11. Въведение в ООП

Програмиране на Java

Съдържание

Процедурно програмиране

- Преди обектно-ориентираното програмиране стандартната техника за програмиране е процедурната.
- Нарича се така защото фокуса е върху процедурите (или задачите), които заедно решават конкретния проблем.
- При този подход се мисли относно това какво трябва да се постигне и как дейностите да се отделят в отделни стъпки, които да бъдат реализирани (имплементирани) като процедури.

Обектно-ориентирано програмиране

- ООП предразполага да се мисли в посока какво трябва да представя програмата.
- Обикновено се идентифицират “неща” от заобикалящия ни свят, които искаме да бъдат моделирани от програмата.
- Това могат да бъдат както физически, така и концептуални “неща”.
- След това се идентифицират основните свойства/атрибути на “нещата”.
- Определя се и какво може да правят тези “неща” - т.е. тяхното поведение, или какво други “неща” могат да им направят.
- За всяко “нещо” идентифицираните свойства/атрибути и определеното му поведение се групира в единична структура, която се нарича ***обект***.
- Като обобщение: при създаване на ОО програма дефинираме обекти, създаваме ги, и ги караме да взаимодействат по между си.

Обект

Един **обект** се състои от:

- набор от свързани данни, които определят текущото ***състояние*** на обекта;
- набор от ***поведения***

Състояние и поведение на обект

Състоянието на обекта се отнася към характеристиките, които определят обекта в момента.

Например, програма, която работи със списък от студенти ще разглежда студентите като обекти, където състоянието на обекта за всеки студент се състои от името на студента, факултетният му номер и средният му успех до момента.

Поведенията на обекта се отнасят към дейностите, които са асоциирани с обекта.

В горния пример, ще имаме нужда от поведение, което да променя средния успех на студента. Този вид поведение кореспондира с поведението от реалния свят - вдигане на успеха или сваляне на успеха.

В Java поведенията се имплементират като методи. Например, метода за промяна на средния успех може да се имплементира като метод с име adjustAvgGrade.

Примери за обекти

Няколко примера за добри кандидати за обекти в ОО програма:

- Физически обекти:
  - **коли** в софтуер за симулация на трафика
  - **самолети** в система за въздушен контрол
  - **електронни компоненти** в програма за проектиране на електронни схеми
- Човека като обект:
  - **служители**
  - **клиенти**
  - **студенти**
- Математически обекти:
  - **точки** в координатна система
  - **сложни числа**
  - **време**

Пример

В примера за симулация на трафика колите са основния кандидат за обект при реализация на системата.

- Какви данни е необходимо да се запазват за всеки обект-кола (тези данни трябва да се запазват като част от състоянието на обекта-кола)?
  - текущата позиция на колата
  - текущата скорост
- Какви поведения са асоциирани с колата?
  - колата трябва да може да потегля;
  - да спира;
  - да намалява;

=> методи start, stop, slowDown

Поведението на обект може да промени неговото състояние. Например, метода start променя позицията и скоростта на колата.

Капсулиране

Обектите предоставят механизъм за *капсулация*. Най-общо казано, нещо е капсулирано когато е обвито в защитна обвивка.

Когато се прилага към обекти, капсулирането означава, че данните на даден обект са защитени, като са "скрити" вътре в обекта.

При скрити данни как останалата част от програмата може да получи достъп до данните на обекта? (Достъпът до данните на даден обект означава или четене на данните, или тяхната промяна.)

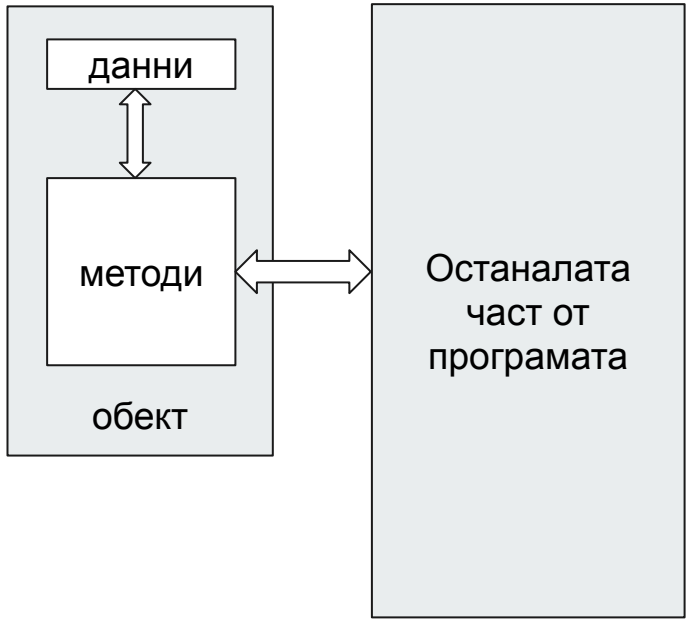
Останалата част от програмата не може да има пряк достъп до данните на обекта, но може да получи достъп до тях с помощта на методите на обекта. Ако методите на обекта са добре написани, те гарантират, че достъпът до данните се осъществява по подходящ начин.

Капсулиране

В примера със системата, която работи с данни на студенти, методът adjustAvgGrade гарантира, че средният успех на студента се модифицира правилно - т.е. предотвратява модификации, които ще променят средния успех на отрицателна стойност.

Капсулиране

За да се стигне до данните на обекта трябва да се използват методите на обекта.



Ползи от ООП

Защо е целият “шум” относно ООП? Защо ООП се предпочита пред процедурното програмиране за повечето съвременни програми?

Ето някой ползи от ООП:

- *ООП програмите имат по-естествена организация.*
  Тъй като на хората са склонни да мислят за проблемите от реалния свят в термините на реални обекти, за тях е по-лесно да разберат програма, която е организирана около обекти.

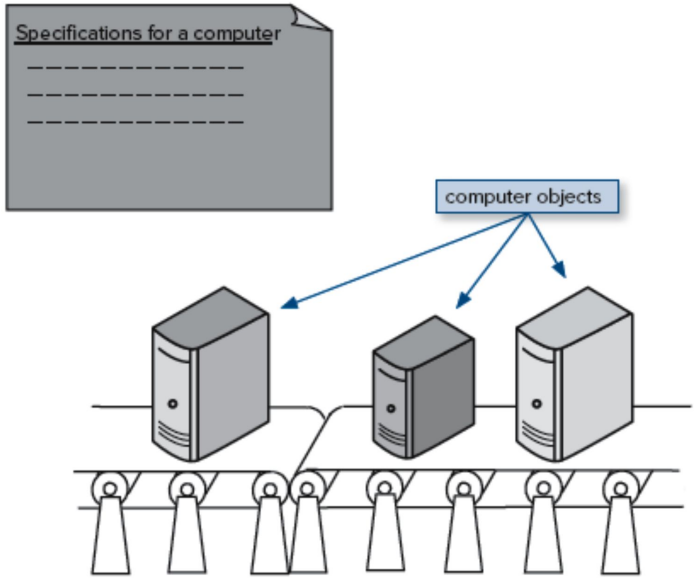
- *ООП помага за по-лесното разработване и поддържане на големи програми.*
  Въпреки че преминаването към ООП програмиране обикновено усложнява една малка програма, то естествено разделя нещата, така че програмата да расте грациозно и да не се превръща в гигантска бъркотия. Тъй като обектите осигуряват капсулиране, бъговете (грешките) и поправките на грешки обикновено са локализирани.

Класове

Най-общо казано, ***един клас представлява описание на всички обекти, които дефинира***. Като такъв той е абстракция - концепция, отделена от всички конкретни екземпляри.

Обърнете внимание на трите компютъра върху конвейерна лента в производствено предприятие. Трите компютъра представляват обекти. Документът със спецификациите, който се намира над компютрите, е проект, който описва компютрите: той изброява компонентите на компютрите и описва техните характеристики. Документът със спецификациите на компютрите представлява **клас**.

Всеки ***обект*** е ***инстанция*** на своя ***клас***. За практически цели "обект" и "инстанция" са синоними.



Класове

Един клас може да има както произволен брой обекти, свързани с него, така и никакви. Това би трябвало да има смисъл, ако се замислим за примера с производството на компютри. Възможно е да има проект на компютър, но все още да няма нито един компютър, произведен по този план.

Класове

Казахме, че класът е описание на набор от обекти. Описанието се състои от:

- списък с **полета** (атрибути)
- списък с **методи**

В термините на програмирането "поле" е място за съхранение, което съхранява определен вид информация. В Java полето по подразбиране е променлива, но ако декларацията му има final модификатор, то става именувана константа. Така че **полето** в Java може да бъде или **променлива**, или **константа**.

Класове

Класовете могат да дефинират:

- два вида атрибути - статични (static) атрибути и атрибути на инстанции.
- два вида методи - статични (static) методи и методи на инстанции.

Примери:

- main метода е статичен

От тук нататък ще се опитваме да избягваме използването на статични атрибути и методи. Обяснението на смисъла на статичните атрибути и методи ще бъде направено по-късно.

Класове

Атрибутите на инстанция и константите на инстанция на класа определят типа данни, които обектът може да съхранява. Например, ако има клас за обекти-компютър и класът Computer съдържа атрибут на инстанция hardDiskSize, тогава всеки обект за компютър съхранява стойност за размера на твърдия му диск.

Методите на инстанцията на класа определят поведението, което обектът може да прояви. Например, ако имате клас Computer - за обекти за компютри, който съдържа метод на инстанция printSpecifications, тогава всеки обект за компютър може да отпечата справка за спецификацията на съответния компютър (справката показва размера на твърдия диск на компютъра, скоростта на процесора, цената и т.н.).

Класове

Терминът "инстанция" в "атрибут на инстанция" и "метод на инстанция" набляга на факта, че атрибутите и методите на инстанция са свързани с конкретна инстанция на обекта. Например всеки обект на студент би имал своя собствена стойност за атрибута на инстанция "среден успех", до която би имал достъп чрез своя метод на инстанция adjustAvgGrade.

Това контрастира със статичните методи. Статичните методи са свързани с целия клас. Например класът Math съдържа статичния метод round, който не е свързан с конкретна инстанция на класа Math.

Първи ООП клас

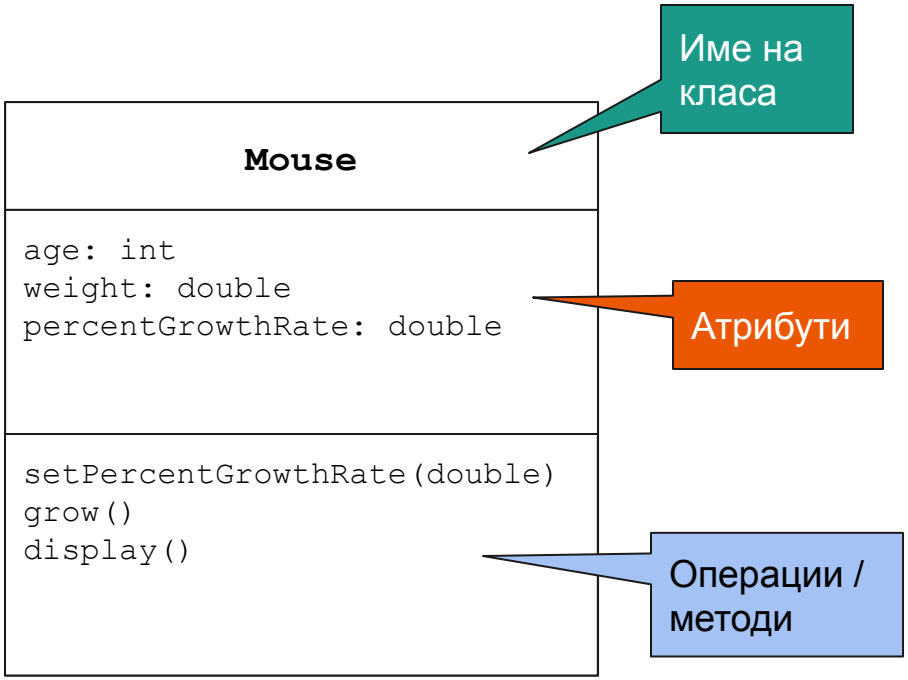
Ще реализираме завършена ОО програма. Програмата ще съдържа клас Mouse, който ще моделира растежа на две истински мишки.

Както е обичайно при ОО програмите, започваме процеса като описваме решението с диаграма на класовете в UML. Тази диаграма е техника за описание на класове и връзките между тях. Тя е широко приета в софтуерната индустрия като стандарт за моделиране на ООП проекти.

UML Class Diagram

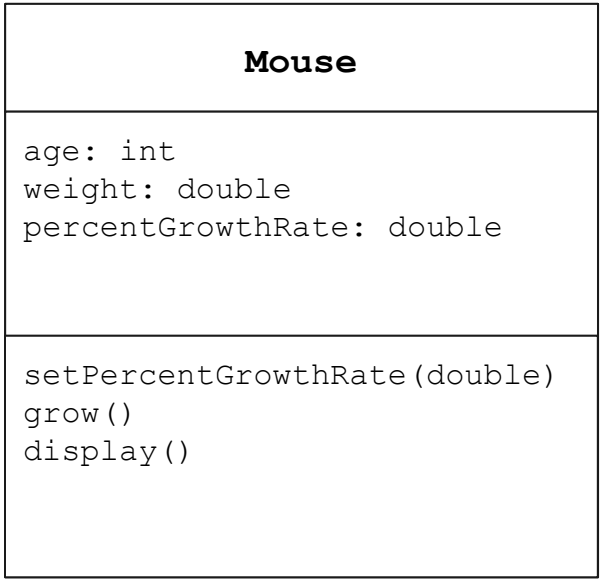
Диаграмата на класовете в UML е разделена на три части - *име на класа* в горната част, *атрибути* в средата и *операции* в долната част. При програмите на Java атрибутите се равняват на променливи, а операциите - на методи.

Променливите и методите на класа се наричат общо членове на класа.



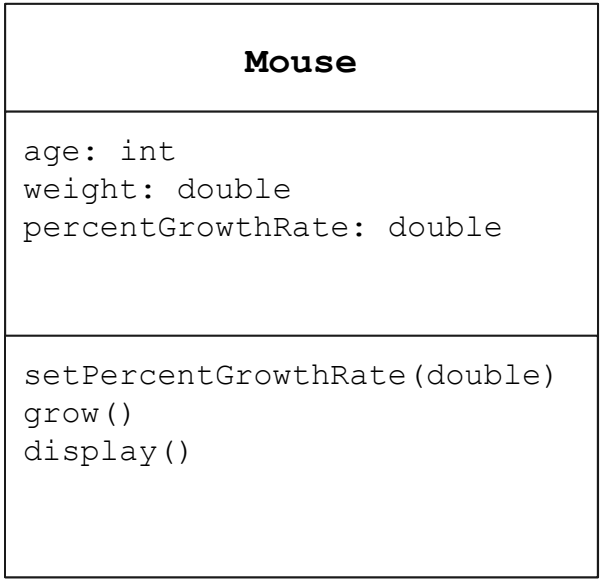
UML Class Diagram

Класът **`Mouse`** има три променливи на инстанция - възраст (`age`), тегло (`weight`) и процент на нарастване (`percentGrowthRate`). Атрибута за възраст показва възрастта на обектът Mouse в дни. Атрибута за тегло показва теглото на обекта Mouse в грамове. Атрибута `percentGrowthRate`е процентът от текущото му тегло, който се добавя към теглото му всеки ден. Ако `percentGrowthRate`е 10% и текущото тегло на мишката е 10 грама, тогава мишката ще наддаде 1 грам до следващия ден.



UML Class Diagram

Класът **`Mouse`** има три метода на инстанция - `setPercentGrowthRate`, `grow` и `display`. Методът `setPercentGrowthRate` присвоява определена стойност на променливата на инстанция `percentGrowthRate`. Методът `grow` (растеж) симулира еднодневно наддаване на тегло на мишка. Методът `display` извежда възрастта и теглото на мишката.



Mouse Class Source Code

Променливите на инстанцията трябва да бъдат декларирани извън всички методи, а за да направите кода си по-ясен, трябва да ги декларирате всички в началото на дефиницията на класа.

Инициализираме възрастта на 0, защото новородените мишки са на нула дни. Инициализираме теглото на 1, защото новородените мишки тежат приблизително 1 грам.

```
public class Mouse {
    private int age = 0;
    private double weight = 1.0;

    private double percentGrowthRate;

    public void setPercentGrowthRate(double pgr) {
        percentGrowthRate = pgr;
    }

    public void grow() {
        weight += 0.01 * percentGrowthRate * weight;

        age++;
    }

    public void display() {
        System.out.printf("Age = %d, weight = %.3f
%n",
                          age , weight);
    }
}
```

Mouse Class

Основната разлика между декларациите на променливи на инстанция и декларациите на локални променливи, е модификаторът за достъп - `private`. Ако декларирате даден член като `private`, то той може да бъде достъпен само от класа на члена, а не от "външния свят" (т.е. от код, който е извън класа, в който се намира членът). Променливите на инстанциите почти винаги се декларират с модификатора за достъп `private`, защото почти винаги искаме данните на обекта да бъдат скрити. Превръщането на променлива на инстанция в `private` ви дава възможност да контролирате начина, по който нейната стойност може да бъде променена. Например, можете да гарантирате, че теглото никога няма да бъде отрицателно.

Ограничаването на достъпа до данни е това, което представлява капсулирането, и е един от крайъгълните камъни на ООП.

Mouse Class

В допълнение към модификатора за достъп `private` има и модификатор за публичен достъп - `public`. Ако декларирате даден член като публичен, тогава той може да бъде достъпен отвсякъде (от класа на члена, а също и извън класа на члена).

Трябва да декларирате даден метод като публичен, когато искате той да бъде портал, чрез който външният свят да има достъп до данните на вашите обекти. Когато искате даден метод да ви помогне да изпълните само локална задача, трябва да го обявите за `private`

Driver Class

"Драйвер" е общоприет компютърен термин, който се отнася за част от софтуера, който управлява или "задвижва" нещо друго. Например драйвер на принтер е програма, която отговаря за работата на принтера. По същия начин, класът драйвер е клас, който отговаря за управлението на друг клас.

```
import java.util.Scanner;

public class MouseDriver {
   public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);

        double growthRate;

        Mouse jerry = new Mouse();
        Mouse mikey = new Mouse();

        System.out.print("Въведете % на нарастване на ден: ");
        growthRate = keyboard.nextDouble();

        jerry.setPercentGrowthRate(growthRate);
        mikey.setPercentGrowthRate(growthRate);

        for (int i = 0; i < 10; i++) {
            jerry.grow();
            mikey.grow();
        }

        jerry.grow();

        jerry.display();
        mikey.display();
    }
}
```

Създаване на два обекта от тип Mouse

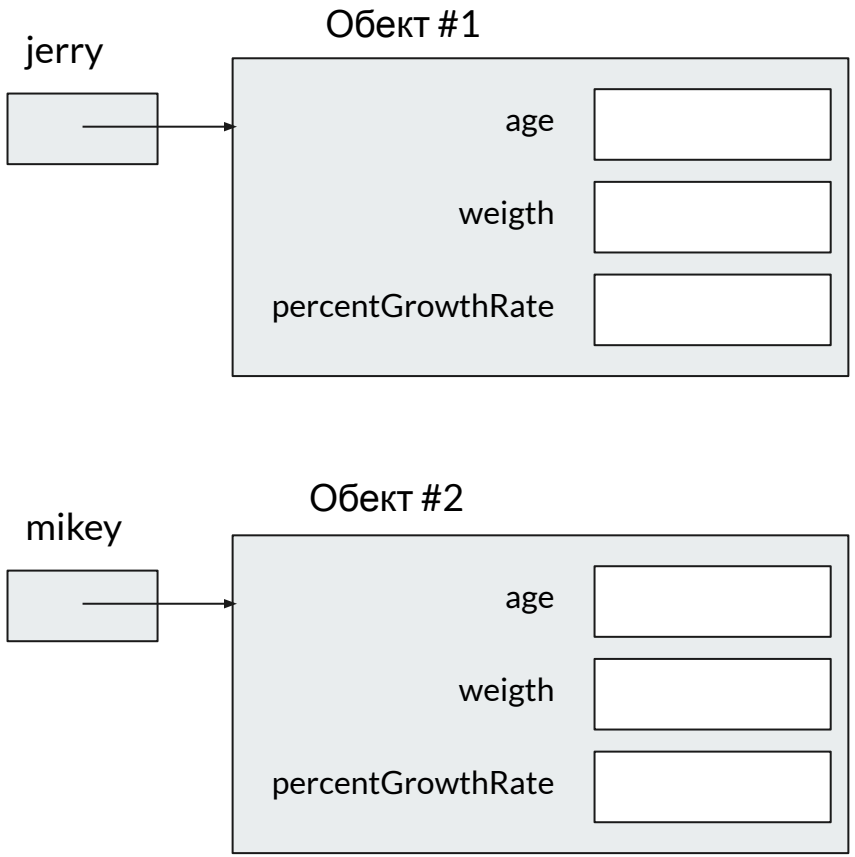
Манипулиране на обектите от тип Mouse

Driver Class

Обикновено класът на драйвера се състои изцяло от един main метод и нищо друго. Класът на драйвера, с неговия main метод, е началната точка на програмата. Той се използва управлявания клас, за да създава обекти и да ги манипулира.

Референтни променливи

В класа MouseDriver създаваме обекти Mouse и се позоваваме на тези обекти Mouse с помощта на jerry и mikey, където jerry и mikey са референтни променливи. Стойността, съдържаща се в референтна променлива, е "препратка" към обект (оттук и името референтна променлива). По-точно, референтната променлива съдържа адреса на мястото, където обектът се съхранява в паметта.



Още теми

- Деклариране и инициализиране на референтна променлива.
- Static vs instance attributes and methods
- Извикване на метод
- Организиране на кода в пакети
- Модификатори за достъп (public, private, package)

Регистриране на присъствие

https://t.ly/PWMB

Отговор: uml

Регистриране на присъствие

https://t.ly/BnyJ

Отговор: public