Napredno programiranje i programski jezici

03 OOP

Fakultet tehničkih nauka, Novi Sad
23-24/Z
Dunja Vrbaški

```
class MyClass {

private:

   int x;


public:

   int y;

};
```

```
int main()
{
    MyClass mc;
    //mc.x = 5;
    mc.y = 3;

    return 0;
}
```

```
class MyClass {

private:
    int x;

public:
    void setX(int xx) {
        x = xx;
    }

    int getX() const {
        return x;
    }

    void print() const {
        cout << "x = " << x << endl;
    }

};
```

```
int main()
{
    MyClass mc;
    mc.setX(5);
    mc.print();

    int y = mc.getX();

    return 0;
}
```

```
class MyClass {

private:
    int x;

public:
    void setX(int xx) {
        x = xx;
    }

    int getX() const {
        return x;
    }

    void print() const {
        cout << "x = " << x << endl;
    }
};
```

```
void print(const MyClass &mc) {
    cout << "Ispis SF: x = " <<
    mc.getX() << endl;
}
```

```
int main()
{
    MyClass mc;
    mc.setX(5);
    mc.print();
    print(mc);

    return 0;
}
```

Slobodne funkcije vs metode

```
class MyClass {

private:
    int x;

public:
    void setX(int xx) {
        x = xx;
    }

    int getX() const {
        return x;
    }

    void print() const {
        cout << "x = " << x << endl;
    }
};
```

```
void print(const MyClass &mc) {
    cout << "Ispis SF: x = " <<
    mc.getX() << endl;
}
```

```
int main()
{
    MyClass mc;
    mc.setX(5);
    mc.print();
    print(mc);

    return 0;
}
```

Šta će se desiti?

Napisati klasu koja modelira krug. Omogućiti izračunavanje površine i obima.
Testirati klasu.

Pseudokod, ideja

```
PI 3.14

class Krug
{
    double r;

    void setR(double rr)...
    double getR()...
    double getO()...
    double getP()..
};
```

```
#define PI 3.14

class Krug
{
private:
    double r;
public:
    void setR(double rr) {
        r = rr;
    }
    double getR() const {
        return r;
    }
    double getO() const {
        return 2 * r * PI;
    }
    double getP() const {
        return r * r * PI;
    }
};
```

```
#define PI 3.14

class Krug
{
private:
    double r;
public:
    void setR(double rr) {
        r = rr;
    }
    double getR() const {
        return r;
    }
    double getO() const {
        return 2 * r * PI;
    }
    double getP() const {
        return r * r * PI;
    }
};
```

```
int main()
{
     Krug k;
     k.setR(5);
     cout << "O = " << k.getO() << endl;
     cout << "P = " << k.getP() << endl;

     return 0;
}
```

Pseudokod, ideja

```
class Krug
{
    double r;

    void setR(double rr)...
    double getR()...
    double getO()...
    double getP()..
};
```

krug.hpp

```
class Krug
{
private:
    double r;
public:
    void setR(double);
    double getR() const;
    double getO() const;
    double getP() const;
};
```

krug.hpp

```
class Krug
{
private:
    double r;
public:
    void setR(double);
    double getR() const;
    double getO() const;
    double getP() const;
};
```

krug.cpp

```
#define PI 3.14
#include "krug.hpp"

void Krug::setR(double rr) {
    r = rr;
}
double Krug::getR() const {
    return r;
}
double Krug::getO() const {
    return 2 * r * PI;
}
double Krug::getP() const {
    return r * r * PI;
}
```

```
void Krug::setR(double rr) {
    r = rr;
}
double Krug::getR() const {
    return r;
}
double Krug::getO() const {
```

```
class Krug
{
private:
    double r;
public:
    void setR(double);
    double getR() const;
    double getO() const;
    double getP() const;
};
```

```
#include "krug.hpp"

int main()
{
    Krug k;
    k.setR(5);
    cout << "O = " << k.getO() << endl;
    cout << "P = " << k.getP() << endl;

    return 0;
}
```

```
#define PI 3.14
#include "krug.hpp"
#include <cmath>


void Krug::setR(double rr) {
    r = rr;
}
double Krug::getR() const {
    return r;
}
double Krug::getO() const {
    return 2 * r * M_PI;
}
double Krug::getP() const {
    return pow(r, 2) * M_PI;
}
```

```
void Krug::setR(double rr) {
    r = rr;
}
double Krug::getR() const {
    return r;
}
double Krug::getO() const {
}
double
}
```

```
class Krug
{
private:
    double r;
public:
    void setR(double);
    double getR() const;
    double getO() const;
    double getP() const;
};
```

```
#include "krug.hpp"

int main()
{
    Krug k;
    k.setR(5);
    cout << "O = " << k.getO() << endl;
    cout << "P = " << k.getP() << endl;

    return 0;
}
```

Šta će biti ispisano?

KONSTRUKTORI

Posebne metode koje se izvršavaju prilikom kreiranja objekata.

Svaka klasa može imati definisano 0 ili više (korisničkih) konstruktora.

Ako se ne definiše nijedan konstruktor kompajler generiše podrazumevani
(isto kao korisnički bez tela i parametara)

```
class Krug
{
private:
    double r;
public:
    Krug();

    void setR(double);
    double getR() const;
    double getO() const;
    double getP() const;
};
```

```
Krug::Krug() {
   r = 1;
}

void Krug::setR(double rr) {
    r = rr;
}
double Krug::getR() const {
    return r;
}
double Krug::getO() const {
    return 2 * r * PI;
}
double Krug::getP() const {
    return r * r * PI;
}
```

podrazumevani konstruktor, konstruktor bez parametara

default ctor

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    void setR(double rr) {...}
    double getR() const {...}
    double getO() cons {...}
    double getP() const {...}
};
```

Implementacija prebačena u hpp

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    void setR(double rr) {...}
    double getR() const {...}
    double getO() cons {...}
    double getP() const {...}
};
```

```
#include "krug.hpp"

int main()
{
    Krug k;
    k.setR(5);
    cout << "O = " << k.getO() << endl;
    cout << "P = " << k.getP() << endl;

    return 0;
}
```

Šta će biti ispisano?

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    Krug(double rr) {
       r = rr;
    }

    void setR(double rr) {...}
    double getR() const {...}
    double getO() cons {...}
    double getP() const {...}
};
```

konstruktor sa parametrima

```
#include "krug.hpp"

int main()
{
    Krug k;
    cout << "O = " << k.getO() << endl;
    cout << "P = " << k.getP() << endl;

    Krug k1(5);
    cout << "O = " << k1.getO() << endl;
    cout << "P = " << k1.getP() << endl;

    return 0;
}
```

Obrati pažnju:
Isto ime, različiti parametri

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    Krug(double rr) {
       r = rr;
    }

    void setR(double rr) {...}
    double getR() const {...}
    double getO() cons {...}
    double getP() const {...}
};
```

```
#include "krug.hpp"

int main()
{
    Krug k, k1(5);
    cout << "O = " << k.getO() << endl;
    cout << "P = " << k.getP() << endl;

    cout << "O = " << k1.getO() << endl;
    cout << "P = " << k1.getP() << endl;

    return 0;
}
```

klasa je tip

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    Krug(double rr) {
       r = rr;
    }

    Krug(const Krug& k) {
       r = k.r;
    }
    ...
};
```

konstruktor kopije

```
#include "krug.hpp"

int main()
{
    Krug k, k1(5), k2(k);
    cout << "O = " << k.getO() << endl;
    cout << "P = " << k.getP() << endl;

    cout << "O = " << k1.getO() << endl;
    cout << "P = " << k1.getP() << endl;

    cout << "O = " << k2.getO() << endl;
    cout << "P = " << k2.getP() << endl;

    return 0;
}
```

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    Krug(double rr) {
       r = rr;
    }

    Krug(const Krug& k) {
       r = k.r;
    }

   void setR(double rr) {...}
   double getR() const {...}
   double getO() cons {...}
   double getP() const {...}
};
```

```
#include "krug.hpp"

int main()
{
    Krug k, k1(5), k2(k);
    cout << "O = " << k.getO() << endl;
    cout << "P = " << k.getP() << endl;

    cout << "O = " << k1.getO() << endl;
    cout << "P = " << k1.getP() << endl;

    cout << "O = " << k2.getO() << endl;
    cout << "P = " << k2.getP() << endl;

    return 0;
}
```

```
class Krug
{
private:
    double r;
public:
    Krug() {
        r = 1;
    }

    Krug(double rr) {
       r = rr;
    }

    Krug::Krug(const Krug& k) {
       r = k.r;
    }
};
```

```
#include "krug.hpp"

int main()
{
    Krug k, k1(5), k2(k);
    ...
    return 0;
}
```

error: no matching function for call to 'Krug::Krug()'|

```
class Krug
{
private:
    double r;
public:
    Krug() {
        r = 1;
    }

    Krug(double rr = 1) {
       r = rr;
    }

    Krug::Krug(const Krug& k) {
       r = k.r;
    }
};
```

```
#include "krug.hpp"

int main()
{
    Krug k, k1(5), k2(k);
    ...

    return 0;
}
```

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    Krug(double rr) {
       r = rr;
    }

    Krug(const Krug& k) {
       r = k.r;
    }

};
```

```
#include "krug.hpp"

int main()
{
    Krug k, k1(5), k2(k);
    ...

    return 0;
}
```

Šta će se desiti?

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    Krug(double rr) {
       r = rr;
    }

    Krug(const Krug& k) {
       r = k.r;
    }

};
```

```
#include "krug.hpp"

int main()
{
    Krug k, k1(5), k2(k);
    ...

    return 0;
}
```

```
O = 6.28319
P = 3.14159
O = 31.4159
P = 78.5398
O = 6.28319
P = 3.14159
```

Nismo morali da pišemo KK.

Zašto pišemo na vežbama?

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    Krug(double rr) {
       r = rr;
    }

    Krug(const Krug& k) {
       r = k.r;
    }

};
```

Konstruktor kopije se poziva i na drugim mestima: prilikom drugih načina inicijalizacije ili prilikom prosleđivanje i vraćanja vrednosti iz funkcije.

Posledično - postoje situacije i kad se ne poziva.

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    Krug(double rr) {
       r = rr;
    }

    Krug(const Krug& k) {
       r = k.r;
    }

};
```

```
#include "krug.hpp"

int main()
{
    Krug k, k1(5), k2(k);

    k2 = k1;

    ...

    return 0;
}
```

Posmatrajmo sada dodelu.
Šta će se desiti?

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    Krug(double rr) {
       r = rr;
    }

    Krug(const Krug& k) {
       r = k.r;
    }

};
```

```
#include "krug.hpp"

int main()
{
    Krug k, k1(5), k2(k);

    k2 = k1;

    ...

    return 0;
}
```

```
O = 6.28319
P = 3.14159
O = 31.4159
P = 78.5398
O = 31.4159
P = 78.5398
```

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    Krug(double rr) {
       r = rr;
    }

    Krug(const Krug& k) {
       r = k.r;
    }

};
```

```
#include "krug.hpp"

int main()
{
    Krug k, k1(5), k2(k);

    k2 = k1;
    k2.setR(10);

    ...

    return 0;
}
```

Ako jednom objektu promenimo vrednost, da li će se promeniti u oba?

```
class Krug
{
private:
    double r;
public:
    Krug() {
       r = 1;
    }

    Krug(double rr) {
       r = rr;
    }

    Krug(const Krug& k) {
       r = k.r;
    }

};
```

```
#include "krug.hpp"

int main()
{
    Krug k, k1(5), k2(k);

    k2 = k1;
    k2.setR(10);

    ...

    return 0;
}
```

```
O = 6.28319
P = 3.14159
O = 31.4159
P = 78.5398
O = 62.8319
P = 314.159
```

OPERATORI NEW/DELETE

```
int *pnizC = (int*)malloc(10*sizeof(int));
```

```
int *p = new int;
```

```
MyClass *pMc = new MyClass;
```

```
int *pniz = new int[10];
```

```
MyClass *pnizMc = new MyClass[10];
```

```
int *pMc = new MyClas;
int *pnizMc = new MyClass[10];
```

Alocira se memorija (i poziva se konstruktor!)

```
delete pMc;
delete[] pnizMc;
```

Oslobađa se memorija

```
class MyClass {

private:
    int x;
    int* niz;

public:
   MyClass() {
      niz = new int[10];
      // pp da smo popunili ovde niz nekim vrednostima
   }

   // pp da postoje metode koje omogućavaju modifikaciju
      elemenata niza
};
```

```
#include "krug.hpp"

int main()
{
    MyClass mc, mc1(mc), mc2;
    mc2 = mc1;
    return 0;
}
```

Imamo implicitno kreiran konstruktor kopije.

Šta se dešava u slučaju eksplicitnog poziva KK ili dodele?

```
class MyClass {

private:
    int x;
    int* niz;

public:
   MyClass() {
      niz = new int[10];
      // pp da smo popunili ovde niz nekim vrednostima
   }

  // pp da postoje metode koje omogućavaju modifikaciju
     elemenata niza
};
```

```
#include "krug.hpp"

int main()
{
    MyClass mc, mc1(mc), mc2;
    mc2 = mc1;
    return 0;
}
```

shallow vs deep copy

Ako koristimo biblioteku: pouzdaniji i lakši razvoj. Prvo treba shvatiti osnove, bar delom, da bismo znali šta nam je ponuđeno.

```
class MyClass {

private:
    int x;
    int* niz;

public:

   MyClass() {
      niz = new int[10];
      ...
   }
   ...

};
```

Ako imamo dinamičko zauzimanje memorije verovatno nam
- treba nam konstruktor kopije
- treba rešiti dodelu. Kako?

DESTRUKTORI

```
class Krug
{
private:
    double r;
public:
    Krug() : Krug(1) { }

    Krug(double rr) {
       r = rr;
    }

    Krug(const Krug& k) {
       r = k.r;
    }

    ~Krug() {
        cout << “destruktor” << endl;
    }
};
```

```
#include "krug.hpp"

int main()
{
    Krug k, k1(5), k2(k);
    cout << "O = " << k.getO() << endl;
    cout << "P = " << k.getP() << endl;

    cout << "O = " << k1.getO() << endl;
    cout << "P = " << k1.getP() << endl;

    cout << "O = " << k2.getO() << endl;
    cout << "P = " << k2.getP() << endl;

    return 0;
}
```

Kada nam treba destruktor?

- specifična akcija koja treba da se desi u tom momentu

- oslobađanje dinamički zauzete memorije
- oslobađanje drugih resursa
- ....

```
class MyClass {

private:
    int x;
    int* niz;

public:
   MyClass() {
      niz = new int[10];
      // pp da smo popunili ovde niz nekim vrednostima
   }

  // pp da postoje metode koje omogućavaju modifikaciju
     elemenata niza

   ~MyClass() {
      delete[] niz;
   }

};
```

Ako imamo dinamičko zauzimanje memorije - treba nam destruktor

```
class MyClass {

private:
    int x;
    int* niz;

public:
   MyClass() {
      niz = new int[10];
      // pp da smo popunili ovde niz nekim vrednostima
   }

  // pp da postoje metode koje omogućavaju modifikaciju
     elemenata niza

   ~MyClass() {
      delete[] niz;
   }

};
```

Ako imamo dinamičko zauzimanje memorije potrebni su nam naši
- destruktor
- konstruktor kopije
- dodela. **Kako?**

Par napomena

Ima još (dosta) priče o konstruktorima i destruktorima
Javljaće se pitanja kako napredujete
Postoji i move ctor

Konstruktori i destruktori važni za upravljanje resursima
Postoje klase i alati u standardnoj biblioteci i opšta uputstva za bolje upravljanje (++)
[rule of three, rule of five, rule of zero]

Zašto kompajliranje? (0 bodova)
Zašto testiranje? (negativni bodovi)