

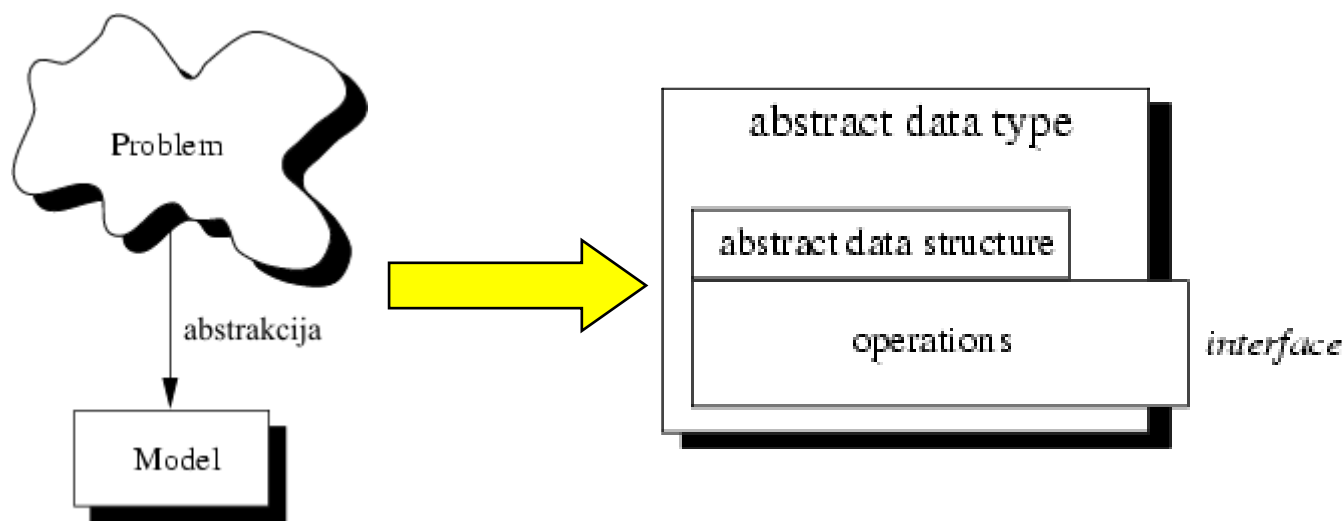
OOP – Objektno orientirano programiranje

Modul RPA

Darjan Toth

Osnove objektnega (predmetnega) programiranja

- Jezik C++ pozna dodaten podatkovni tip, ki ga C ni poznal: razred (ang. Class)
- Razred nam omogoča abstrakcijo podatkov:



Zakaj objektno orientirano programiranje?

- Lažja in bolj neposredno predstavitev realnega problema v programu z objekti
- Razred ima svoje lastnosti (podatkovni del programiranja)
- Razred vsebuje tudi svoje obnašanje (funkcionalni del programiranja)
- **Iz naštetega sledi, da razred vsebuje tako podatke kot tudi funkcije**



Osnovni pojmi objektnega programiranja



- **Razred** – sestavljen podatkovni tip ,ki vsebuje abstraktno predstavitev modela in vsebuje definicije lastnosti in metod
- **Objekt** – konkretna instanca (primerek) razreda – vsebuje vrednosti
- **Lastnost** (property) – označuje neko lastnost razreda
- **Metoda** (methode) – funkcija v razredu, ki bere ali spremeni neko lastnost objekta

Primeri razredov, objektov, lastnosti in metod

- *Primer računalnika – računalnik bi bil razred, lastnosti bi bile hitrost CPE, velikost RAM, tip trdega diska, itd., metode bi bile metoda za vnos lastnosti računalnika, metoda za izračun cene računalnika z DDV, objekt pa bi bil npr. službeni računalnik, ki bi vseboval konkretne vrednosti (za CPE – I7, velikost RAM 8 GB, itd.).*



- **Zaradi preglednosti običajno pišemo imena razredov z veliko začetnico, imena objektov pa z malo začetnico**



Deklaracija razreda in objektov



```
class ime_tipa_razreda
{
  tip1 ime_lastnost1;
  ....
  tipN ime_lastnostN;
  metoda1();
  ....
  metodaM();
};
class ime_tipa_razreda
  ime_objekta1,...,
  ime_objektaN;
// besedo class lahko
tudi izpustimo
```

```
class ime_tipa_razreda
{
  tip1 ime_lastnost1;
  ....
  tipN ime_lastnostN;
  metoda1();
  ....
  metodaM();
} ime_objekta1,...,
  ime_objektaN;
```

Enkapsulacija ali ograjevanje

- Z definiranjem razreda se v omejeni obseg zaprejo lastnosti in postopki (metode), ki dostopajo do lastnosti. Njihov dostop je namreč privzeto nastavljen kot privatni dostop.
- Ta značilnost objektnega programiranja se imenuje **enkapsulacija (ograjevanje, kapsuliranje)** in omogoča izvedbo **abstraktnega podatkovnega tipa** (razreda).
- **Do privatnih lastnosti ali metod lahko dostopajo samo člani razreda!**




Primer enkapsulacije

```
#include <iostream>
#include <string>
using namespace std;
class Kuza
{string pasma;
  int starost;
public:
  void lajanje()
  {cout << "Vau Vau Vau" << endl;}
  void beri()
  {cout << "Vnesi pasmo:"; getline(cin,pasma);
    cout << "Vnesi starost"; cin >> pasma;}
};
int main ()
{Kuza aron, piki; //deklaracija objektov Aron, Piki;
  aron.beri();//NAPAKA - PRIVATNO!!
  aron.lajanje(); //NAPAKA - PRIVATNO !!
return 0; }
```

Annotations:

- lastnosti** razreda (points to `string pasma;` and `int starost;`)
- metoda** ali **postopek** (points to `void lajanje()` and `void beri()`)

Warning icon: 

Javni postopki in metode

Za dostop do posameznih komponente razreda, moramo te označiti kot javne (uporabimo rezervirano besedo **public**):

```
class Kuza  
{public:
```

```
    string pasma;  
    int starost;
```

```
    void lajanje();  
};
```

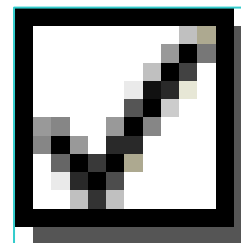
```
Kuza aron, piki;
```

```
/*Vse lastnosti in metode  
(postopki) so definirane kot  
javne in imamo dostop do  
njih. Vendar je tak način  
programiranja slab, saj ima  
enkapsulacija svoj namen !!  
*/
```

Dostop do zasebnih lastnosti objekta

```
class Kuza
{string pasma;
 int starost;
public:
 void lajanje()
 {cout << "Vau Vau Vau" << endl;}
....
....
};
int main()
{Kuza aron, piki;
....
....
}
```

/* Dober način programiranja:
lastnosti sta skriti (zasebni) in do
njiju lahko dostopamo samo z
metodama razreda ne pa iz drugih
funkcij ali glavnega programa! */



Dostop do zasebnih lastnosti objekta - nad.

- Recimo, da bi želeli v glavnem programu primerjati, kateri kuža je mlajši.
- Poizkusimo napisati:
if (piki.starost < aron. Starost) cout << "Piki je mlajši ";
- Prevajalnik nam bo javil napako, ker je starost zasebna lastnost razreda!
- Lahko pa razredu dodamo še dodatno javno dostopno metodo (vmesnik ali interface):
- `int vrniStarost () { return starost; }`
- Sedaj lahko napišemo:
if (piki.vrniStarost() < aron.vrniStarost()) cout << "Piki je mlajši" << endl;
else if (piki.vrniStarost() > aron.vrniStarost())
cout << "Aron je mlajši" << endl;
else cout << "Oba sta enako stara"<< endl;

Povzetek ograjevanja

- Vse javne komponente razreda predstavljajo **vmesnik razreda** (class interface) in z njim določimo način dela z lastnostmi. Poznamo t.i. get vmesnike in set vmesnike.
- Razrede definiramo tako, da skrijemo večino lastnosti (lahko vse) in metode, ki jih uporabnik razreda ne potrebuje za delo z razredom. **Na ta način se zmanjšuje možnost napak ali spreminjanje podatkov, ki jih ne bi smeli spremeniti.**



KONSTRUKTORJI IN DESTRUKTORJI



Pri definiranju metode oz. postopka, ki inicializira začetne vrednosti objekta, lahko pride do napak – ni klica v programu ali pa je inicializiran večkrat.

Zato je v uporabi posebna vrsta metode, ki se kliče avtomatično, ko se ustvari objekt.

To je t.i. konstruktor - rezervira pomnilnik za objekt in inicializira vrednosti za dani tip.



Konstruktor ima vedno enako ime kot tip razreda v katerem je definiran.

Primer konstruktorja



```
#include <iostream>
using namespace std;
class Moj
{ int a;
  public:
    void vpisi(int na);
    void izpisi();
    Moj();    /*Konstruktor (enako ime kot ime razreda)*/
};
/* Konstruktor ne vrača vrednosti in ne sme imeti
definiranega tipa funkcije, niti tipa void!! */
Moj::Moj()
{
a=333; }
```

Primer konstruktorja - nadaljevanje



```
void Moj::vpisi(int na)
{a=na;}
```

```
void Moj::izpisi()
{cout << a << endl;}
```

```
void main()
{
    Moj obj;
    obj.izpisi();           /*izpiše začetno vrednost: 333*/
    obj.vpisi(66);
    obj.izpisi();           /*izpiše vrednost 66*/
}
```

Pravila za konstruktorje



- **Konstruktor ima vedno enako ime kot razred v katerem je definiran**
- **Vsak razred ima lahko poljubno število konstruktorjev**
- **Kliče se tisti konstruktor, ki po številu in vrsti parametrov ustreza klicu pri inicializaciji**
- **Konstruktorji nikoli ne vračajo vrednosti**



Destruktor



- **Destruktor je metoda , ki izbriše objekt – sprosti pomnilnik. Konstruktor in destruktor imata isto ime, le da ima destruktor pred imenom znak ~:**
- `~Moj();`
- Vedno obstaja privzet destruktor, ki ne more sprejeti argumentov.



Destruktor



- **Destruktor je metoda , ki izbriše objekt – sprosti pomnilnik. Konstruktor in destruktor imata isto ime, le da ima destruktor pred imenom znak ~:**
- `~Moj();`
- Vedno obstaja privzet destruktor, ki ne more sprejeti argumentov.

