный код?

У вышеизложенного варианта реализации задания есть ряд серьёзных ошибок, которые могут привести к краху всей программы или к неверному результату. Попробуйте найти и устранить данные ошибки.

Контрольные вопросы

- 1) Какова была главная идея создания процедурных языков программирования? На что эти языки программирования ставили акцент и что являлось исходным понятием методологии?
- 2) К чему не были готовы процедурные языки программирования в начале 2000 годов? Какой был кризис в сфере разработки программного обеспечения?
- 3) Суть объектно-ориентированного подхода и его главный лозунг?
- 4) Опишите базовые концепции, которые лежат в основе методологии ООП.
- 5) Что такое объект в реальном мире? Чем он характеризуется в реальном мире? (*тип, состояние, поведение, уникальность, эквивалентность*)
- 6) Что такое объект в программировании и для чего он вводится при моделировании будущей программной системы?
- 7) Чем характеризуется объект в программном (виртуальном) мире? Как реализуются данные вещи в объектно-ориентированных языках программирования?
- 8) Какая разница между уникальностью и эквивалентностью?
- 9) Какую роль выполняет класс в ООП?
- 10) Как и что можно описать в Java-классе?
- 11) Что представляет собой абстракция в ООП?
- 12) Простейшая форма абстракции в ООП?
- 13) Какие разновидности классов существуют в ООП?
- 14) Каким способом можно создать объект (экземпляр класса) в языке Java?
- 15) Как можно обратиться к полям и методам объекта в языке Java?
- 16) Какова разница между классом и объектом в языке Java?
- 17) Перечислите преимущества и недостатки объектно-ориентированного программирования.
- 18) Что такое UML?
- 19) Что отображает и зачем нужна UML-диаграмма классов?
- 20) Как с помощью UML можно описать классы и их взаимосвязь друг с другом (т.е. описать)?

7 смертных грехов программирования



Усман Шаукат, опыт в сфере веб-разработки: PHP, JavaScript, Node.js, ...

1. **Программировать, не планируя.** Самый страшный из всех грехов.
2. **Пытаться изобрести колесо.** Если есть возможность, всегда используйте алгоритмы, предложенные в книгах и научных статьях (например, алгоритмы сортировки, поиска и т.д.), а не пишите собственные.
3. Писать несистематизированные/некачественные коды и **не придерживаться стандартов программирования.**
4. Считать, что тестирование – это не ваша забота. Я вас очень прошу, пожалуйста, **тестируйте свои коды.**
5. Писать сложный код, когда с тем же успехом можно обойтись простым. **Простые коды – это элегантно.**
6. Слепое копирование-вставка с сайтов вроде stackoverflow.com без ознакомления с пояснениями и комментариями.
7. Последнее, и самое важное – **совершенствуйтесь сами и осваивайте новый инструментарий.** Никогда не бойтесь новшеств. Знакомьтесь с ними раньше всех. Это поможет вам оставаться востребованным.

Source: <https://www.kv.by/post/1053298-7-smertnyh-grehov-programmirovaniya>