КЛАСОВЕ. ОБЕКТИ.

гл.ас. д-р. Нора Ангелова

КЛАСОВЕ

В езика C++ има стандартен набор от типове данни като int, double, float, char, string и др. Този набор може да бъде разширен чрез дефинирането на класове.

Дефинирането на клас въвежда нов тип, който може да бъде интегриран в езика.

Класовете са в основата на ООП.

Класове

- Класовете са подобни на записите (структурите).
- Класовете имат допълнителни ограничения по отношение на правата за достъп.

Напомняне (абстракция)

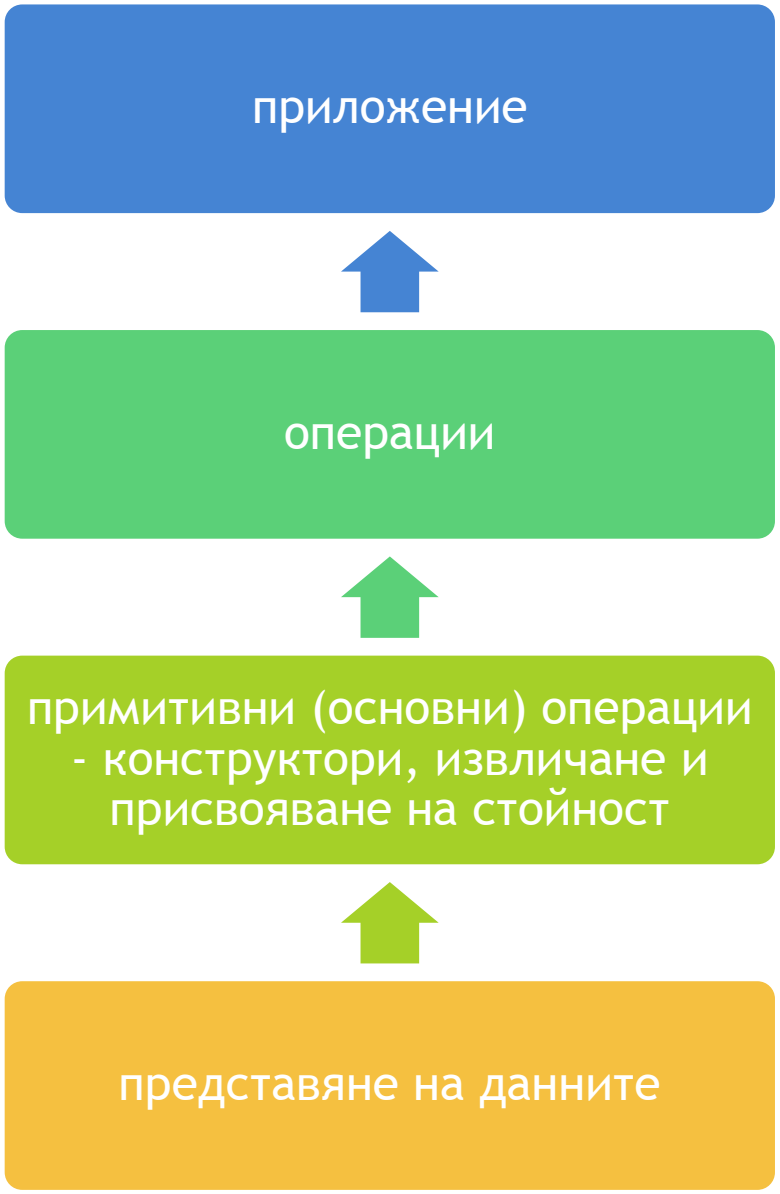
Идея: методите за използването на данните се разделят от тяхното представяне.

1. Всяка програма се проектира така, че да работи с „абстрактни данни“ – данни с неясно представяне.
2. Представянето на данните се конкретизира с помощта на множество функции – конструктури, селектори (гетъри), мутатори (сетъри), предикати.

Напомняне (абстрактен тип данни)

- **Абстрактен тип данни** – тип данни, за който се изисква скриване на реализацията на типа. Неговото „поведение“ се дефинира от множество от данни и множество от операции.

КЛАСОВЕ


приложение
операции
примитивни (основни) операции - конструктори, извличане и присвояване на стойност
представяне на данните

КЛАСОВЕ

- Всяко ниво използва единствено средствата на предходното

Какви са предимствата ?

КЛАСОВЕ

```
struct rat {
  int numer;
  int denom;
};
```

две полета:

numer – числителя

denom - знаменателя

двете полета се наричат **член-данни на записа (структурата)**

КЛАСОВЕ

- **Конструктори**

Конструкторите са член-функции, чрез които се инициализират **член-данните** на структурата.

Забележки:

- Името на конструктора съвпада с името на класа (структурата).
- Не се указва тип на връщания резултат.
- Изпълнява се автоматично при създаването на обекти.
- Не може да се извика явно.

КЛАСОВЕ

- **Конструктори**

Конструктор без параметри - **конструктор по подразбиране**.

```
struct rat {
   ...
   rat();
};
```

КЛАСОВЕ

- **Конструктори**

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

Пример:

```
rat number;// number се инициализира с
           // рационалното число 0/1
```

КЛАСОВЕ

- **Конструктори**

Конструктор с параметри

Възможно е да има произволен брой конструктори с различни параметри.

```
struct rat {
   ...
   rat(int, int);
};
```

КЛАСОВЕ

- **Конструктори**

```
rat::rat(int x, int y) {
   numer = x;
   denom = y;
}

// number се инициализира с
// рационалното число 3/4
rat number(3,4);

// number2 се инициализира с
// рационалното число 4/5
rat number2(4,5);
```

КЛАСОВЕ

- **Мутатори (сетъри)**

Член-функции, които променят член-данните на структурата.

```
struct rat {
  ...
  void setNumer(int);
};

void rat::setNumer(int x) {
  numer = x;
}
```

КЛАСОВЕ

- **Член-функции за достъп (гетъри)**

Член-функции, които извличат информация за член-данните на структурата.

Тези функции не променят член-данните на структурата.

Това се указва чрез записването на запазената дума const в декларацията и в края на заглавието в дефиницията им.

Трябва да извиква в себе си само const функции.

```
struct rat {
  ...
  int getNumer() const;
};

int rat::getNumer() const {
  return numer;
}
```

КЛАСОВЕ

- **Член-функции за достъп (гетъри)**

```
struct rat {
  ...
  int getDenom() const;
};

int rat::getDenom() const {
  return denom;
}
```

КЛАСОВЕ

- **Операции**

```
struct rat {
  ...
};

rat multRats(rat const & r1, rat const & r2) {
  rat r(
    r1.getNumer() * r2.getNumer(),
    r1.getDenom() * r2.getDenom());
  return r;
}
```

КЛАСОВЕ

- Спецификатори за достъп

Областта на един спецификатор за достъп започва от спецификатора и продължава до следващия спецификатор или до края на декларацията на класа.

```
struct rat {
 private:
   int numer;
   int denom;
 public:
   rat();
   rat(int, int);

   int getNumer() const;
   int getDenom() const;
};
```

КЛАСОВЕ

- Спецификаторите за достъп могат да се пропускат.
- По подразбиране спецификаторът за достъп в записа (структурата) е public.
- По подразбиране спецификаторът за достъп в класовете е private.

КЛАСОВЕ

- Дефиниране на клас

```
class rat {
 private:
   int numer;
   int denom;
 public:
   rat();
   rat(int, int);

   int getNumer() const;
   int getDenom() const;
};
```

КЛАСОВЕ

- **Капсулиране на информация**

Спецификаторът private забранява използването на член-данните numer и denom извън класа.

Процесът на скриване на информация се нарича капсулиране на информация.

КЛАСОВЕ

- **Интерфейс**

Член-функциите на класа rat са обявени като public. Те са видими извън класа и могат да се използват от външни функции.

Те се наричат интерфейс на класа.

Дефиниране на клас

- Декларация на клас

```
class rat;
```

- Декларация на член-функциите на клас

```
class rat {
  …
  public:
  int getNumer() const;
};
```

- Дефиниция на член-функциите (методите) на клас

```
int rat::getNumer() const {
  return numer;
}
```

КЛАСОВЕ

- Дефиницията на клас не заделя памет за него. Памет се заделя едва при дефинирането на обект от класа.
- Всеки обект разполага със собствени член-данни.
- Кодът на методите на класа НЕ се копира във всеки обект, а се намира само на едно място в паметта.

Дефиниране на клас

- Декларация на клас.
- Дефиниция на неговите член-функции.

Възможно е член-функциите на класа да се дефинират в тялото на класа.

```
class rat {
  int testFunc () {
    return numer + 1;
  }
};
```

** В този случай те се третират като вградени.*

Вградени функции

Вградени функции

- Кодът на тези функции не се съхранява на едно място, а се копира на всяко място в паметта, където има обръщение към тях.

- Използват се като останалите функции, но заглавието им се предшества от модификатора inline (при декларация и дефиниция).

Пример:

```
inline int func(int a, int b) {
  return (a+b)*(a-b);
}
```

Област на клас

- **Деклариран глобално** - започва от декларацията и продължава до края на програмата.

- **Деклариран локално** (вътре във функция или в тялото на клас) - всички негови член-функции трябва да са вградени (inline). В противен случай ще се получат функции, дефинирани във функция, което не е възможно.

КЛАСОВЕ

- Обекти - екземпляри на класа/структурата.
  За създаването се използват конструкторите.

<дефиниция-на_обект_на_клас> ::= <име_на_клас> <обект>
[=<име_на_клас>(<фактически_параметри>)]$_{опц}$
{,<обект>[=<име_на_клас>(<фактически_параметри>)]$_{опц}$}$_{опц}$
{, <обект>(<фактически_параметри>)}$_{опц}$
{, <обект> = <вече_дефиниран_обект>}$_{опц}$;
<обект> ::= <идентификатор>

Пример:

```
rat chislo1, chislo2(-3,4), chislo3 = rat(1,7);
```

са обекти инициализирани съответно с рационалните числа:
0/1, 1/7, -3/4

КЛАСОВЕ

- **Достъп до компонентите на клас**

Вътрешен достъп

Достъп на компонентите на класа до други компоненти на същия клас.

Член-функциите на клас имат пряк достъп до всички компоненти на класа в т.ч. до себе си без значение на секцията, в която се намират компонентите.

КЛАСОВЕ

- **Достъп до компонентите на клас**

Външен достъп

Достъп до компоненти на клас чрез обекти на класа, дефинирани във външни за класа функции.

- **Външни функции** - функциите, които не са член-функции на класа.

Външен достъп е възможен единствено до public компонентите на класа.

КЛАСОВЕ

- **Достъп до компонентите на клас**

Външен достъп

Достъпът до компонентите на клас (ако той е възможен) се реализира с помощта на обект и знака точка между името на обекта и член-данните и член-функциите на класа.

Пример:

Достъпът до конструктурите не се реализира с обект и точка!!! Те се извикват автоматично.

```
rat rNumber;
rNumber.getNumer(); // в public секция
```

КЛАСОВЕ

- Кодът на методите на класа не се копира във всеки обект, а се намира само на едно място в паметта.

- По какъв начин методите на един клас “разбират” за кой обект на този клас са били извикани?

Указател this

Работа на компилатора

1. Преобразува всяка член-функция на даден клас в обикновена функция с уникално име и един допълнителен параметър – указателят this.

```
int rat::getNumer() const{...}→
int rat::getNumer(rat* this) const {...}
```

2. Всяко обръщение към член-функция се транслира в съответствие първата част.

```
rat rNumber;
rNumber.getNumer(); → getNumer(&rNumber);
```

Указател this

Да направим функцията multRats член-функция на класа.

Външна функция:

```
rat multRats(rat const & r1, rat const & r2) {
  rat r(
    r1.getNumer() * r2.getNumer(),
    r1.getDenom() * r2.getDenom());
  return r;
}
```

Вътрешна функция:

```
void rat::multRats(rat const & r1, rat const & r2) {
  numer = r1.numer * r2.numer;
  denom = r1.denom * r2.denom;
}
```

Указател this

Да направим функцията multRats член-функция на класа.

Външна функция:

```
rat multRats(rat const & r1, rat const & r2) {
  rat r(
    r1.getNumer() * r2.getNumer(),
    r1.getDenom() * r2.getDenom());
  return r;
}
```

Вътрешна функция:

```
struct rat {
  public:
  rat multRats(rat const & r1, rat const & r2);
  …
};


void rat::multRats(rat const & r1, rat const & r2) {
  numer = r1.numer * r2.numer;  // this→numer
  denom = r1.denom * r2.denom;  // this→denom
}
```

КРАЙ

?