# Пояснительная записка

# Для задания №3 по архитектуре вычислительных систем

Мельник Всеволод
БПИ207
10 октября 2021 г.

# Описание полученного задания

Разработать консольное приложение, позволяющее работать с контейнером трёхмерных геометрических фигур. Фигура может быть шаром с целочисленным радиусом, параллелепипедом с тремя целочисленными рёбрами или правильным тетраэдром. Каждая фигура имеет плотность. Также программа может удалить из контейнера те фигуры, для которых площадь поверхности меньше, чем средняя площадь поверхности всех фигур в контейнере.

Работа с программой осуществляется из командной строки одним из следующих способов:

***<Имя программы> -f <входной файл> <выходной файл> <выходной файл для контейнера после выполнения функции 19>*** – ввод данных в программу из файла.

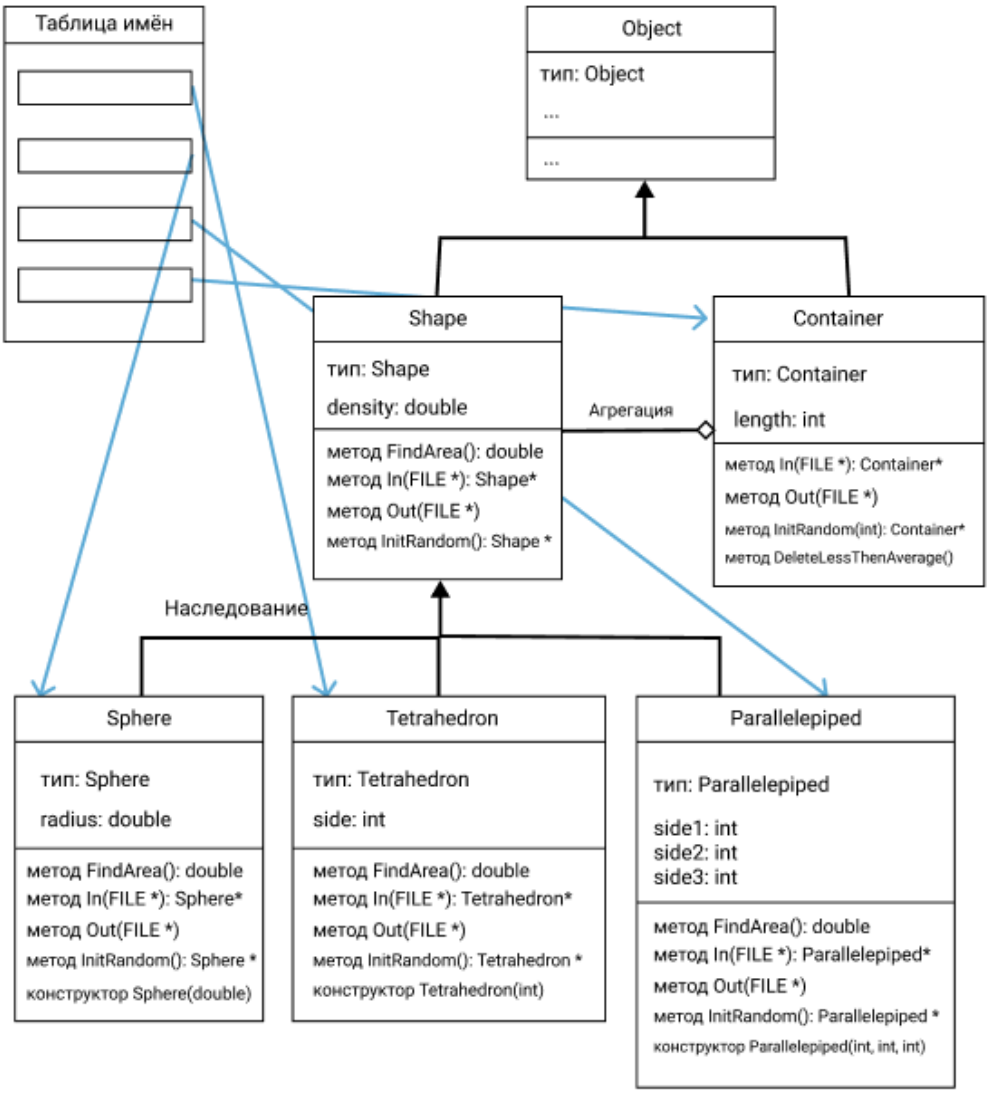
***<Имя программы> -r <количество фигур> <выходной файл> <выходной файл для контейнера после выполнения функции 19>*** – случайная генерация фигур для заполнения контейнера.

***<Имя программы> -h*** – справка по программе.

**Важно**: вывод контейнера после удаления фигур осуществляется в отдельный файл для упрощения работы с программой и её тестированием. Его указывать не обязательно. Если этот файл не указан, удаление и последующий вывод осуществляться **не будет**.

Тесты храняться в папке tests, ответы на тесты - tests_answers, ответы на тесты после выполнения функции удаления - tests_function_answers.

# Структурная схема изучаемой архитектуры ВС с размещенной на ней разработанной программы



# Метрики, определяющие характеристики программы

Код программы содержит 3 модуля.

Исходный код занимает ~ 8 КБ памяти, примерно на 3 КБ меньше, чем в предыдущем задании, исполняемый файл не генерируется, так как Python является интерпретируемым языком программирования.

Время выполнения для одной фигуры - 0,117 секунд, для 5000 фигур - 0,164 секунд, для 10000 фигур - 0,381 секунд. Программа работает примерно в 5,6 раз медленнее, чем предыдущая, которая была написана на статически типизированном  компилируемым языке. Это объясняется тем, что в среднем интерпретируемые языки программирования медленнее, чем компилируемые.

Динамическая типизация позволила ускорить написание программы, однако отладка заняла больше времени, так как тип объектов не известен до запуска, что усложнило поиск ошибок.