От записи към класове
(преговор с разширение)

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: рационални числа

- Логическо описание: обикновена дроб
- Физическо представяне: запис с числител и знаменател
- Базови операции: създаване на рационални числа, намиране на числител и на знаменател
- Аритметични операции: събиране, изваждане, умножение, деление
- Други операции: въвеждане и извеждане

Нива на абстракция

Представяне на рационално число

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

- numer и denom за член-данни на новия тип данни Rational

Заглавни (header) файлове

- В заглавните файлове се пишат
  - дефинициите на записи и класове
  - декларации на функции
  - декларации на методи
- В заглавните файлове обикновено не се пишат
  - дефиниции на променливи
  - дефиниции на функции
  - дефиниции на методи
- Защо?
  - Абстракция: разделя се интерфейса от реализацията

Конструктори

- Конструкторите инициализират член-данните
- Конструктор по подразбиране
  - Rational r = Rational();
  - Rational r;

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

Конструктори

- Конструктори

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

Селектори

- int getNumerator() const;
- int getDenominator() const;
- void print() const;

Мутатори

- void read();

Аритметични операции

- Rational add(Rational const&, Rational const&);
- Rational subtract(Rational const&, Rational const&);
- Rational multiply(Rational const&, Rational const&);
- Rational divide(Rational const&, Rational const&);

Предимства на абстракцията

- Ограничаваме „знанието“ за представянето на данните
  - не е нужно да познаваме в подробности как са реализирани операциите и алгоритмите за работа с член-данните
  - при промяна на член-данните, променяме малък брой методи
- Можем да подменим изцяло представянето
  - стига да запазим примитивните операции (конструктор, селектори, мутатори)
  - останалата част от програмата ще продължава да работи
- Подобренията се разпространяват автоматично
  - ако представянето е по-ефективно, цялата програма става по-ефективна

Капсулиране на данните

- спецификатори за достъп
  - private - само методите на класа могат да достъпват
  - public - външни програми могат да достъпват
- препоръчва се всички член-данни да са капсулирани чрез private
- смислено ли е методи да са private?
  - да, когато са за вътрешно ползване

От структури към класове

- **struct** се заменя с **class**
- какво се променя?
  - почти нищо!
  - нивото на достъп по подразбиране в записите е **public**
  - а в класовете е **private**