МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ

КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Математическая кибернетика и информационные технологии»

Лабораторная работа №2

**по дисциплине**

**«Информационные технологии и программирование»**

Выполнил: студент гр. БВТ2201

Аблязов И.В.

Проверил:

Москва, 2023 г.

# ВВЕДЕНИЕ

Цель задания: создать иерархию классов в соответствии с вариантом номер 3 (Базовый класс: Человек, Дочерние классы: Студент, Преподаватель, Ассистент преподавателя). Иерархия должна содержать:

* абстрактный класс
* 2 уровня наследуемых классов (классы должны содержать в себе минимум 3 поля и 2 метода, описывающих поведение объекта)
* демонстрацию реализации всех принципов ООП (абстракция, модификаторы доступа, перегрузка, переопределение)
* наличие конструкторов (в том числе по умолчанию)
* наличие геттеров и сеттеров
* ввод/вывод информации о создаваемых объектах
* предусмотрите в одном из классов создание счетчика созданных объектов с использованием статической переменной, продемонстрируйте работу

# ОСНОВНАЯ ЧАСТЬ

Для начала был создан абстрактный класс Human, который хранит в себе имя, фамилию, возраст и пол человека. Абстрактным называют класс, экземпляр которого невозможно создать. Весь функционал таких классов раскрывается в дочерних классах. Также можно заметить статическую переменную count. С помощью неё мы сможем узнать, сколько было создано подклассов класса Human. Для этого будет увеличивать ее значение на один при каждом вызове конструктора, указанного ниже. В нем происходит инициализация всех переменных данного класса.

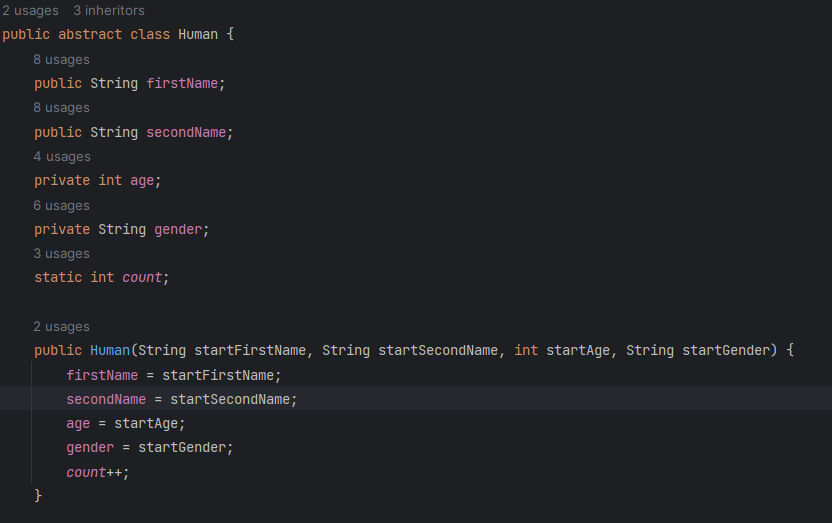


Рисунок 1 Класс Human

Можно заметить, что некоторые переменные являются публичными, а некоторые – приватными. Так реализован принцип модификации доступа. Подклассы класса Human смогут получать значения имени и фамилии напрямую, но для возраста и пола придется воспользоваться методами.

Одними из самых известных методов являются геттеры и сеттеры. Они выдают или, соответственно, заменяют необходимую переменную класса. Но это не мешает создавать любой необходимый нам метод.



Рисунок 2 Методы класса Human

Рассмотрим теперь дочерний класс Student. Наследование класса указывается после его названия с помощью extends «название наследуемого класса». В данном подклассе также реализован конструктор. В стандартной конструкторе данные, принадлежащие к основному классу указываются командой super(). В конструкторе по умолчанию мы заданию начальные значения с помощью команды this().

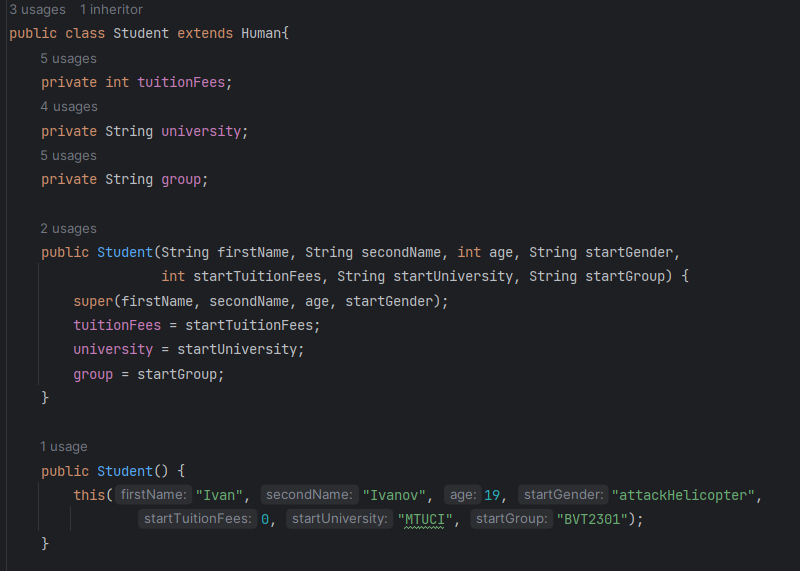


Рисунок 3 Подкласс Student

Рассмотрим случай, когда нам необходимо переписать метод, который уже существует в родительском классе. Для это до начала объявления метода прописываем @Override как показано на рисунке 4. Это называется принципом переопределения.

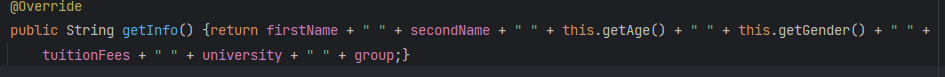


Рисунок 4 Метод переопределения

По похожим принципам создадим ещё 2 подкласса – Teacher и TeachingAssistant. Полный код данных подклассов можно подробнее изучить в репозитории GitHub. Из интересного стоит выделить методы подкласса Teacher, указанные на рисунке 5.

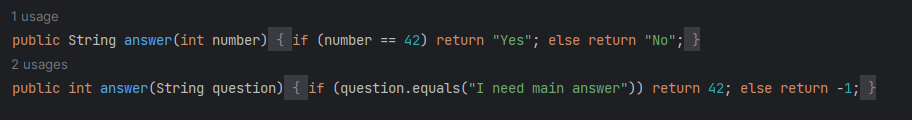


Рисунок 5 Метод перегрузки

Как мы можем заметить, они обладают одинаковым названием, но тип входных и выходных данных различается. Данный принцип называется перегрузкой. В зависимости от входных данных мы можем использовать необходимую нам логику.

Перейдем к проверке кода. Для начала создадим экземпляр класса Student и попробуем обратиться к его методам.

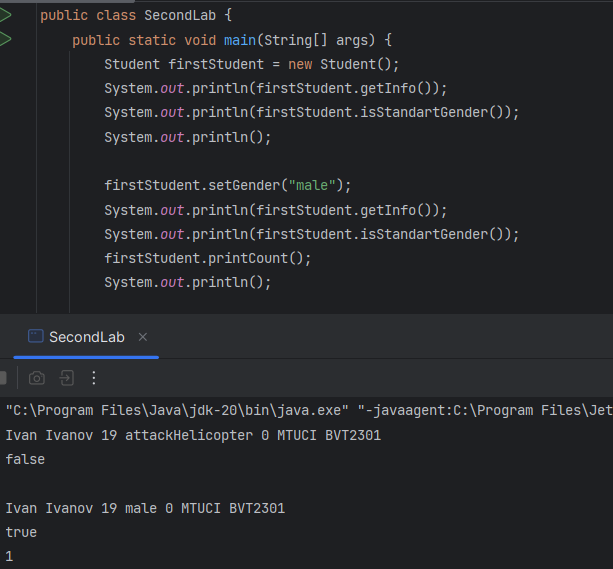


Рисунок 6 Пример создания экземпляра класса и вызова его методов (1)

Как можно заметить из рисунка 6, наш экземпляр класса принял те значения, которые были выставлены по умолчанию. Получилось также изменить данные класса с помощью сеттера setGender(). Обратившись к методу класса Human printCount() получаем, что пока был создан только 1 экземпляр класса Human.

Далее создадим экземпляр класса Teacher. Так как метод получения полной информации был переписан, при использовании метода getInfo() мы получим слегка другую строку. Также можно заметить, что нам не важно, к какому из экземпляров применять метод вывода количества экземпляров.

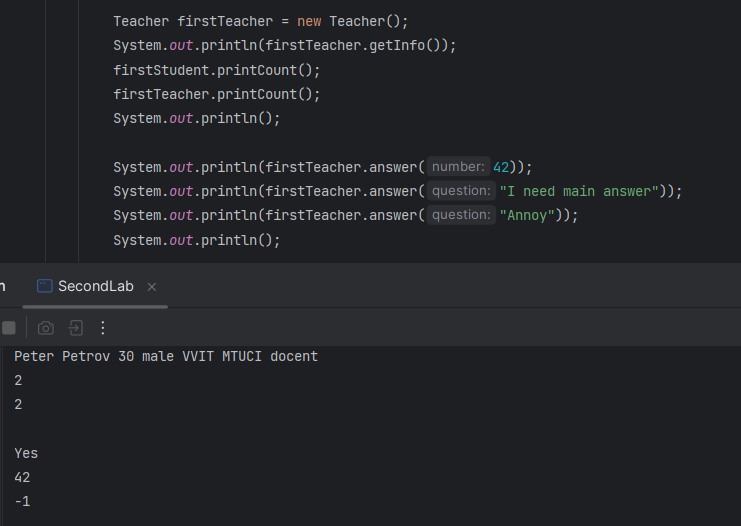


Рисунок 7 Пример создания экземпляра класса и вызова его методов (2)

Также из рисунка виден метод реализации перегрузки методов.

В качестве последнего примера создадим экземпляр класса TeachingAssistant. Так как в конструкторе указана необходимость передачи экземпляра класса учитель, мы указываем ранее созданный экземпляр.

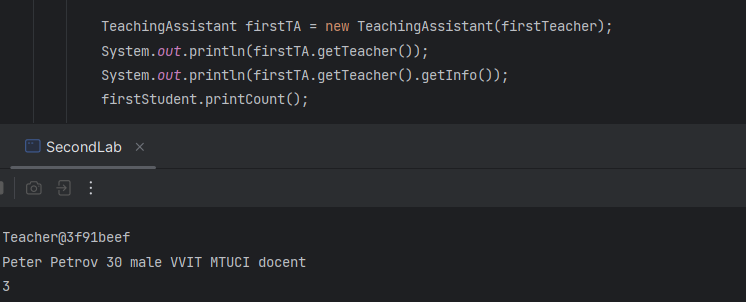


Рисунок 8 Пример создания экземпляра класса и вызова его методов (3)

Можно также заметить, что мы можем взаимодействовать со всеми методами подкласса Teacher, который находится в подклассе TeachingAssistant.

# ВЫВОД

Данная лабораторная работа помогла лучше освоить язык Java. Была проделана работа с классами и подклассами. Помимо стандартного наследования были усвоены такие принципы, как абстракция, перегрузка и переопределние. Все знания были закреплены на практике и сохранены на GitHub: <https://github.com/HungryM1NT/ITIP>