

Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Лабораторна робота №8

**Основи програмування**

*Варіант 25*

|  |  |  |
| --- | --- | --- |
| Виконав  студент групи ІТ-03: |  | Перевірила: |
|  |  |  |
|  |  |  |
|  |  |  |
| Чабан А. Є. |  | Смолій В. В. |

Київ 2020

**Тема:** Основи ООП

**Мета:** Навчитися створювати нащадків класа. Зрозуміти як використовувати overloading та overriding. Визначити різницю між mutable та immutable класами. Навчитися створювати гетери та сетери, визначити в чому їх перевага.

**Хід роботи:**

1. Повторити теоретичні відомості

2. Проаналізувати предметну область завдання свого варіанту (табл.1)

3. Розробити базовий клас (відповідно до завдання можливо абстрактний клас або інтерфейс), клас-нащадок, а також допоміжні класи та/або інтерфейси за необхідністю. Відповідно до предметної області завдання передбачити відповідні методи бізнес-логіки, а також конструктори, сетери та/або гетери, методи equals() та toString(). Продемонструвати використання:

- this;

- super;

- перевантаження (overloading) та заміщення(overriding) методів;

- перевантаження (overloading) конструкторів.

4. Відповісти на контрольні запитання

2.

\*\* immutable - після створення стан об’єктів змінити неможливо. При виконанні завдання у класах мають бути реалізовані методи, що створюють нові об’єкти, що мають новий стан (приклад: String.substring(), String.toLowerCase())

Створимо базовий клас Device:

Так як клас immutable в ньому не має бути “сетерів”, всі поля мають бути приватними/final, а класи нащадки мають також бути приватними та не мати змоги переписувати базовий клас.

**package** ua.kpi.fict.acts.it03;

**class** **Device** {

**private** String company;

**private** String matrixType;

**private** **int** price;

**public** **Device**(String company, String matrixType, **int** price) {

**this**.company = company;

**this**.matrixType = matrixType;

**this**.price = price;

}

**@Override**

**public** String **toString**() {

**return** "Device{" +

"company='" + company + '\'' +

", matrixType='" + matrixType + '\'' +

", price=" + price +

'}';

}

**public** String **getCompany**() {

**return** company;

}

**public** String **getMatrixType**() {

**return** matrixType;

}

**public** **int** **getPrice**() {

**return** price;

}

}

Створимо класи нащадки в яких теж всі поля будуть приватними та не матимуть сетерів. Класи нащадки зробимо final.

**package** ua.kpi.fict.acts.it03;

**final** **class** **Laptop** **extends** Device {

**private** String cpu;

**private** **int** ram;

**private** String gpu;

**public** **Laptop**(String company, String matrixType, **int** price, String cpu, **int** ram, String gpu) {

**super**(company, matrixType, price);

**this**.cpu = cpu;

**this**.ram = ram;

**this**.gpu = gpu;

}

**@Override**

**public** String **toString**() {

**return** "Laptop by: " + getCompany() + " with " + getMatrixType() + " matrix " + "cost: " + getPrice()+ " UAH " + "Properties: " + getCpu() +" , "+ getGpu() +" , "+ getRam() + "GB";

}

**public** String **getGpu**() {

**return** gpu;

}

**public** String **getCpu**() {

**return** cpu;

}

**public** **int** **getRam**() {

**return** ram;

}

}

------------------------

**package** ua.kpi.fict.acts.it03;

**final** **class** **Tablet** **extends** Device

{

**private** **boolean** isWebcam;

**private** **int** ram;

**private** String cpu;

**private** String gpu;

**public** **boolean** **isWebcam**() {

**return** isWebcam;

}

**@Override**

**public** String **toString**() {

**return** "Tablet by: " + getCompany() + " with " + getMatrixType() + " matrix " + "cost: " + getPrice()+ " UAH " + "Properties: " + getCpu() +" , "+ getGpu() +" , "+ getRam() + "GB " + "Webcam: " + isWebcam();

}

**public** **int** **getRam**() {

**return** ram;

}

**public** String **getCpu**() {

**return** cpu;

}

**public** String **getGpu**() {

**return** gpu;

}

**public** **Tablet**(String company, String matrixType, **int** price, **boolean** isWebcam, **int** ram, String cpu, String gpu) {

**super**(company, matrixType, price);

**this**.isWebcam = isWebcam;

**this**.ram = ram;

**this**.cpu = cpu;

**this**.gpu = gpu;

}

}

-----------------------------------

**package** ua.kpi.fict.acts.it03;

**final** **class** **PC** **extends** Device

{

**private** **int** ram;

**private** String cpu;

**private** String gpu;

**private** **int** usbPorts;

**public** **PC**(String company, String matrixType, **int** price, **int** ram, String cpu, String gpu, **int** usbPorts) {

**super**(company, matrixType, price);

**this**.ram = ram;

**this**.cpu = cpu;

**this**.gpu = gpu;

**this**.usbPorts = usbPorts;

}

**public** **int** **getRam**() {

**return** ram;

}

**public** String **getCpu**() {

**return** cpu;

}

**@Override**

**public** String **toString**() {

**return** "PC by: " + getCompany() + " with " + getMatrixType() + " matrix " + "cost: " + getPrice()+ " UAH " + "Properties: " + getCpu() +" , "+ getGpu() +" , "+ getRam() + "GB " + "USB: "+ getUsbPorts();

}

**public** String **getGpu**() {

**return** gpu;

}

**public** **int** **getUsbPorts**() {

**return** usbPorts;

}

}

Далі весь USER Interface реалізуємо в класі і методі Main:

**package** ua.kpi.fict.acts.it03;

**public** **class** **Main** {

**public** **static** **void** **main**(String[] args) {

Laptop lap1 = **new** Laptop("Lenovo", "IPS", **30000**, "INTEL Core i5", **16**, "GTX 1650TI");

System.out.println(lap1.toString());

System.out.println("//////");

Tablet tab1 = **new** Tablet("Apple", "OLED", **45000**, **true**, **6**, "Apple A12Z", "M12");

System.out.println(tab1.toString());

System.out.println("//////");

PC pc1 = **new** PC("HyperPC", "IPS", **68999**, **32**, "AMD Ryzen 9", "RTX 3090", **6**);

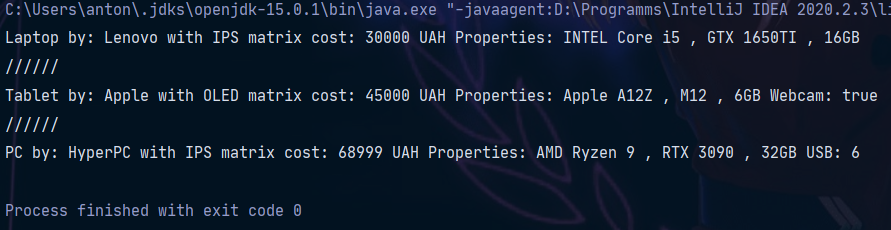
System.out.println(pc1.toString());

}

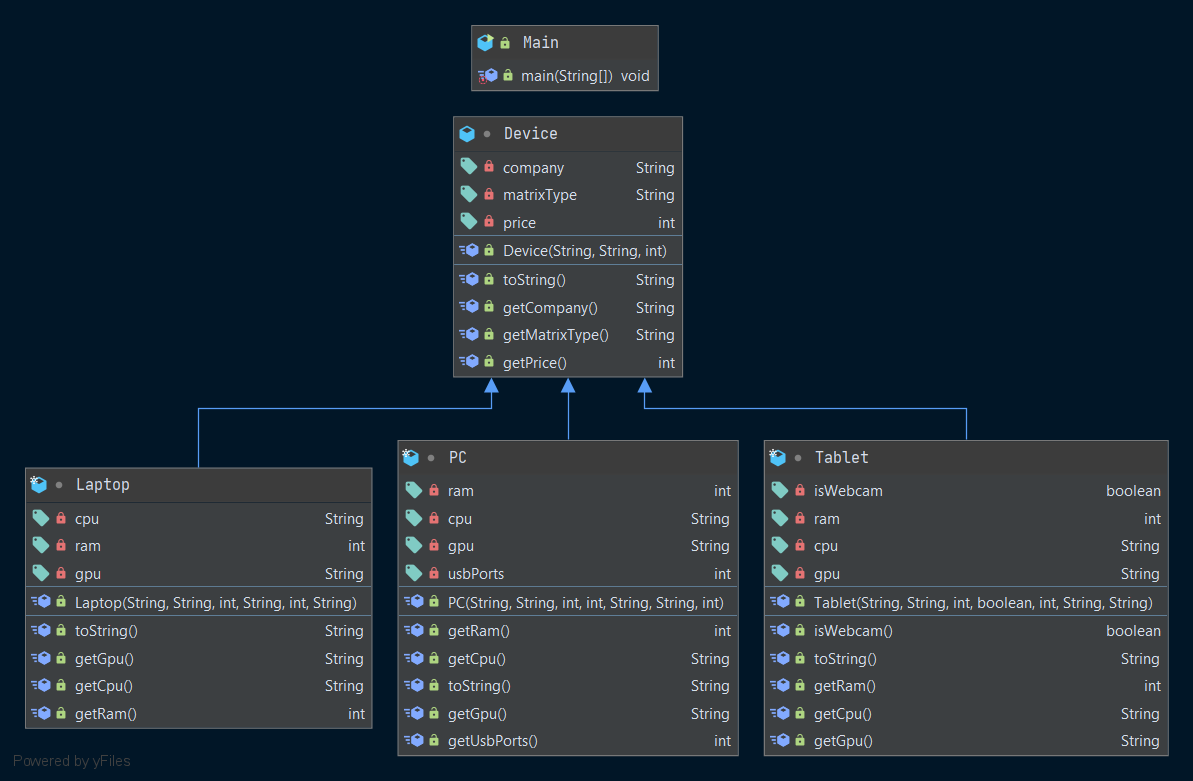
}

[Для зручності читання коду — код завантажено на GitHub.](https://github.com/antonchaban/oplab8)

Результат виконання коду:



UML діаграмма класів:



**Висновок:**

Під час виконання данної роботи я навчився створювати базові класи та класи нащадки, зрозумів як використовувати overloading та overriding. Освоїв різницю між mutable та immutable класами.

**Контрольні питання:**

1)Конструктор – блок інструкцій, що виконуються при створенні об’єкту. Використовується в основному для задання стану об’єкту.

2)Перевантаження – визначення одного методу, що при різних аргументах може працювати по-різному. Перевизначення – задання методу в класі нащадку по-новому, можливо, для доповнення функціоналу.

Override та override можна використовувати одночасно.

3)Immutable об’єкти - це об’єкти в яких неможна змінювати стан. Замість зміни об’єкту методи класу повертають новий об’єкт. Зазвичай immutable класи використовують при багато-поточному програмуванні, оскільки це безпечніше, оскільки їм не потрібна синхронізація, так як вони не змінюються.

4) Вони дають программі додаткову безпеку, так як користувач не може змінити поля на будьякі значення (за правильного прописання гетерів та сетерів). За рахунок приховання полів користувач не взаємодіє на пряму з бізнес логікою.

5) Коли викликається конструктор нащадка, в ньому зразу викликається конструктор суперкласу, після виконання якого продовжується виконання конструктора нащадка)

6)

Статичний блок ініціалізації викликається першим і єдиний раз (при створенні першого об’єкта класу) для ініціалізації статичних полів класу.

Нестатичний блок ініціалізації викликаються при кожному новому створенні об’єкта.