От записи към класове

(преговор с разширение)

Трифон Трифонов

Обектно-ориентирано програмиране,
спец. Компютърни науки, 1 поток,
2018/19 г.

20 февруари 2019 г.

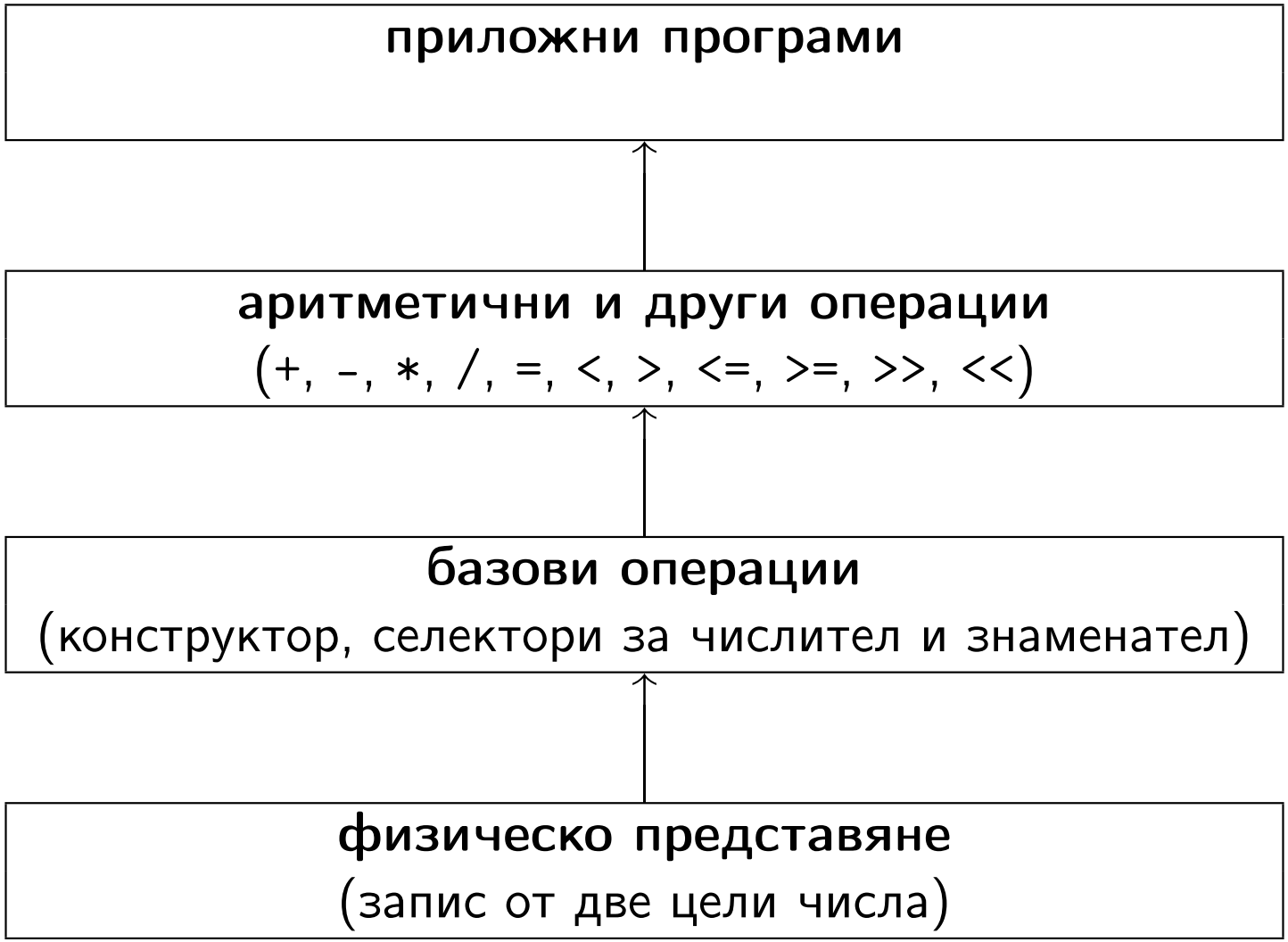
Абстракция със структури от данни

- Създаване на нови типове данни чрез групиране на съществуващи типове данни в запис (`struct`)
- Полетата на записа ще наричаме член-данни
- Създаваме операции, които работят върху член-данните
  - операциите "знаят" как е представен обекта
- Външните функции, които използват новия тип, работят с него чрез операциите
  - външните функции "не знаят" как е представен обекта

Пример: тип "рационално число"

- **Логическо описание:** обикновена дроб
- **Физическо представяне:** запис с числител и знаменател
- **Базови операции:**
  - конструиране на рационално число
  - получаване на числител
  - получаване на знаменател
- **Аритметични операции:**
  - събиране, изваждане
  - умножение, деление
  - сравнение
- **Други операции:**
  - въвеждане
  - извеждане
  - преобразуване до число с плаваща запетая
- **Приложни програми**

Нива на абстракция



Ниво 0: представяне на рационално число

```
struct Rational {
  int numer, denom;
};
```

- `numer` представя числителя на рационалното число
- `denom` представя знаменателя на рационалното число
- `numer` и `denom` са **член-данни** на новия тип данни `Rational`

Ниво 1: Конструктори

- Конструкторите са функции, които инициализират член-данните
- Конструкторите са "вътрешни функции" (**член-функции**, **методи**)
- Конструкторите имат същото име като типа данни, който конструират
- Конструкторите винаги връщат типа данни, който конструират
  - затова тип на резултата не се указва изрично
- Конструктор по подразбиране (`Rational()`)
  - Използва се при дефиниране без изрична инициализация
  - `Rational r;`
  - `Rational r = Rational();`

```
Rational() {
  numer = 0;
  denom = 1;
}
```

Ниво 1: Конструктори

- Конструктор с параметри (`Rational(n, d)`)
  - Приема по един параметър за всяка член-данна
  - `Rational r(1, 2);`
  - `Rational r = Rational(1, 2);`

```
Rational(int n, int d) {
  numer = n;
  denom = d;
}
```

Ниво 1: Селектори и мутатори

- Селекторите са **член-функции**, които позволяват преглед на член-данните
- Селектори за достъп
  - `int getNumerator()   { return numer; }`
  - `int getDenominator() { return denom; }`
- Селектори за извеждане/конвертиране
  - `void print();`
  - `double toDouble();`
- Мутаторите са **член-функции**, които позволяват промяна на член-данните
- `void read();`

Ниво 2: Аритметични операции

Искаме да моделираме математическите операции над рационални числа:

- `Rational add(Rational p, Rational q);`
- `Rational subtract(Rational p, Rational q);`
- `Rational multiply(Rational p, Rational q);`
- `Rational divide(Rational p, Rational q);`

Ниво 3: Приложни програми

Задача. Да се намери рационално число, което приближава e^k.
Решение. Ще напишем функция, която пресмята сумата:

$$\sum_{i=0}^{n} \frac{k^i}{i!}$$

Проблем. Числителите и знаменателите стават много големи!
Решение. Да използваме `long`.
Проблем. Само отлага препълването...
Решение. Да съкращаваме дробите.
Проблем. На кое ниво да извършим съкращението?
Решение. На възможно най-ниското: ниво 0.

Предимства на абстракцията

- Изолиране на промените
  - промените по едно ниво налагат промени само в следващото ниво
- Разпространяване на промените
  - подобренията в едно ниво се отразяват положително на всички по-горни нива
- Ограничаване на знанието
  - за реализацията на елемент на някое от нивата е нужна само информация за елементите на долното ниво
  - не е нужно да познаваме в подробности как са реализирани операциите и алгоритмите за работа с член-данните
  - работата по отделните нива може да се извършва паралелно и независимо

Капсулация

Принцип на капсулацията:
Разделя се (абстрахира се) описанието на типа данни от конкретната му реализация.

- Описание (**интерфейс**): име на типа, сигнатури на функции и методи
- Представяне (**имплементация**): полета на типа, тела на функции и методи
- Абстракцията със структури от данни се възползва от капсулацията
  - представянето на ниво $n + 1$ работи с описанието на ниво n
  - представянето на ниво n не зависи от представянето на нивата $< n$
- Предимства на капсулацията
  - можем да подменим представянето без да се отрази на описанието
  - външните функции зависят само от описанието
  - промени по представянето не налагат промени по външните функции
  - описанието обикновено е по-просто от представянето, изолира се сложността