# Napredno programiranje i programski jezici

## 03 C++ (strukturne veze)

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

- Jednom napisane klase (razvijane, testirane) se mogu stalno iznova koristiti - reuse, reusability

- Enkapsulacija (private/public, header fajlovi)
- Zasebno unapređivanje i menjanje

- jeftiniji, brži, pouzdaniji razvoj (?)

- razvoj sopstvenih klasa vs korišćenje standardne biblioteke vs third-party

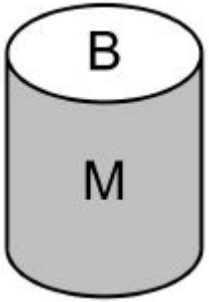
Objekti različitih klasa

- komuniciraju među sobom i/ili utiču jedni na druge
- sadrže, na različite načine, druge objekte i/ili su izgrađeni od njih

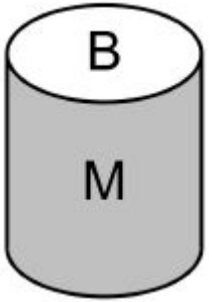
## PRIMER: VALJAK

Date su klase Krug i Pravougaonik. Realizovati klasu Valjak uz pomoć datih klasa.

```
class Krug {
   ...
}

class Pravougaonik {
   ...
}

class Valjak {
    // baza → Krug
    // omotac → Pravougaonik
}
```



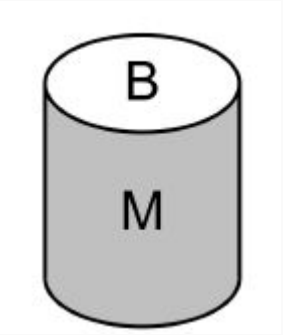
```
class Krug {
   ...
}

class Pravougaonik {
   ...
}

class Valjak {
private:
    Krug B;
    Pravougaonik M;
public:
    ...
}
```



```
class Krug {
private:
   double r;
public:
   Krug(double rr = 1) { r = rr; }
   double getR() const { return r; }
   double getO() const { return 2 * r * M_PI; }
   double getP() const { return r * r * M_PI; }
};
```

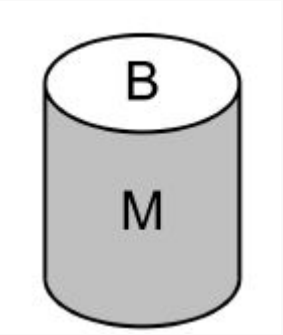
```
class Pravougaonik {
private:
   double a;
   double b;
public:
   Pravougaonik(double aa = 1, double bb = 2) { a = aa; b = bb; }
   double getA() const { return a; }
   double getB() const { return b; }
   double getO() const { return 2*a+2*b; }
   double getP() const { return a*b; }
};
```

```
class Valjak {
private:
   Krug B;
   Pravougaonik M;
public:
   Valjak(...)

   double getR() const { return B.getR(); }
   double getH() const { return M.getB(); }

   double getP() const { return 2*B.getP()+M.getP(); }
   double getV() const { return B.getP()*getH(); }
};
```



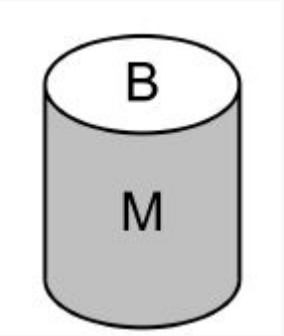

*public (javni programski interfejs) VS  private (kako je interno realizovano)*
*+*
*reuse funkcionalnosti (metode: getO, getP, getH VS metode getR, getA, getB)*

```
class Valjak {
private:
   Krug B;
   Pravougaonik M;
public:
   Valjak(...)

   double getR() const { return B.getR(); }
   double getH() const { return M.getB(); }

   double getP() const { return 2*B.getP()+M.getP(); }
   double getV() const { return B.getP()*getH(); }
};
```

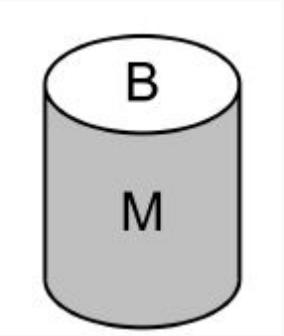



Obratiti pažnju: Nema setera
Kako definišemo valjak?

```
class Valjak {
private:
   Krug B;
   Pravougaonik M;
public:
   Valjak(double rr=1, double hh=1):... {...}

   double getR() const { return B.getR(); }
   double getH() const { return M.getB(); }

   double getP() const { return 2*B.getP()+M.getP(); }
   double getV() const { return B.getP()*getH(); }
};
```



```
class Valjak {
private:
   Krug B;
   Pravougaonik M;
public:
   Valjak(double rr=1, double hh=1): B(...), M (...) {...}

   double getR() const { return B.getR(); }
   double getH() const { return M.getB(); }

   double getP() const { return 2*B.getP()+M.getP(); }
   double getV() const { return B.getP()*getH(); }
};
```




Inicijalizacija

```
private:
   int x;
   double y;
```

```
MyClass(int xx, int yy) { x = xx; y = yy;}
```

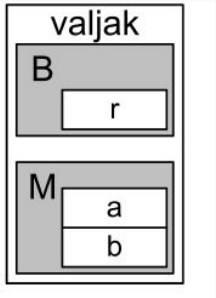
ILI

```
MyClass(int xx, int yy): x(xx), y(yy) {}
```

```
private:
   Krug B;
   Pravougaonik M;
```

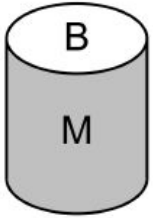
```
Valjak(double rr, double hh) {...}
```

```
Valjak(double rr, double hh): B(...), M (...) {...}
```



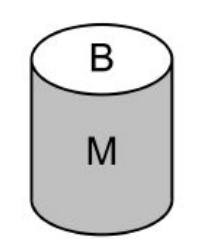
```
private:
    Krug B;
    Pravougaonik M;
```

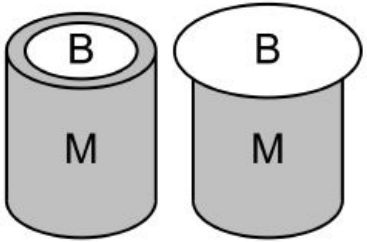
```
Valjak(double rr, double hh): B(rr), M (2 * rr * M_PI, hh) {}
```



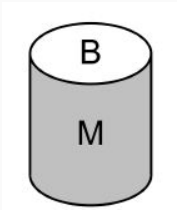
```
private:
    Krug B;
    Pravougaonik M;
```

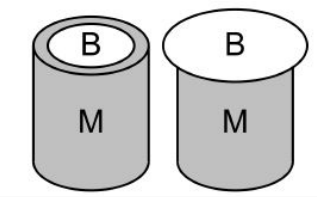
```
Valjak(double rr, double hh): B(rr), M (2 * rr * M_PI, hh) {}
```



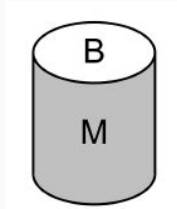
```
class Valjak {
private:
   Krug B;
   Pravougaonik M;
public:
   Valjak(double rr, double hh): B(rr), M (2 * rr * M_PI, hh) {}

   double getR() const { return B.getR(); }
   double getH() const { return M.getB(); }

   double getP() const { return 2*B.getP()+M.getP(); }
   double getV() const { return B.getP()*getH(); }
};
```

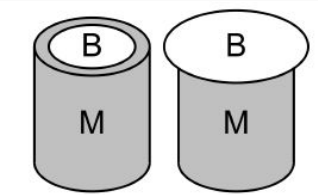



- iskorišćene klase Krug i Pravougaonik i njihova logika (algoritam) za O i P
- implementacija sakrivena, funkcionalnost od značaja javna
- nema objekata koji nisu valjani

```
class Valjak {
private:
   Krug B;
   Pravougaonik M;
public:
   Valjak(double rr=1, double hh=1): B(rr), M (2 * rr * M_PI, hh) {}

   double getR() const { return B.getR(); }
   double getH() const { return M.getB(); }

   double getP() const { return 2*B.getP()+M.getP(); }
   double getV() const { return B.getP()*getH(); }
};
```



## PRIMER: OSOBA

Data je klasa DinString. Realizovati klasu Osoba koja ima ime i prezime.

```
class Osoba {
private:
   DinString ime, prezime;
public:
   Osoba(const char *s1 = "", const char *s2 = "") ...

   Osoba(const DinString &ds1, const DinString &ds2) ...

   Osoba(const Osoba &ro) ...

   void predstaviSe() const { ... }
};
```

```
class Osoba {
private:
   DinString ime, prezime;
public:
   Osoba(const char *s1 = "", const char *s2 = "") : ime(s1), prezime(s2) {}

   Osoba(const DinString &ds1, const DinString &ds2) : ime(ds1), prezime(ds2) {}

   Osoba(const Osoba &ro) : ime(ro.ime), prezime(ro.prezime){}

   void predstaviSe() const {
      cout << "Zovem se " << ime << " " << prezime << "." << endl;
   }
};
```

```
class Osoba {
private:
   DinString ime, prezime;
public:
   Osoba(const char *s1 = "", const char *s2 = "") : ime(s1), prezime(s2) {
      cout << "Osoba: Konstruktor 1." << endl;
   }

   Osoba(const DinString &ds1, const DinString &ds2) : ime(ds1), prezime(ds2) {
      cout << "Osoba: Konstruktor 2." << endl;
   }

   Osoba(const Osoba &ro) : ime(ro.ime), prezime(ro.prezime){
      cout << "Osoba: Konstruktor 3." << endl;
   }

   ~Osoba() {
      cout << "Osoba: Destruktor." << endl;
   }

   void predstaviSe() const {
      cout << "Zovem se " << ime << " " << prezime << "." << endl;
   }
};
```

Koristite listu inicijalizatora + inicijalizujte sva polja

## ZADATAK

*(neobavezno)*

Izmeniti konstruktore kod kruga i kvadrata tako da imaju listu inicijalizatora.

Primer za Valjak.
Dodati destruktore u svaku klasu (iako nisu neophodni).
Dodati u svaki konstruktor i destruktor svake klase ispis koji će se razlikovati.
Kreirati različite objekte (valjak) koristeći konstruktor bez parametara, sa parametrima i kopije (implicitni) i posmatrati kojim redom se izvršavaju konstruktori i destruktori.

Preklopiti operator ispisa za sve tri  klase i testirati.