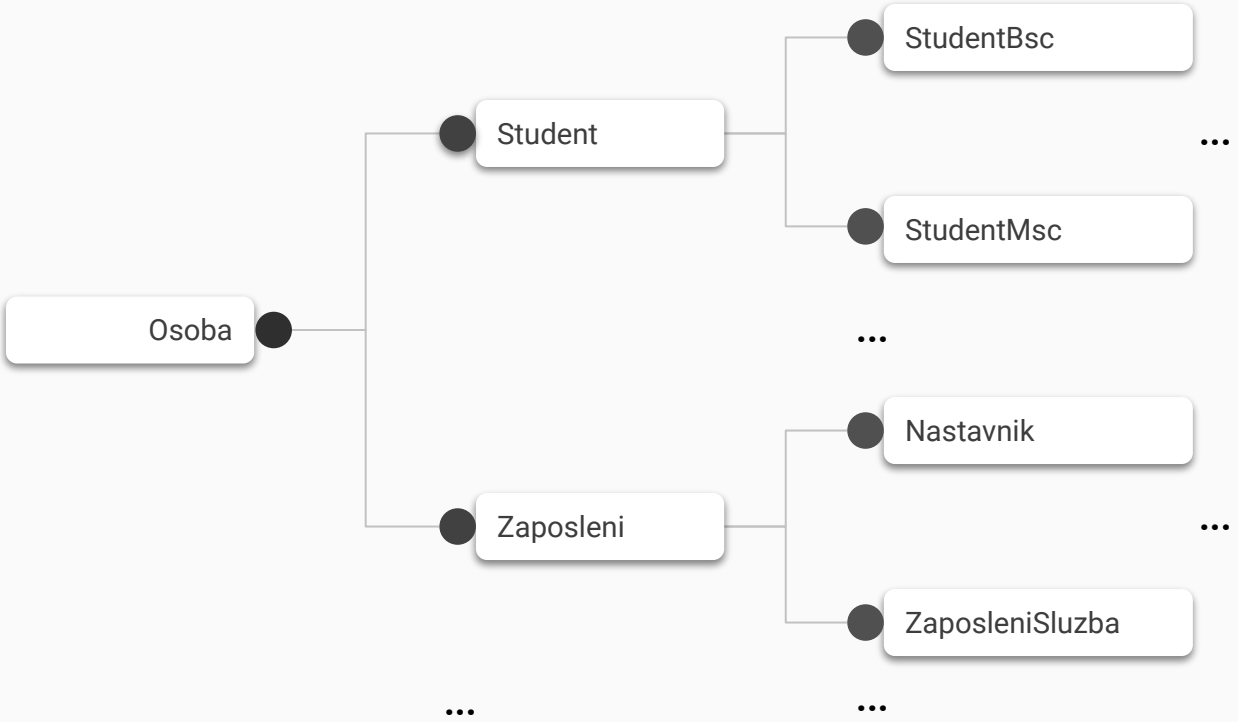
Napredno programiranje i programski jezici

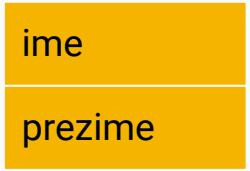
04 C++ (nasleđivanje)

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

Valjak: Krug, Pravougaonik
Osoba: DinString (ime, prezime)


Osoba
Student
Zaposleni
StudentBsc
StudentMsc
Nastavnik
ZaposleniSluzba
...
...
...
...
...

```
class Student : Osoba
```



```
class Student : Osoba
```

```
class B : A
```

A - nadklasa, roditeljska klasa, parent class, superclass
B - podklasa, izvedena klasa, subclass
“nasleđena klasa“?

B preuzima polja i metode iz klase A i dodaje svoja nova polja i metode
Svaki objekat klase B je ujedno i objekat klase A

Dodavanje neke metode u B može, zapravo, predstavljati izmenu ponašanja metode iz A

```
class Student : Osoba
```

Svaki Student je i Osoba

```
class Student : Osoba
```

Svaki Student **je** i Osoba

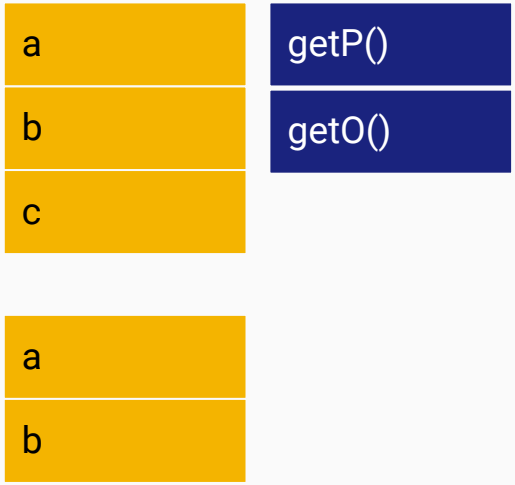
Svaka Osoba, pa i Student, **ima** ime i prezime (DinString)

Nasleđivanje:
Svaki Student **je** i Osoba — **is-a**

Agregacija/Kompozicija:
Svaka Osoba, pa i Student, **ima** ime i prezime (DinString) — **has-a**
Svaka Osoba, pa i Student, je **sačinjena od** imena i prezimena (DinString) — **part of**

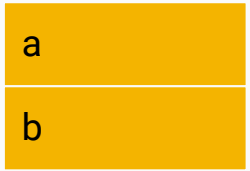
Posmatrajmo klase Trougao i JednakokrakiTrougao.

```
class JKTrougao : Trougao
```



```
class JKTrougao : Trougao
```

Formiramo podskup instanci
Postavljamo ograničenja
Definišemo specijalizaciju

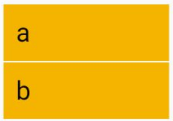


```
class Student : Osoba
```

Ekstenzija, proširivanje
Student ima više polja i više operacija


```
class JKTrougao : Trougao
```

Specijalizacija
JKTrougao ima manje polja i nema nove metode



Vrlo često: jedno i drugo, specijalizacija + proširivanje

Neka ograničenja (na vrednosti polja, formiranje podskupa) ali i nove operacije za te specijalizovane slučajeve

```
class A {
public:
    int x;

    void inc() {
        x++;
    }

};

class B : public A {

};
```

```
class A {
public:
    int x;

    void inc() {
        x++;
    }

};

class B : public A {

};
```

```
int main()
{
    A a;
    a.x = 5;
    B b;
    b.x = 3;

    cout << "a.x = " << a.x << endl;
    cout << "b.x = " << b.x << endl;

    return 0;
}
```

```
class A {
public:
    int x;

    void inc() {
        x++;
    }

};

class B : public A {
public:
    void dec() {
        x--;
    }
};
```

```
int main()
{
    A a;
    a.x = 5;
    B b;
    b.x = 3;

    cout << "a.x = " << a.x << endl;
    cout << "b.x = " << b.x << endl;

    return 0;
}
```

```
class A {
private:
    int x;
public:
    void inc() {
        x++;
    }
};

class B : public A {
public:
    void dec() {
        x--;
    }
};
```

```
int main()
{
    A a;
    a.x = 5;
    B b;
    b.x = 3;

    cout << "a.x = " << a.x << endl;
    cout << "b.x = " << b.x << endl;

    return 0;
}
```

```
error: 'int A::x' is private
5X
```

```
class A {
private:
    int x;
public:
    void inc() {
        x++;
    }
};

class B : public A {
public:
    void dec() {
        x--;
    }
};
```

private članovi →
dostupni samo u klasi u kojoj su definisani

Da li onda B klasa nasleđuje private članove?

```
class A {
protected:
    int x;
public:
    void inc() {
        x++;
    }
};

class B : public A {
public:
    void dec() {
        x--;
    }
};
```

protected članovi →
dostupni u klasi u kojoj su definisani i svim klasama koje nasleđuju od nje.

Da li onda B klasa ima pristup protected članovima?

```
class A {
protected:
    int x;
public:
    void inc() {
        x++;
    }
};

class B : public A {
public:
    void dec() {
        x--;
    }
};
```

```
int main()
{
    A a;
    a.x = 5;
    B b;
    b.x = 3;

    cout << "a.x = " << a.x << endl;
    cout << "b.x = " << b.x << endl;

    return 0;
}
```

```
error: 'int A::x' is private
4X
```

private članovi	→ dostupni samo u klasi u kojoj su definisani.
protected članovi	→ dostupni u klasi u kojoj su definisani i svim klasama koje nasleđuju od nje.
public članovi	→ dostupni svima.

private članovi → dostupni samo u klasi u kojoj su definisani.

protected članovi → dostupni u klasi u kojoj su definisani i svim klasama koje nasleđuju od nje.

public članovi → dostupni svima.

```
class A {
...
};

class B : public A {
...
};
```

```
class A {
...
};

class B : private A {
...
};
```

```
class A {
...
};

class B : protected A {
...
};
```

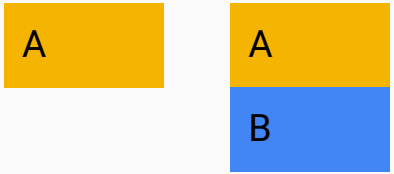
```
class A {
...
};

class B : public A {
...
};
```

Član u natklasi je:	Način izvođenja je:	Isti član u potklasi je:
public	public	public
public	protected	protected
public	private	private
protected	public	protected
protected	protected	protected
protected	private	private
private	public	nije vidljiv
private	protected	nije vidljiv
private	private	nije vidljiv

```
class A {
protected:
    int x;
public:
   void ispisA() {
        cout << "A x: " << endl;
   }
};

class B : public A {
private:
    int y;
public:
   void ispisB() {
        cout << "B x: " << x << endl;
        cout << "B y: " << y << endl;
   }
};
```

```
int main()
{
    A a1, a2;
    B b1, b2;

    a1.ispisA();
    b1.ispisA();

    a1.ispisB();
    b1.ispisB();

    return 0;
}
```

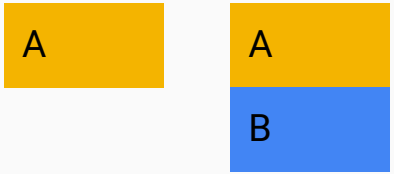
?



Nisu dva objekta, samo je jedan, ali sa preuzetim roditeljskim delom

```
class A {
protected:
    int x;
public:
   void ispisA() {
        cout << "A x: " << endl;
   }
};

class B : public A {
private:
    int y;
public:
   void ispisB() {
        cout << "B x: " << x << endl;
        cout << "B y: " << y << endl;
   }
};
```

```
int main()
{
    A a1, a2;
    B b1, b2;

    a1.ispisA();
    b1.ispisA();

    a1.ispisB();
    b1.ispisB();

    return 0;
}
```

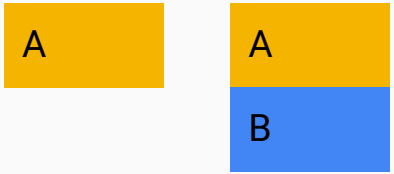



```
error: 'class A' has no member named 'ispisB'
```

```
class A {
protected:
    int x;
public:
   void ispisA() {
        cout << "A x: " << endl;
   }
};

class B : public A {
private:
    int y;
public:
   void ispisB() {
        cout << "B x: " << x << endl;
        cout << "B y: " << y << endl;
   }
};
```

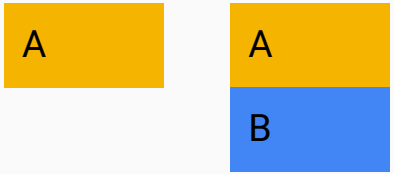
```
int main()
{
    A a1, a2;
    B b1, b2;

    a1 = b1;
    b2 = a1;

    return 0;
}
```

?



```
class A {
protected:
    int x;
public:
   void ispisA() {
        cout << "A x: " << endl;
   }
};

class B : public A {
private:
    int y;
public:
   void ispisB() {
        cout << "B x: " << x << endl;
        cout << "B y: " << y << endl;
   }
};
```

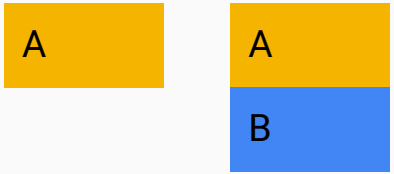
```
int main()
{
    A a1, a2;
    B b1, b2;

    a1 = b1;
    b2 = a1;

    return 0;
}
```




```
error: no match for 'operator=' (operand types are 'B'
and 'A')|
```

```
class A {
protected:
    int x;
public:
   void ispisA() {
        cout << "A x: " << endl;
   }
};

class B : public A {
private:
    int y;
public:
   void ispisB() {
        cout << "B x: " << x << endl;
        cout << "B y: " << y << endl;
   }
};
```

```
int main()
{
    A a1, a2;
    B b1, b2;

    a1 = b1;

    return 0;
}
```



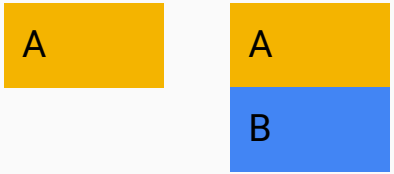
Slicing

(ume da bude problematično kad se uključe reference. Kasnije...)

```
class A {
protected:
    int x;
public:
   void ispisA() {
        cout << "A x: " << endl;
   }
};

class B : public A {
private:
    int y;
public:
   void ispisB() {
        cout << "B x: " << x << endl;
        cout << "B y: " << y << endl;
   }
};
```

```
int main()
{
    A a1, a2;
    B b1, b2;

    a1 = b1;

    a1.ispisA();
    a1.ispisB();

    return 0;
}
```

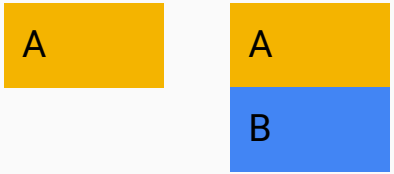
?



```
class A {
protected:
    int x;
public:
   void ispisA() {
        cout << "A x: " << endl;
   }
};

class B : public A {
private:
    int y;
public:
   void ispisB() {
        cout << "B x: " << x << endl;
        cout << "B y: " << y << endl;
   }
};
```

```
int main()
{
    A a1, a2;
    B b1, b2;

    a1 = b1;

    a1.ispisA();
    a1.ispisB();

    return 0;
}
```

?

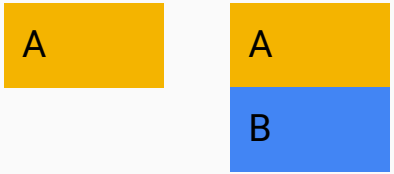



```
error: 'class A' has no member named 'ispisB'
```

```
class A {
protected:
    int x;
public:
   void ispis() {
        cout << "A x: " << endl;
   }
};

class B : public A {
private:
    int y;
public:
   void ispisB() {
        cout << "B x: " << x << endl;
        cout << "B y: " << y << endl;
   }
};
```

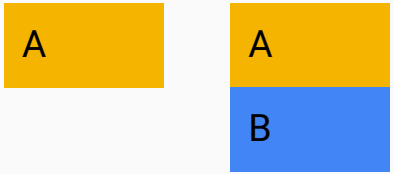
```
int main()
{
    A a1, a2;
    B b1, b2;

    a1.ispis();
    b1.ispis();

    return 0;
}
```

?



```
class A {
protected:
    int x;
public:
   void ispis() {
        cout << "A x: " << endl;
   }
};

class B : public A {
private:
    int y;
public:
   void ispis() {
        cout << "B x: " << x << endl;
        cout << "B y: " << y << endl;
   }
};
```

```
int main()
{
    A a1, a2;
    B b1, b2;

    a1.ispis();
    b1.ispis();

    return 0;
}
```

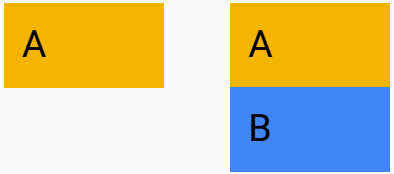


Redefinišemo metod kada nam treba drugačije ponašanje

```
class A {
protected:
    int x;
public:
   void ispis() {
        cout << "A x: " << endl;
   }
};

class B : public A {
private:
    int y;
public:
   void ispis() {
        cout << "B x: " << x << endl;
        cout << "B y: " << y << endl;
   }
};
```

```
int main()
{
    A a1, a2;
    B b1, b2;

    a1 = b2;

    a1.ispis();
    b1.ispis();

    return 0;
}
```

?



```
class A {
public:
   metod1
   metod2
};

class B : public A {
public:
   //metod1  preuzima ponašanje
   metod2    // redefiniše ponašanje
   metod3    // uvodi novo ponašanje

};
```

```
class A {
protected:
    int x;
public:
   int x;

   void inc() {
      x++;
   }
   void ispis() {
        cout << "A x: " << endl;
   }
};

class B : public A {
public:

   void dec() {
      x--;
   }
   void ispis() {
        cout << "B x: " << x << endl;
        cout << "B y: " << y << endl;
   }
};
```

inc - preuzeto
dec - novo
ispis - redefinisano

```
class A {
protected:
    int x;
public:
    A() { ... }
    A(int xx) { ... }
    A(const A &a) {... };
};

class B : public A {
public:
    B() { ... }
    B() { ... }
    B(const B &b) { ... }

};
```

```
class A {
protected:
    int x;
public:
    A() { x = 0; }
    A(int xx) { x = xx; }
    A(const A &a) { x = a.x; }
};

class B : public A {
public:
    // konstruktori
};
```

```
class A {
protected:
    int x;
public:
    A() { x = 0; }
    A(int xx) { x = xx; }
    A(const A &a) { x = a.x; }
};

class B : public A {
public:
    B() { ... }
    B(int xx) { ... }
    B(const B &b) { ... }
};
```

?

Šta rade ctors u B?
Da li B treba da inicijalizuje x?
Da li B treba da ima ctor sa parametrima?

Kada i kojim redom se pozivaju konstruktori?

```
class A {
protected:
    int x;
public:
    A() {
        x = 0;
        cout << "A K1" << endl;
    }
    A(int xx) {
        x = xx;
        cout << "A K2" << endl;
    }
    A(const A &a) {
        x = a.x;
        cout << "A K3" << endl;
    }
};

class B : public A {
public:
    B() {
        cout << "B K1" << endl;
    }
    B(int xx) {
        cout << "B K2" << endl;
    }
    B(const B &b) {
        cout << "B K3" << endl;
    }
};
```

```
class A {
protected:
    int x;
public:
    A() {
        x = 0;
        cout << "A K1" << endl;
    }
    A(int xx) {
        x = xx;
        cout << "A K2" << endl;
    }
    A(const A &a) {
        x = a.x;
        cout << "A K3" << endl;
    }
};

class B : public A {
public:
    B() {
        cout << "B K1" << endl;
    }
    B(int xx) {
        cout << "B K2" << endl;
    }
    B(const B &b) {
        cout << "B K3" << endl;
    }
};
```

```
int main()
{
    A a1;
    A a2(5);
    A a3(a1);

    return 0;
}
```

```
A K1
A K2
A K3
```

```
class A {
protected:
    int x;
public:
    A() {
        x = 0;
        cout << "A K1" << endl;
    }
    A(int xx) {
        x = xx;
        cout << "A K2" << endl;
    }
    A(const A &a) {
        x = a.x;
        cout << "A K3" << endl;
    }
};

class B : public A {
public:
    B() {
        cout << "B K1" << endl;
    }
    B(int xx) {
        cout << "B K2" << endl;
    }
    B(const B &b) {
        cout << "B K3" << endl;
    }
};
```

```
int main()
{
    B b1;

    return 0;
}
```

?

```
class A {
protected:
    int x;
public:
    A() {
        x = 0;
        cout << "A K1" << endl;
    }
    A(int xx) {
        x = xx;
        cout << "A K2" << endl;
    }
    A(const A &a) {
        x = a.x;
        cout << "A K3" << endl;
    }
};

class B : public A {
public:
    B() {
        cout << "B K1" << endl;
    }
    B(int xx) {
        cout << "B K2" << endl;
    }
    B(const B &b) {
        cout << "B K3" << endl;
    }
};
```

```
int main()
{
    B b1;

    return 0;
}
```

```
A K1
B K1
```

Izgrađuje se i roditeljski deo i tom prilikom se izvršava (neki) konstruktor nadklase

```
class A {
protected:
    int x;
public:
    A() {
        x = 0;
        cout << "A K1" << endl;
    }
    A(int xx) {
        x = xx;
        cout << "A K2" << endl;
    }
    A(const A &a) {
        x = a.x;
        cout << "A K3" << endl;
    }
};

class B : public A {
public:
    B() {
        cout << "B K1" << endl;
    }
    B(int xx) {
        cout << "B K2" << endl;
    }
    B(const B &b) {
        cout << "B K3" << endl;
    }
};
```

```
int main()
{
    B b1;
    B b2(3);
    B b3(b1);

    return 0;
}
```

?

```
class A {
protected:
    int x;
public:
    A() {
        x = 0;
        cout << "A K1" << endl;
    }
    A(int xx) {
        x = xx;
        cout << "A K2" << endl;
    }
    A(const A &a) {
        x = a.x;
        cout << "A K3" << endl;
    }
};

class B : public A {
public:
    B() {
        cout << "B K1" << endl;
    }
    B(int xx) {
        cout << "B K2" << endl;
    }
    B(const B &b) {
        cout << "B K3" << endl;
    }
};
```

```
int main()
{
    B b1;
    B b2(3);
    B b3(b1);

    return 0;
}
```

?

```
A K1
B K1
A K1
B K2
A K1
B K3
```

```
class A {
protected:
    int x;
public:
    A() {
        x = 0;
        cout << "A K1" << endl;
    }
    A(int xx) {
        x = xx;
        cout << "A K2" << endl;
    }
    A(const A &a) {
        x = a.x;
        cout << "A K3" << endl;
    }
};

class B : public A {
public:
    B() {
        cout << "B K1" << endl;
    }
    B(int xx) {
        cout << "B K2" << endl;
    }
    B(const B &b) {
        cout << "B K3" << endl;
    }
};
```

```
int main()
{
    B b1;
    B b2(3);
    B b3(b1);

    return 0;
}
```

?

```
A K1
B K1
A K1
B K2
A K1
B K3
```

```
class A {
protected:
    int x;
public:
    A() {
        x = 0;
        cout << "A K1" << endl;
    }
    A(int xx) {
        x = xx;
        cout << "A K2" << endl;
    }
    A(const A &a) {
        x = a.x;
        cout << "A K3" << endl;
    }
};

class B : public A {
public:
    B() {
        cout << "B K1" << endl;
    }
    B(int xx) {
        cout << "B K2" << endl;
    }
    B(const B &b) {
        cout << "B K3" << endl;
    }
};
```

```
int main()
{
    B b1;
    B b2(3);
    B b3(b1);

    return 0;
}
```

```
class B : public A {
public:
    B() : A(0) {
        cout << "B K1" << endl;
    }
    B(int xx) : A(xx){
        cout << "B K2" << endl;
    }
    B(const B &b) : A(b) {
        cout << "B K3" << endl;
    }
};
```

```
class A {
protected:
    int x;
public:
    A() {
        x = 0;
        cout << "A K1" << endl;
    }
    A(int xx) {
        x = xx;
        cout << "A K2" << endl;
    }
    A(const A &a) {
        x = a.x;
        cout << "A K3" << endl;
    }
};

class B : public A {
public:
    B() : A(0) {
        cout << "B K1" << endl;
    }
    B(int xx) : A(xx){
        cout << "B K2" << endl;
    }
    B(const B &b) : A(b) {
        cout << "B K3" << endl;
    }
};
```

```
int main()
{
    B b1;
    B b2(3);
    B b3(b1);

    return 0;
}
```

?

```
class A {
protected:
    int x;
public:
    A() {
        x = 0;
        cout << "A K1" << endl;
    }
    A(int xx) {
        x = xx;
        cout << "A K2" << endl;
    }
    A(const A &a) {
        x = a.x;
        cout << "A K3" << endl;
    }
};

class B : public A {
public:
    B() : A(0) {
        cout << "B K1" << endl;
    }
    B(int xx) : A(xx){
        cout << "B K2" << endl;
    }
    B(const B &b) : A(b) {
        cout << "B K3" << endl;
    }
};
```

```
int main()
{
    B b1;
    B b2(3);
    B b3(b1);

    return 0;
}
```

```
A K2
B K1
A K2
B K2
A K3
B K3
```

```
class A {
protected:
    int x;
public:
    A() {
        x = 0;
        cout << "A K1" << endl;
    }
    A(int xx) {
        x = xx;
        cout << "A K2" << endl;
    }
    A(const A &a) {
        x = a.x;
        cout << "A K3" << endl;
    }
};

class B : public A {
public:
    B() : A(0) {
        cout << "B K1" << endl;
    }
    B(int xx) : A(xx){
        cout << "B K2" << endl;
    }
    B(const B &b) : A(b) {
        cout << "B K3" << endl;
    }
};
```

```
int main()
{
    B b1;
    B b2(3);
    B b3(b1);

    return 0;
}
```

```
A K2
B K1
A K2
B K2
A K3
B K3
```

```
class B : public A {
public:
    B() {
        cout << "B K1" << endl;
    }
    B(int xx) {
        cout << "B K2" << endl;
    }
    B(const B &b) {
        cout << "B K3" << endl;
    }
};
```

```
class B : public A {
public:
    B() {
        cout << "B K1" << endl;
    }
    B(int xx) : A(xx){
        cout << "B K2" << endl;
    }
    B(const B &b) : A(b) {
        cout << "B K3" << endl;
    }
};
```

```
class B : public A {
public:
    B() : A(1) {
        cout << "B K1" << endl;
    }
    B(int xx) : A(xx){
        cout << "B K2" << endl;
    }
    B(const B &b) : A(b) {
        cout << "B K3" << endl;
    }
};
```

```
class B : public A {
public:
    B() : A(1) {
        cout << "B K1" << endl;
    }
    B(int xx) : A(xx + 1){
        cout << "B K2" << endl;
    }
    B(const B &b) : A(b) {
        cout << "B K3" << endl;
    }
};
```

```
class B : public A {
public:
    B() : A(1) {
        cout << "B K1" << endl;
    }
    B(int xx) {
        cout << "B K2" << endl;
    }
    B(const B &b) : A(b) {
        cout << "B K3" << endl;
    }
};
```

...

```
class A {
protected:
    int x;
public:
    ...
};

class B : public A {
public:
    B() : A(0) {
        cout << "B K1" << endl;
    }

    B(int xx) : A(xx){
        cout << "B K2" << endl;
    }

    B(const B &b) : A(b) {
        cout << "B K3" << endl;
    }

};
```

```
class A {
protected:
    int x;
public:
    ...
};

class B : public A {
public:
    B() {
        x == 0;
        cout << "B K1" << endl;
    }
    B(int xx) {
        x = xx;
        cout << "B K2" << endl;
    }
    B(const B &b) {
        x = b.x
        cout << "B K3" << endl;
    }
};
```

```
class A {
private:
    int x;
public:
    ...
};

class B : public A {
public:
    B() : A(0) {
        cout << "B K1" << endl;
    }

    B(int xx) : A(xx){
        cout << "B K2" << endl;
    }

    B(const B &b) : A(b) {
        cout << "B K3" << endl;
    }

};
```

```
class A {
private:
    int x;
public:
    ...
};

class B : public A {
public:
    B() {
        x = 0;
        cout << "B K1" << endl;
    }
    B(int xx) {
        x = xx;
        cout << "B K2" << endl;
    }
    B(const B &b) {
        x = b.x
        cout << "B K3" << endl;
    }
};
```

```
class A {
private:
    int x;
public:
    ...
};

class B : public A {
private:
    int y;
public:
    B() : A(0){
        y = 0;
        cout << "B K1" << endl;
    }
    B(int xx, int yy) : A(xx){
        y = yy;
        cout << "B K2" << endl;
    }
    B(const B &b) : A(b){
        y = b.y;
        cout << "B K3" << endl;
    }
};
```

```
class A {
private:
    int x;
public:
    ...
};

class B : public A {
private:
    int y;
public:
    B() : A(0){
        y = 0;
        cout << "B K1" << endl;
    }
    B(int xx, int yy) : A(xx){
        y = yy;
        cout << "B K2" << endl;
    }
    B(const B &b) : A(b){
        y = b.y;
        cout << "B K3" << endl;
    }
};
```

```
class A {
private:
    int x;
public:
    ...
};

class B : public A {
private:
    int y;
public:
    B() : A(0), y(0){
        cout << "B K1" << endl;
    }

    B(int xx, int yy) : A(xx), y(yy){
        cout << "B K2" << endl;
    }

    B(const B &b) : A(b), y(b.y){
        cout << "B K3" << endl;
    }

};
```

```
class Osoba {
private:
   DinString ime, prezime;
public:
   // Osoba … konstruktori

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

```
class Osoba {
private:
   DinString ime, prezime;
public:
   // Osoba ... konstruktori

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

```
class Student : public Osoba {
private:
   int brojIndeksa;
public:
   // Student ... konstruktori

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

```
class Osoba {
private:
   DinString ime, prezime;
public:
   // Osoba ... konstruktori

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

```
class Student : public Osoba {
private:
   int brojIndeksa;
public:
   // Student ... konstruktori

   void predstaviSe() const {
      cout << "Zovem se " << ime << " " << prezime << "." << endl;
      cout << "Broj mog indeksa je " << brojIndeksa << "." << endl;
   }
};
```

```
class Osoba {
private:
   DinString ime, prezime;
public:
   // Osoba ... konstruktori

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

```
class Student : public Osoba {
private:
   int brojIndeksa;
public:
   // Student ... konstruktori

   void predstaviSe() const {
      cout << "Zovem se " << ime << " " << prezime << "." << endl;
      cout << "Broj mog indeksa je " << brojIndeksa << "." << endl;
   }
};
```

```
class Osoba {
private:
   DinString ime, prezime;
public:
   // Osoba ... konstruktori

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

```
class Student : public Osoba {
private:
   int brojIndeksa;
public:
   // Student ... konstruktori

   void predstaviSe() const {
      Osoba::predstaviSe();
      cout << "Broj mog indeksa je " << brojIndeksa << "." << endl;
   }
};
```

```
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 Student : public Osoba {
private:
   int brojIndeksa;
public:
public:
   Student(const char *s1="", const char *s2="", int i=0) : Osoba(s1,s2), brojIndeksa(i) {}
   Student(const DinString &ds1, const DinString &ds2, int i) : Osoba(ds1,ds2), brojIndeksa(i) {}
   Student(const Osoba &os, int i) : Osoba(os), brojIndeksa(i){}
   Student(const Student &s) : Osoba(s), brojIndeksa(s.brojIndeksa) {}

   void predstaviSe() const {
      Osoba::predstaviSe();
      cout << "Broj mog indeksa je " << brojIndeksa << "." << endl;
   }
};
```

```
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 Student : public Osoba {
private:
   int brojIndeksa;
public:
public:
   Student(const char *s1="", const char *s2="", int i=0) : Osoba(s1,s2), brojIndeksa(i) {}
   Student(const DinString &ds1, const DinString &ds2, int i) : Osoba(ds1,ds2), brojIndeksa(i) {}
   Student(const Osoba &os, int i) : Osoba(os), brojIndeksa(i){}
   Student(const Student &s) : Osoba(s), brojIndeksa(s.brojIndeksa) {}

   void predstaviSe() const {
      Osoba::predstaviSe();
      cout << "Broj mog indeksa je " << brojIndeksa << "." << endl;
   }
};
```

```
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 Student : public Osoba {
private:
   int brojIndeksa;
public:
public:
   Student(const char *s1="", const char *s2="", int i=0) : Osoba(s1,s2), brojIndeksa(i) {}
   Student(const DinString &ds1, const DinString &ds2, int i) : Osoba(ds1,ds2), brojIndeksa(i) {}
   Student(const Osoba &os, int i) : Osoba(os), brojIndeksa(i){}
   Student(const Student &s) : Osoba(s), brojIndeksa(s.brojIndeksa) {}

   void predstaviSe() const {
      Osoba::predstaviSe();
      cout << "Broj mog indeksa je " << brojIndeksa << "." << endl;
   }
};
```

```
class Trougao {
    // a, b, c

    // izracunaj P i O
};

class JKTrougao : public Trougao {
   // a, b

   // izracunaj P i O
};
```

Dve strane su jednake u jednakokrakom trouglu - ne treba da pamtimo i vodimo evidenciju o dva identična polja - dovoljno i poželjno je jedno

Formule za obim i površinu važe kod svih trouglova
Ako imamo implementirano u nadklasi, podklasa može da koristi

```
class Trougao {
private:
    int a, b, c;

    // izracunaj P i O
};

class JKTrougao : public Trougao {
   // a, b

   // izracunaj P i O
};
```

```
class Trougao {
private:
    int a, b, c;
public:
    Trougao() { a = 3; b = 4; c = 5; }
    Trougao(double aa, double bb, double cc) : a(aa), b(bb), c(cc) { }
    Trougao(const Trougao &t) : a(t.a), b(t.b), c(t.c) { }

    // izracunaj P i O
};

class JKTrougao : public Trougao {
   // a, b

   // izracunaj P i O
};
```

```
class Trougao {
private:
    int a, b, c;
public:
    Trougao() { a = 3; b = 4; c = 5; }
    Trougao(double aa, double bb, double cc) : a(aa), b(bb), c(cc) { }
    Trougao(const Trougao &t) : a(t.a), b(t.b), c(t.c) { }

    // izracunaj P i O
};

class JKTrougao : public Trougao {
   // a, b

   // izracunaj P i O
};
```

```
class Trougao {
private:
    int a, b, c;
public:
    Trougao() { a = 3; b = 4; c = 5; }
    Trougao(double aa, double bb, double cc) : a(aa), b(bb), c(cc) { }
    Trougao(const Trougao &t) : a(t.a), b(t.b), c(t.c) { }

    // getA, getB, getC ?
    // setA, setB, setC ?

    // izracunaj P i O
};

class JKTrougao : public Trougao {
   // a, b

   // izracunaj P i O
};
```

```
class Trougao {
private:
    int a, b, c;
public:
    Trougao() { a = 3; b = 4; c = 5; }
    Trougao(double aa, double bb, double cc) : a(aa), b(bb), c(cc) { }
    Trougao(const Trougao &t) : a(t.a), b(t.b), c(t.c) { }

    double getO() const { return a+b+c; }
    double getP() const { double s=(a+b+c)/2; return sqrt(s*(s-a)*(s-b)*(s-c)); }

};

class JKTrougao : public Trougao {
   // a, b

   // izracunaj P i O
};
```

```
class Trougao {
private:
    int a, b, c;
public:
    Trougao() { a = 3; b = 4; c = 5; }
    Trougao(double aa, double bb, double cc) : a(aa), b(bb), c(cc) { }
    Trougao(const Trougao &t) : a(t.a), b(t.b), c(t.c) { }

    double getO() const { return a+b+c; }
    double getP() const { double s=(a+b+c)/2; return sqrt(s*(s-a)*(s-b)*(s-c)); }

};

class JKTrougao : public Trougao {
    // nema polja

public:
    JKTrougao() : Trougao(1, 2, 2) {}
    JKTrougao(double aa, double bb) : Trougao(aa, bb, bb) {}
    JKTrougao(const JKTrougao &jkt) : Trougao(jkt) P{}

   // izracunaj P i O -> koristi nasleđeno
};
```

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { cout << "AK2" << endl; }
    A(const A &a) : x(a.x) { cout << "AK3" << endl; }

   ~A() {  cout << "AD" << endl; }
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){ cout << "BK1" << endl; }
    B(int xx, int yy) : A(xx), y(yy){ cout << "BK2" << endl; }
    B(const B &b) : A(b), y(b.y){ cout << "BK3" << endl; }

    ~B() { cout << "B D" << endl; }
};
```

```
int main()
{
    B b1, b2(3, 5), b3(b2);

    return 0;
}
```

?

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { cout << "AK2" << endl; }
    A(const A &a) : x(a.x) { cout << "AK3" << endl; }

   ~A() {  cout << "AD" << endl; }
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){ cout << "BK1" << endl; }
    B(int xx, int yy) : A(xx), y(yy){ cout << "BK2" << endl; }
    B(const B &b) : A(b), y(b.y){ cout << "BK3" << endl; }

    ~B() { cout << "B D" << endl; }
};
```

```
int main()
{
    B b1, b2(3, 5), b3(b2);

    return 0;
}
```

```
A K2
B K1
A K2
B K2
A K3
B K3
B D
A D
B D
A D
B D
A D
```

Prvo se oslobađa deo koji se odnosi na B pa tek onda deo koji se odnosi na A

Polimorfizam

- poly, morph - više oblika, više ponašanja

- preklapanje imena funkcija (function overloading)
- preklapanje operatora (operator overloading)

- redefinisanje metode u podklasi (method overriding)
- + virtualnost

C++ podržava višestruko nasleđivanje

```
class A : B, C
```

ZADATAK

(neobavezno)

Dodati setIme u klasu Osoba. (`void setIme(const char *);`)

Napraviti dva objekta klase Osoba. Jedan uz pomoć K2, a drugi kao kopija prvog uz pomoć K3.

Promeniti ime prvom objektu. Ispisati oba i videti rezultat.

Izbaciti konstruktor kopije za DinString. Posmatrati sada i uočiti razliku.

Šta je operatorom dodele iz DinString klase? Pokušati sa izbacivanjem. Posmatrati različite slučajeve (ima/nema =, k3,...).