МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

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

Кафедра «Информационные технологии»

Информационные технологии и программирование

Лабораторная работа №2

Выполнила: Студентка группы

БВТ2207

Назарова Анна

Москва

2023

**Цель работы:** в данной лабораторной работе необходимо использовать классы, чтобы описать, как эти объекты работают.

**Введение.**

Объектно-ориентированное программирование (ООП) – это подход к разработке с помощью классов и объектов. Он позволяет писать значительно меньше кода и при этом реализовать больше возможностей. С помощью объектов и классов можно создать любой функционал быстрее.

Основные концепции ООП:

* **Абстракция** – выделение наиболее важных характеристик и информации об объекте. С помощью абстракции мы выделяем общее для всех объектов.
* **Инкапсуляция** – атрибуты и поведение объекта объединяются в одном классе, внутренняя реализация объекта скрывается от пользователя, а для работы с объектом предоставляется открытый интерфейс.
* **Полиморфизм** – когда программа может использовать объекты с одинаковым интерфейсом без информации о внутреннем устройстве объекта
* **Наследование.** Наследование заключается в использовании уже существующих классов для описания новых. Важнейшим преимуществом наследования является повторное использование кода, поскольку подклассы наследуют переменные и методы суперкласса.

Обычно при переопределении метода, используется аннотация **@Override**, которая подсказывает компилятору о необходимости проверить сигнатуры переопределяемого и переопределяющего методов.

Приватные члены суперкласса недоступны для подкласса напрямую, но могут быть косвенно доступна через методы – **геттеры и сеттеры**. Название происходит от английского “get” — “получать” (т.е. “метод для получения значения поля”) и set — “устанавливать” (т.е. “метод для установки значения поля”). Сеттер — это полноценный метод. А в метод, в отличие от поля, мы можем заложить необходимую логику проверки.

Конструкторы суперкласса не наследуются подклассом. **Конструктор** создает своеобразный “каркас” класса, которому каждый новый объект класса должен соответствовать.

Java не поддерживает множественное наследование, подкласс может расширять только один класс.

Мы можем создать экземпляр подкласса и затем присвоить его переменной суперкласса, это называется **upcasting** (повышающее преобразование).

Когда экземпляр суперкласса присваивается переменной подкласса, это называется **downcasting** (понижающее преобразование). Нам необходимо явно привести этот экземпляр к подклассу.

Мы можем вызывать методы суперкласса и обращаться к переменным суперкласса, используя ключевое слово **super**. Это удобно, когда у нас есть одноименная переменная/метод в подклассе, но мы хотим получить доступ к переменной/методу суперкласса. Это также используется, когда в суперклассе и подклассе определены конструкторы и нам необходимо явно вызвать конструктор суперкласса.

Мы можем использовать инструкцию **instanceof** для проверки наследования между объектами.

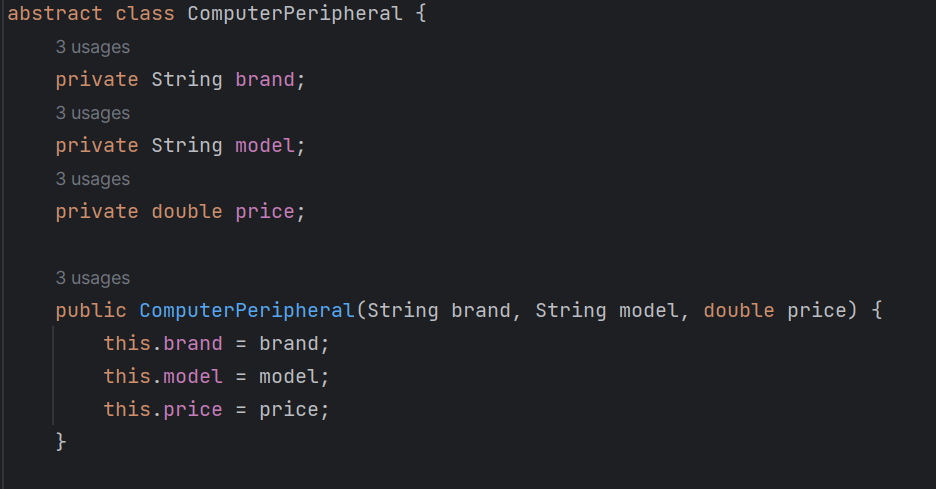
**Выполнение.**

Создайте иерархию классов. Ваша 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 работ

Вариант №15:

* Базовый класс: Компьютерная периферия
* Дочерние классы: Клавиатура, Наушники, Графический планшет

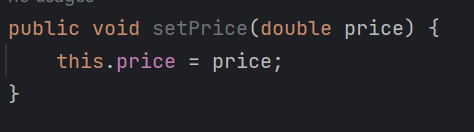
Рис.1 – Абстрактный класс Компьютерная периферия.

Сначала создаем абстрактный класс "ComputerPeripheral" (периферийные устройства компьютера).

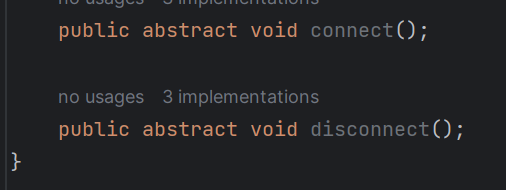
Объявляем приватные переменные "brand", "model" и "price".

Создаём конструктор класса "ComputerPeripheral" с параметрами "brand", "model" и "price".

Внутри конструктора устанавливаются значения переданных параметров в соответствующие переменные класса.

Рис.2 – Геттеры и сеттеры.

Эти методы позволяют получать значения и изменять значения переменных `brand`, `model` и `price` объекта класса `ComputerPeripheral` извне класса. Геттеры возвращают текущие значения переменных, а сеттеры устанавливают новые значения переменных на основе переданных параметров.

Рис.3 – Абстрактные методы для описания поведения периферийного устройства.

Объявлены два абстрактных метода: connect() и disconnect(). Абстрактный метод не содержит реализации внутри абстрактного класса, но должен быть реализован в каждом классе-наследнике.

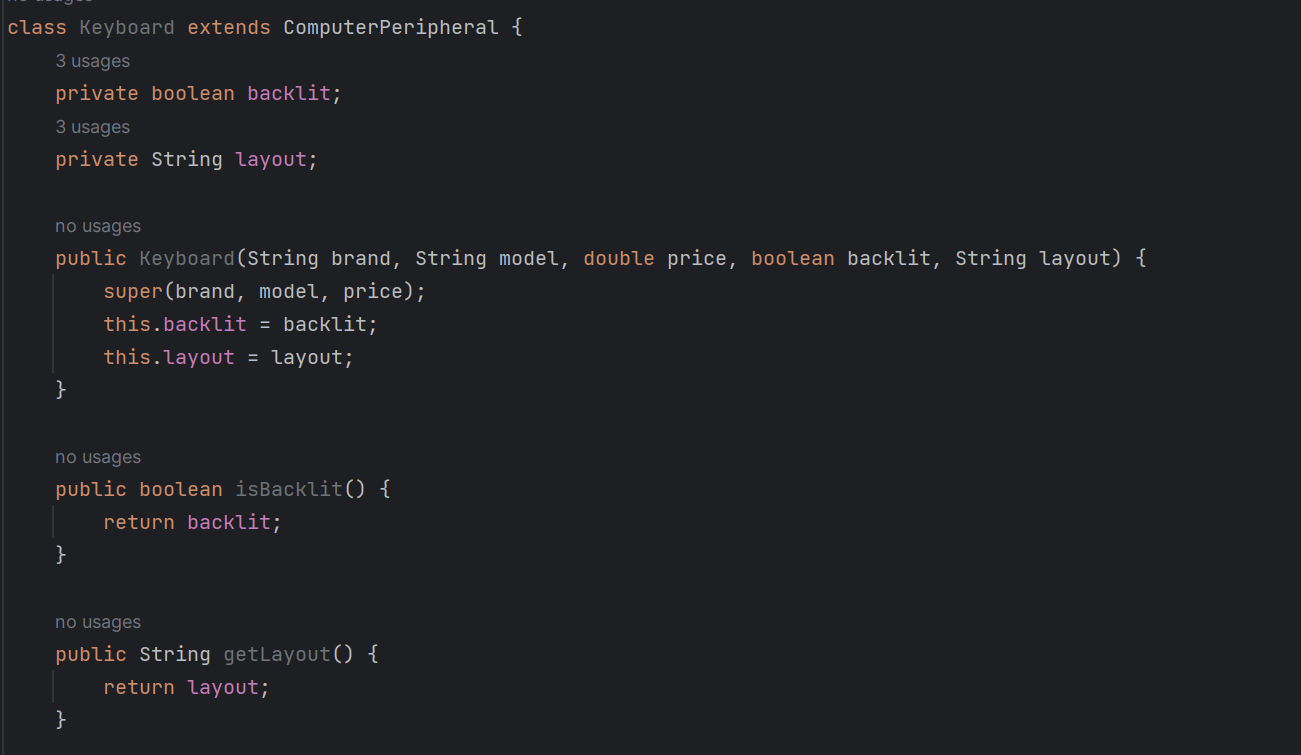
* public abstract void connect() – это абстрактный метод, который описывает процесс **подключения** периферийного устройства. Он не определяет конкретную реализацию, так как каждое периферийное устройство может иметь свою собственную логику подключения.
* public abstract void disconnect() – это абстрактный метод, который описывает процесс **отключения** периферийного устройства. Он также не определяет конкретную реализацию, так как каждое периферийное устройство может иметь свою собственную логику отключения.

Рис.4(1) – Класс клавиатура

Рис.4(2) – Класс клавиатура

Keyboard наследует все переменные и методы, определенные в классе ComputerPeripheral.

private boolean backlit: Объявление приватной переменной `backlit` типа `boolean`, которая указывает, есть ли подсветка на клавиатуре.

private String layout: Объявление приватной переменной `layout` типа `String`, которая указывает размеры клавиатуры.

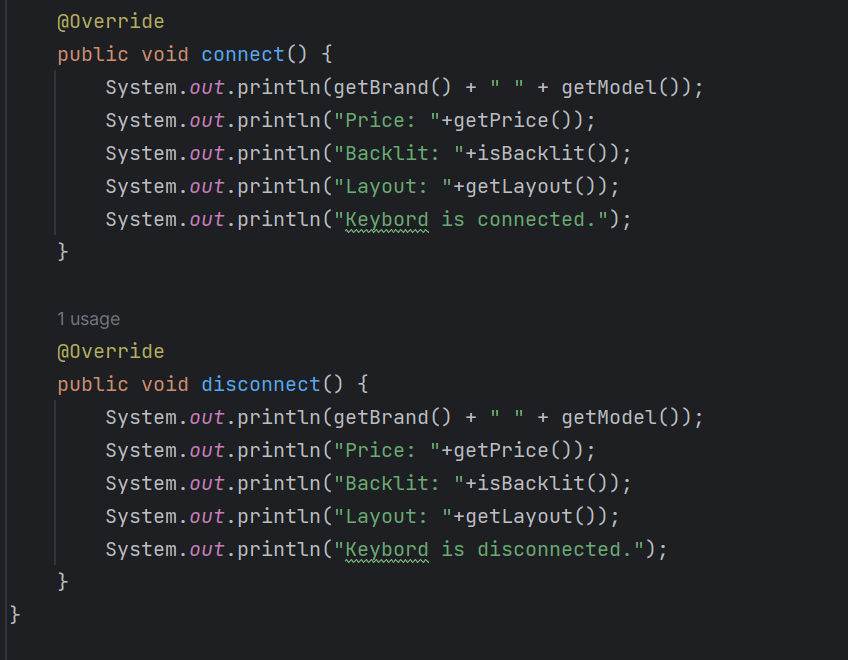
public Keyboard(String brand, String model, double price, boolean backlit, String layout): Создание конструктора класса `Keyboard` с параметрами `brand`, `model`, `price`, `backlit` и `layout`. Внутри конструктора вызывается конструктор суперкласса `ComputerPeripheral` с параметрами `brand`, `model` и `price`, чтобы установить значения этих переменных. Затем значения параметров `backlit` и `layout` устанавливаются для соответствующих переменных класса `Keyboard`.

@Override`: Аннотация, указывающая, что следующий метод переопределяет метод суперкласса.

public void connect(): Переопределение метода connect() из суперкласса ComputerPeripheral. В этом методе выводится сообщение о подключении клавиатуры с указанием марки и модели, полученных с помощью геттеров getBrand() и getModel().

public void disconnect(): Переопределение метода disconnect() из суперкласса ComputerPeripheral. В этом методе выводится сообщение о отключении клавиатуры с указанием марки и модели, полученных с помощью геттеров getBrand() и getModel().

Таким образом, класс `Keyboard` расширяет функциональность абстрактного класса `ComputerPeripheral` и добавляет свои собственные переменные и методы, специфичные для клавиатуры.

Рис.5 – connect и disconnect для класса клавиатура.

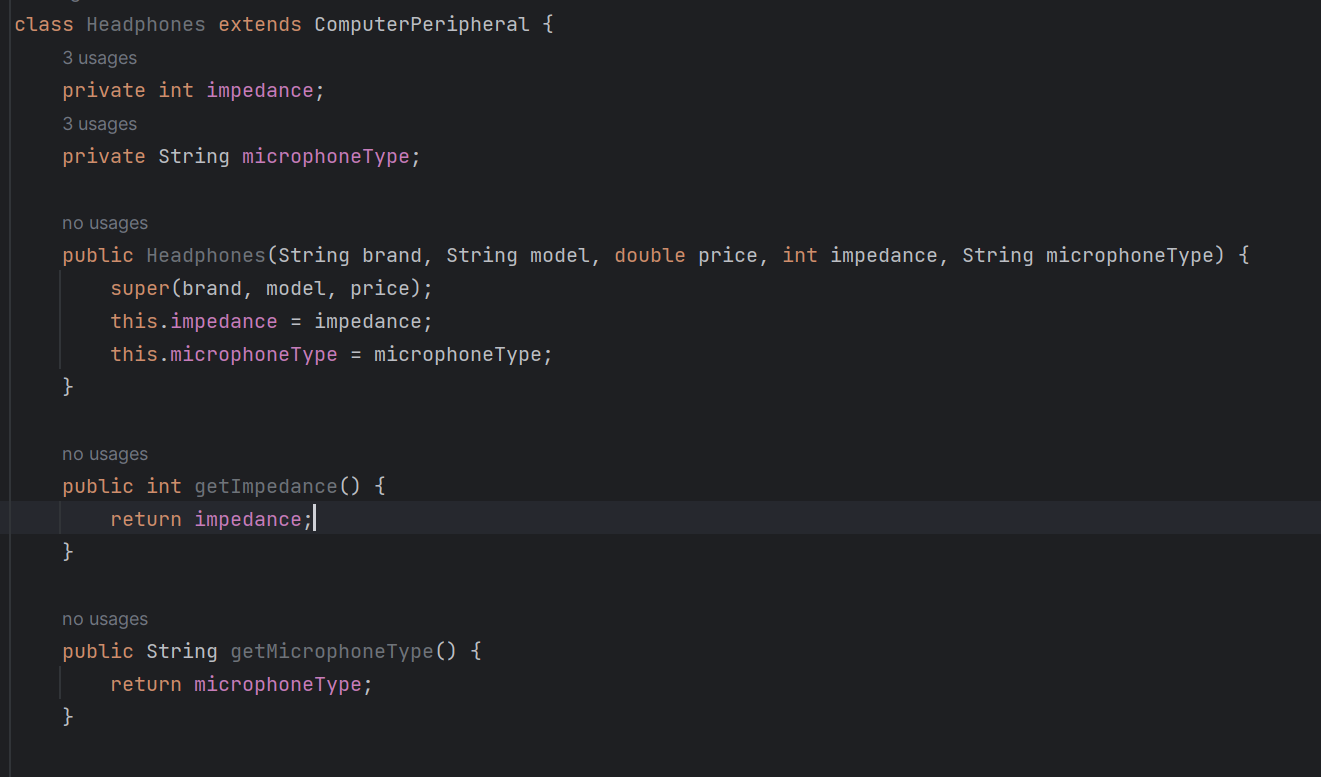
Рис.6(1) – Класс наушники

Рис.6(2) – Класс наушники.



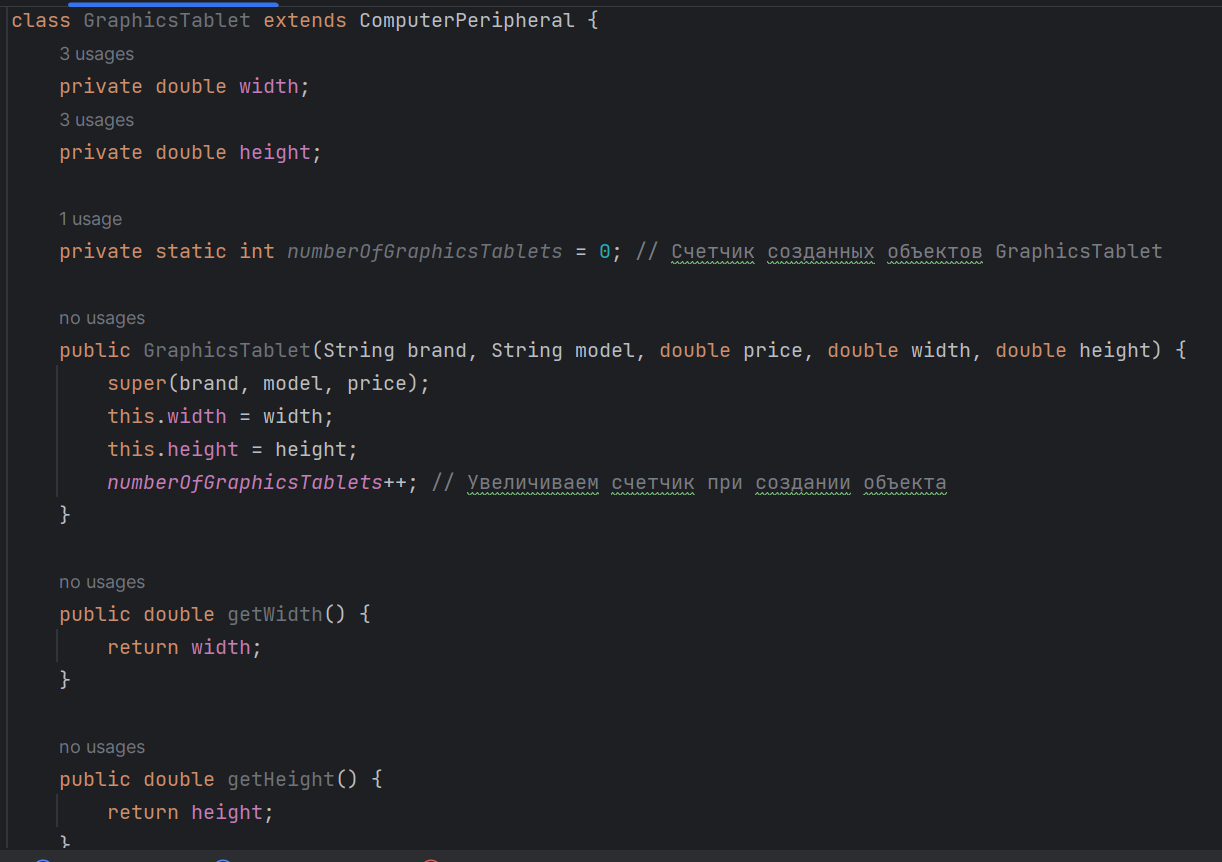
Рис.7 – connect и disconnect для класса наушники

Рис.8(1) – класс графический планшет.

Рис.8(2) – Класс графический планшет.

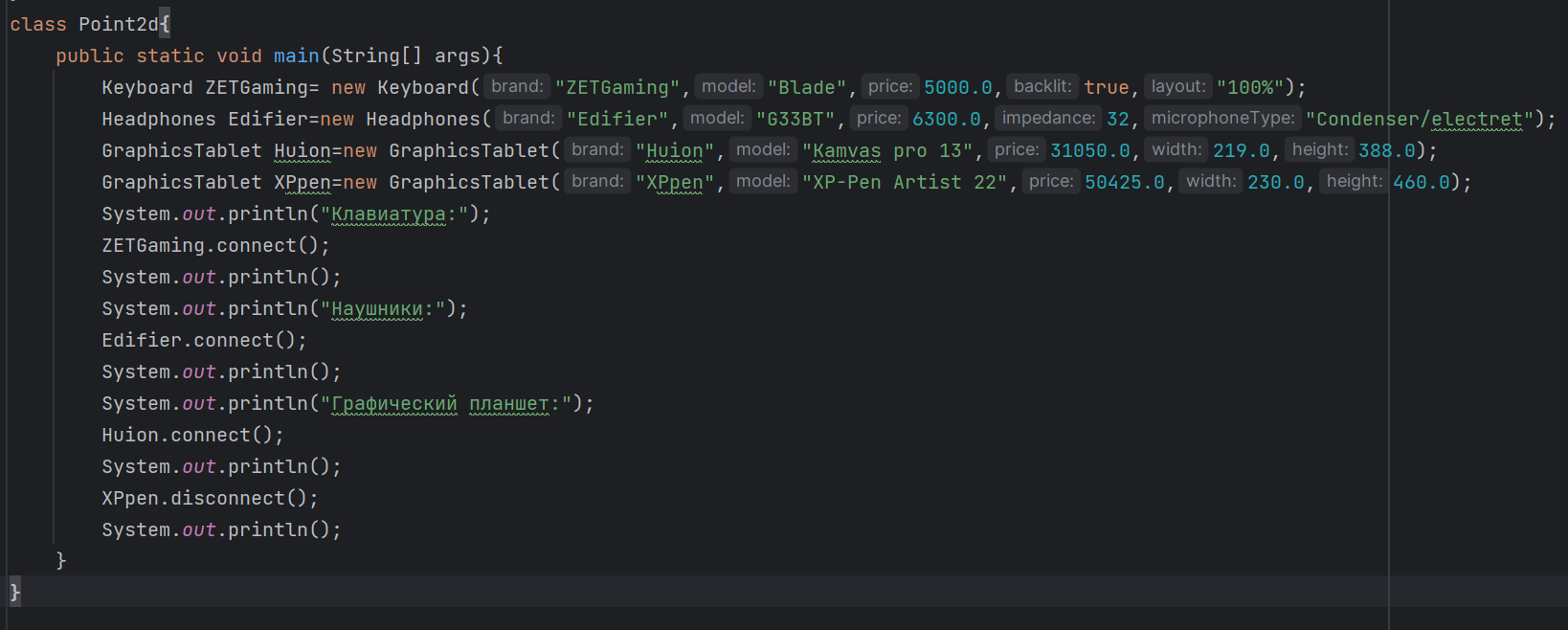
Рис.9 – connect и disconnect для класса наушники

Рис.10 – вывод.

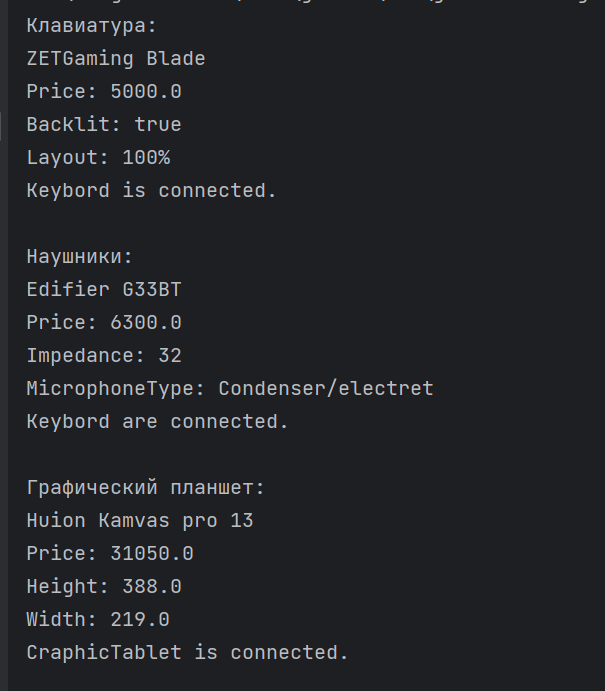


Рис.11 – вывод.

**Вывод.**

В данной лабораторной работе я использовала классы, чтобы описать, как эти объекты работают.

**<https://github.com/Ani4ka1/itip_labs.git>**