Лабораторная работа №2

**Задача 1.2. Человек**

**Текст задачи:**

Создайте сущность Человек, которая описывается:

\*Имя: строка

\*Рост: целое число

Может возвращать текстовое представление вида “Name, рост: height”, где Name и height это

переменная с именем и ростом.

Необходимо создать и вывести на экран следующих людей:

\*Человек с именем “Клеопатра” и ростом 152

\*Человек с именем “Пушкин ” и ростом 167

\*Человек с именем “Владимир ” и ростом 189

**Описание:**

При создании объекта Human инициализируются поля name и height. Метод toString() формирует строку путем конкатенации имени и роста.

**Тестирование:**

Создан объект Human с именем "Клеопатра" и ростом 152.

Вывод в консоль: "Клеопатра, рост: 152".

Создан объект Human с именем "Пушкин" и ростом 167.

Вывод в консоль: "Пушкин, рост: 167".

Создан объект Human с именем "Владимир" и ростом 189.

Вывод в консоль: "Владимир, рост: 189".

**Задача 1.3. Имена**

Текст задачи:

Создайте сущность Имя, которая описывается тремя параметрами: Фамилия, Личное имя, Отчество. Имя может быть приведено к строковому виду, включающему традиционное представление всех трех параметров. Необходимо предусмотреть возможность того, что какой-либо из параметров может быть не задан.

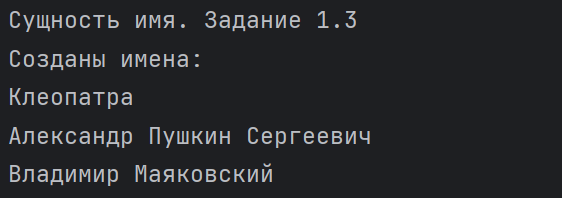
Необходимо создать следующие имена:

• Клеопатра • Пушкин Александр Сергеевич • Маяковский Владимир

**Описание:**

Метод toString() проверяет наличие значений в полях. Если все три поля заполнены, возвращает ФИО в порядке "Фамилия Имя Отчество". Если отсутствует фамилия или отчество, возвращает только имеющиеся части.

Тестирование:



**Задача 2.2. Человек с именем**

**Текст задачи:**

Объедините сущности Человек и Имя таким образом, чтобы имя человека задавалось с использованием сущности Имя, а не строки.

Необходимо объединить ранее созданные объекты имен и людей, с получением:

• Человека с Именем Клеопатра и ростом 152

• Человека с Именем Пушкин Александр Сергеевичи ростом 167

• Человека с Именем Маяковский Владимир и ростом 189

**Описание:**

Класс Human модифицирован: поле name теперь имеет тип Name. Конструктор принимает объект Name и рост. Метод toString() возвращает строку в формате "ФИО, рост: число". Метод toString() использует метод toString() объекта Name для получения строкового представления имени.

**Тестирование:**

Созданы объекты Human с именами (через объект Name):

Клеопатра (рост 152), Пушкин Александр Сергеевич (рост 167), Владимир Маяковский (рост 189).

Вывод в консоль:

"Клеопатра, рост: 152",

"Пушкин Александр Сергеевич, рост: 167",

"Владимир Маяковский, рост: 189".

**Задача 2.3. Человек с родителем**

**Текст задачи:**

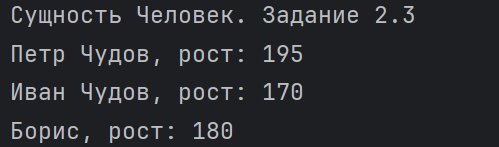
Измените сущность Человек, добавив возможность задавать отца. При приведении человека к строковой форме необходимо проверить параметры имени, если у человека нет фамилии и есть отец, взять фамилию отца (если она есть). Если нет отчества, а у отца есть имя - сгенерировать отчество.

**Описание:**

Класс Human дополнен полем father типа Human. Добавлены конструкторы с учетом отца. Метод correctName() автоматически дополняет фамилию и отчество на основе данных отца. Метод generatePatronymic() создает отчество по имени отца.

**Тестирование:**

Созданы объекты: Иван Чудов (отец), Петр Чудов (сын Ивана), Борис (сын Петра). Вывод в консоль:



**Задача 3.3. Города**

**Текст задачи:**

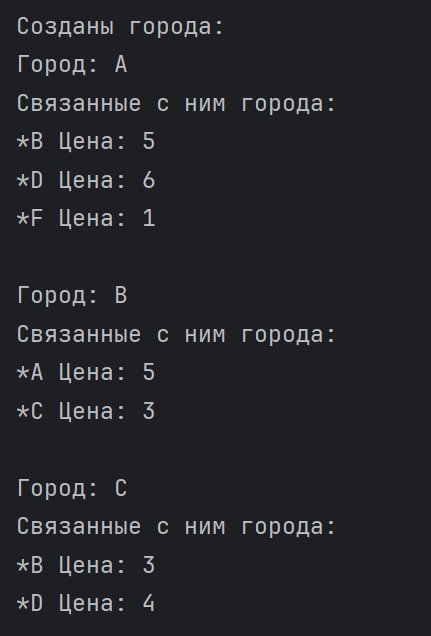
Создайте сущность Город, которая будет представлять собой точку на карте с названием и набором путей к другим городам с указанием стоимости поездки. Город может возвращать текстовое представление, в виде названия города и списка связанных с ним городов (в виде пары: “название:стоимость”).

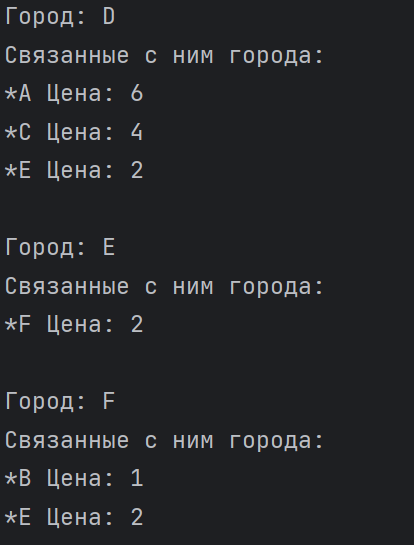
**Описание:**

Класс City содержит поля: cityName (строка), routes (Map<String, Integer> для хранения городов и стоимости путей). Конструкторы: по умолчанию, с именем города. При создании города инициализируется HashMap для хранения маршрутов. Метод addRoute() добавляет пару "город-стоимость" в карту. Метод toString() формирует строку с названием города и списком всех связанных городов с стоимостями.

**Тестирование:**

Созданы города A, B, C, D, E, F с различными маршрутами. Вывод в консоль:





**Задача 4.5. Создаем Имена**

**Текст задачи:**

Измените сущность Имя: можно создать, указав только имя, имя и фамилию, или все три параметра.

Необходимо создать следующие имена:

1. Клеопатра

2. Александр Сергеевич Пушкин

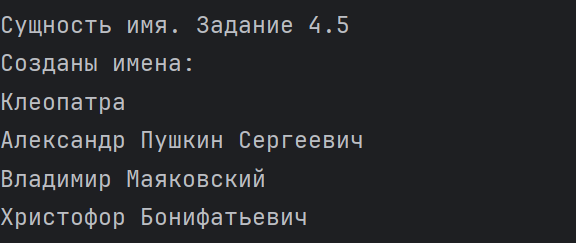
3. Владимир Маяковский

4. Христофор Бонифатьевич (здесь Христофор это имя, а Бонифатьевич - фамилия)

**Описание:**

Класс Name содержит те же поля, что и в задаче 1.3. Добавлены конструкторы для разных комбинаций параметров. Метод toString() возвращает строку в зависимости от заполненных полей.

**Тестирование:**



**Задача 4.6. Создаем Человека**

**Текст задачи:**

Измените сущность Человек. Новые требования включают:

• Человека можно создать с указанием имени в виде строки и роста в виде целого числа. При таком способе задания имени должно считаться, что Человеку задано только личное имя.

• Человека можно создать с указанием имени в виде строки, роста в виде целого числа и отца в виде Человека. В этом случае необходимо проставить отчество в соответствии с именем отца и присвоить текущему человеку фамилию отца.

• Человека можно создать с указанием имени в виде объекта типа Имя (из задачи 4.5) и роста в виде целого числа.

• Человека можно создать с указанием имени в виде объекта типа Имя (из задачи 4.5), роста в виде целого числаи отца в виде Человека. В этом случае необходимо проверить что в Имени задано отчество и фамилия, и если они не заданы, то необходимо их задать (отчество в соответствии с именем отца и фамилию отца).

• Реализуйте описанные способы создания Человека таким образом, чтобы операции присвоения использовались только в одном из конструкторов.

• Необходимо модифицировать способ приведения Человека к строке, теперь текстовая форма должна быть представлена строкой: “Имя, рост”

Создайте и выведите на экран следующие объекты:

1. Человека с именем Лев (в виде строки) и ростом 170

2. Человека с именем Пушкин Сергей (как Имя), ростом 168 и отцом Львом (предыдущий Человек)

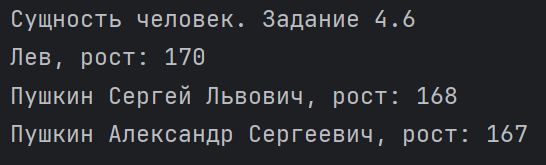
3. Человека с именем Александр (в виде строки), ростом 167 и отцом Сергеем (предыдущий Человек)

**Описание:**

Класс Human расширен множеством конструкторов для разных способов создания. Все конструкторы используют основной конструктор.

**Тестирование:**

Созданы объекты: Лев (по строке имени), Сергей Пушкин (по объекту Name с отцом Львом), Александр (по строке с отцом Сергеем).



**Задача 5.2. Кот мяукает**

**Текст задачи:**

Создайте сущность Кот, которая описывается следующим образом:

• Имеет Имя (строка)

• Для создания необходимо указать имя кота.

• Может быть приведен к текстовой форме вида: “кот: Имя”

• Может помяукать, что приводит к выводу на экран следующего текста: “Имя: мяу!”, вызвать мяуканье можно без параметров.

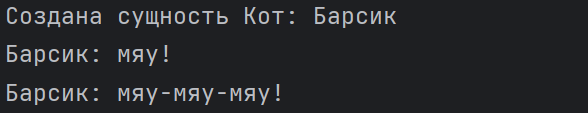
• Может помяукать Nраз, что приводит к выводу на экран следующего текста: “Имя: мяумяу-…-мяу!”, где количество “мяу”равно N.

Создайте кота по имени “Барсик”, и затем пусть он помяукает сначала один раз, а затем три раза.

Описание класса:

Класс Cat содержит поле name (строка). Конструкторы: по умолчанию, с именем. Методы: may() для однократного мяуканья, mayN(int) для N-кратного мяуканья, toString() для строкового представления.

**Тестирование:**



**Класс InputValidation**

Метод inputValid

Метод предназначен для проверки целочисленного ввода. Он принимает проверяемое значение test и границы диапазона min и max. Метод в цикле проверяет, принадлежит ли переданное значение заданному диапазону. Если значение некорректно, метод выводит сообщение об ошибке и запрашивает новый ввод с клавиатуры до тех пор, пока не будет введено число из правильного диапазона. Если ввод не является целым числом, метод выводит сообщение об ошибке, очищает буфер и ожидает новый ввод. Метод возвращает корректное целое число.

Метод inputLetters

Метод предназначен для проверки строки на содержание только буквенных символов и пробелов. Он принимает строку test, удаляет начальные и конечные пробелы. Затем метод проверяет каждый символ строки. Допустимыми считаются английские буквы (заглавные и строчные), русские буквы (включая 'ё' и 'Ё') и пробелы. Если встречается любой другой символ, метод выводит сообщение об ошибке и запрашивает новую строку с клавиатуры. Цикл продолжается до тех пор, пока не будет введена строка, состоящая только из разрешенных символов. Метод возвращает проверенную и обрезанную строку.

**Класс Name**

Приватные поля:

- String lastName: хранит фамилию.

- String firstName: хранит личное имя.

- String patronymic: хранит отчество.

Публичные геттеры:

- getLastName(): возвращает значение поля lastName.

- getFirstName(): возвращает значение поля firstName.

- getPatronymic(): возвращает значение поля patronymic.

Публичные сеттеры:

- setLastName(String lastName): устанавливает значение поля lastName, предварительно проверяя строку с помощью метода inputLetters.

- setFirstName(String firstName): устанавливает значение поля firstName, предварительно проверяя строку с помощью метода inputLetters.

- setPatronymic(String patronymic): устанавливает значение поля patronymic, предварительно проверяя строку с помощью метода inputLetters.

Конструкторы:

- Name(): конструктор по умолчанию. Инициализирует поля значениями "Иванов", "Иван", "Иванович".

- Name(String firstName): создает объект с заданным личным именем. Фамилия и отчество - пустые строки.

- Name(String firstName, String lastName): создает объект с заданными личным именем и фамилией. Отчество - пустая строка.

- Name(String firstName, String lastName, String patronymic): создает объект с заданными личным именем, фамилией и отчеством.

Метод toString()

Метод формирует строковое представление объекта Name. Сначала проверяется, заполнены ли все три поля (фамилия, имя, отчество). Если да, то возвращается строка в формате "Фамилия Имя Отчество". Если не все поля заполнены, метод последовательно добавляет в результирующую строку непустые значения полей в порядке: имя, фамилия, отчество, разделяя их пробелами. В конце метод удаляет лишние пробелы с начала и конца строки и возвращает результат.

**Класс Human**

Приватные поля:

- Name name: объект типа Name, хранящий имя человека.

- int height: рост человека.

- Human father: ссылка на объект типа Human, представляющий отца.

Публичные геттеры:

- getName(): возвращает объект name.

- getHeight(): возвращает значение поля height.

- getFather(): возвращает объект father.

Публичные сеттеры:

- setName(Name name): устанавливает объект name.

- setHeight(int height): устанавливает значение поля height, предварительно проверяя его на принадлежность диапазону от 100 до 300 с помощью метода inputValid.

- setFather(Human father): устанавливает объект father.

Конструкторы:

- Human(Name name, int height, Human father): основной конструктор. Инициализирует поля name, height и father. После инициализации вызывает метод correctName() для автоматической корректировки имени на основе данных отца.

- Human(): конструктор по умолчанию. Создает объект с именем "Иванов Иван Иванович" (через конструктор Name по умолчанию), ростом 100 и без отца.

- Human(String firstName, int height): создает объект, передавая строку firstName в конструктор Name с одним параметром, инициализирует рост.

- Human(String firstName, int height, Human father): создает объект, передавая строку firstName в конструктор Name с одним параметром, инициализирует рост и отца. После инициализации вызывается correctName().

- Human(Name name, int height): создает объект с заданными именем и ростом.

Приватные методы:

- correctName():

Метод проверяет, задан ли у текущего человека отец. Если отец задан, метод получает имя отца. Далее проверяется, если у текущего человека не задана фамилия, а у отца фамилия задана, то фамилия текущего человека становится такой же, как фамилия отца. Затем проверяется, если у текущего человека не задано отчество, а у отца задано личное имя, то для текущего человека генерируется отчество. Генерация отчества происходит в методе generatePatronymic.

- generatePatronymic(String fatherName):

Метод принимает имя отца и возвращает соответствующее отчество. Для некоторых имен используется специальный суффикс (например, для "Лев" - "Львович", для "Илья" - "Ильич"). Для остальных имен к имени отца добавляется суффикс "ович". Если имя отца пустое или null, возвращается пустая строка.

Метод toString()

Метод возвращает строковое представление человека в формате "Имя, рост: height", где Имя - это результат вызова метода toString() у объекта name, а height - значение поля роста.

**Класс Cat**

Приватные поля:

- String name: имя кота.

Публичные геттеры:

- getName(): возвращает значение поля name.

Публичные сеттеры:

- setName(String name): устанавливает значение поля name, предварительно проверяя строку с помощью метода inputLetters.

Конструкторы:

- Cat(): конструктор по умолчанию. Устанавливает name -"Кот".

- Cat(String name): создает объект с заданной кличкой.

Метод toString()

Описание метода: Метод возвращает строковое представление кота в формате "Кот: name".

Публичные методы:

- may():

Метод выводит на консоль сообщение в формате "name: мяу!".

- mayN(int a):

Метод выводит на консоль сообщение: мяуканье кота A раз. Сначала выводится "name: ", затем a-1 раз выводится "мяу-", и в конце выводится "мяу!". Таким образом, для a=3 вывод будет "name: мяу-мяу-мяу!".

**Класс City**

Приватные поля:

- String cityName: название города.

- Map<String, Integer> routes: хранение связей с другими городами. Ключом является название города-назначения, значением - стоимость поездки.

Публичные геттеры:

- getCityName(): возвращает название города.

- getRoutes(): возвращает карту routes.

Публичные сеттеры:

- setCityName(String name): устанавливает название города, предварительно проверяя строку с помощью метода inputLetters.

Конструкторы:

- City(): конструктор по умолчанию. Устанавливает cityName - "Какой-то город" и инициализирует routes пустой HashMap.

- City(String cityName): создает объект с заданным названием города и инициализирует routes пустой HashMap.

Публичные методы:

- addRoute(String city, int cost):

Метод добавляет новый маршрут в карту routes. Он принимает название города-назначения city и стоимость поездки cost. Пара записывается в карту, где city является ключом, а cost - значением.

- getAllRoutes():

Метод формирует строку, содержащую список всех маршрутов из города. Если карта routes пуста, возвращается строка "Нет связанных городов".

- toString():

Если маршрутов нет, возвращается строка с названием города и сообщением об отсутствии связей. Если маршруты есть, метод формирует строку с названием города, и на новых строках перечисляются все маршруты.

Класс Main

Case 1: Демонстрация сущности City.

Case 2: Демонстрация сущности Name.

Case 3: Демонстрация сущности Human.

Case 4: Демонстрация сущности Cat.