# Napredno programiranje i programski jezici

## 05 C++ (virtualnost)

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

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

    void ispis() const { cout << "A: x = " << x << endl; }
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
    A a(1);
    B b(3, 5);
    a.ispis();
    b.ispis();

    return 0;
}
```

?

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

    void ispis() const { cout << "A: x = " << x << endl; }
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
    A a(1);
    B b(3, 5);
    a.ispis();
    b.ispis();

    a = b;
    b = a;
    a.ispis();
    b.ispis();

    return 0;
}
```

?

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

    void ispis() const { cout << "A: x = " << x << endl; }
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
    A a(1);
    B b(3, 5);
    a.ispis();
    b.ispis();

    a = b;
    a.ispis();
    b.ispis();

    A &aref = b;
    aref.ispis();
    b.ispis();

    return 0;
}
```

?

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

    void ispis() const { cout << "A: x = " << x << endl; }
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
    A a(1);
    B b(3, 5);
    a.ispis();
    b.ispis();

    a = b;
    a.ispis();
    b.ispis();

    A &aref = b;
    aref.ispis();
    b.ispis();

    return 0;
}
```

```
A: x = 1
B: x = 3 y = 5

A: x = 3
B: x = 3 y = 5

A: x = 3
B: x = 3 y = 5
```

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

    virtual void ispis() const { cout << "A: x = " << x << endl; }
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
    A a(1);
    B b(3, 5);
    a.ispis();
    b.ispis();

    a = b;
    a.ispis();
    b.ispis();

    A &aref = b;
    aref.ispis();
    b.ispis();

    return 0;
}
```

```
A: x = 1
B: x = 3 y = 5


A: x = 3
B: x = 3 y = 5


B: x = 3 y = 5
B: x = 3 y = 5
```

```
int main()
{
    A a(1);
    B b(3, 5);
    a.ispis();
    b.ispis();

    a = b;
    a.ispis();
    b.ispis();

    A &aref = b;
    a.ispis();
    b.ispis();

    return 0;
}
```

```
class A
    void ispis()

class B : public A
    void ispis()
```

```
A: x = 1
B: x = 3 y = 5

A: x = 3
B: x = 3 y = 5

A: x = 3
B: x = 3 y = 5
```

```
class A
    virtual void ispis()

class B : public A
    void ispis()
```

```
A: x = 1
B: x = 3 y = 5

A: x = 3
B: x = 3 y = 5

B: x = 3 y = 5
B: x = 3 y = 5
```

*Kasnije se poveže sa metodom koja treba da se pozove*

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

    virtual void ispis() const { cout << "A: x = " << x << endl; }
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
    A a(1);
    B b(3, 5);
    a.ispis();
    b.ispis();

    a = b;
    a.ispis();
    b.ispis();

    A &aref = b;
    aref.ispis();
    b.ispis();

    A *apok = &b;
    apok -> ispis();
    b.ispis();

    return 0;
}
```

?

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

    virtual void ispis() const { cout << "A: x = " << x << endl; }
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
    A a(1);
    B b(3, 5);
    a.ispis();
    b.ispis();

    a = b;
    a.ispis();
    b.ispis();

    A &aref = b;
    aref.ispis();
    b.ispis();

    A *apok = &b;
    apok -> ispis();
    b.ispis();

    return 0;
}
```

```
A: x = 1
B: x = 3 y = 5


A: x = 3
B: x = 3 y = 5


B: x = 3 y = 5
B: x = 3 y = 5


B: x = 3 y = 5
B: x = 3 y = 5
```

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

    virtual void ispis() const { cout << "A: x = " << x << endl; }
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
    A a(1);
    B b(3, 5);
    a.ispis();
    b.ispis();

    a = b;
    a.ispis();
    b.ispis();

    A &aref = b;
    aref.ispis();
    b.ispis();

    A *apok = &b;
    apok -> ispis();
    b.ispis();

    return 0;
}
```

```
A: x = 1
B: x = 3 y = 5
A: x = 3
B: x = 3 y = 5
A: x = 3
B: x = 3 y = 5
A: x = 3
B: x = 3 y = 5
```

Paziti!

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

    virtual void ispis() const { cout << "A: x = " << x << endl; }
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
void ispis(A a) {
    a.ispis();
}

void ispisRef(A &a) {
    a.ispis();
}

void ispisPok(A *a) {
    a -> ispis();
}


int main()
{
    A a(1);
    B b(3, 5);

    ispis(b);
    ispisRef(b);
    ispisPok(&b);
}
```

?

```
Stampaj (const Osoba&);

svima na fakultetu
PosaljiCestitkuZaNovuGodinu(const Osoba&);

za svakog studenta (bsc, msc, phd)
OtvoriNalogNaCanvasu(const Student&);


Ucionica: studenti + nastavnici + admini
Lista Osoba

...
```

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

    virtual void ispis() const = 0;
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

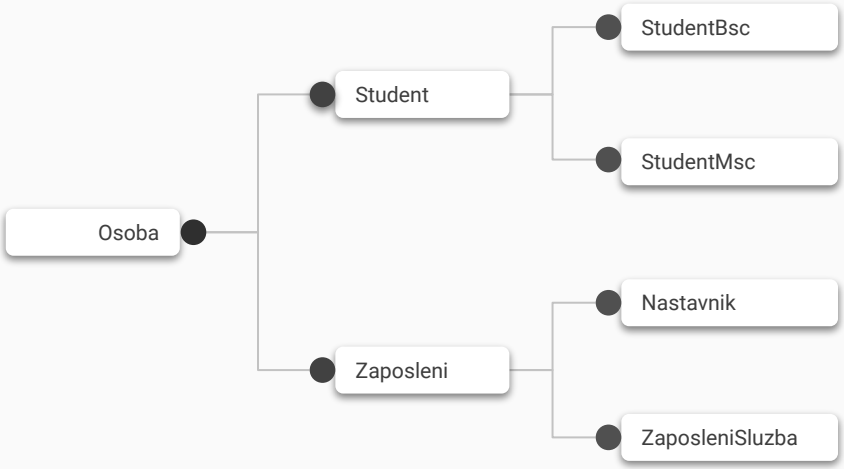
pure virtual methods
apstraktne metode

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

    virtual void ispis() const = 0;
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
  A a(1);
    B b(3, 5);

    a.ispis();
    b.ispis();
}
```

Apstraktna klasa - ona koja ima bar jednu ovakvu, apstraktnu, metodu

Ne može se instancirati!




```
Stampaj (const Osoba&);
PosaljiCestitkuZaNovuGodinu(const Osoba&);
OtvoriNalogNaCanvasu(const Student&);
Lista Osoba
...
```

Problem: Realizovati aplikaciju za rad sa geometrijskim figurama.

Model (klase) ?

```
class Figura { };

class Krug : public Figura { };

class Pravougaonik : public Figura { };
class Kvadrat: public Pravougaonik { };

class Trougao : public Figura { };
class JKTrougao : public Trougao { };
class JSTrougao : public JKTrougao { };
```

```
class Figura {
    // polja ?

    double getO() const {...}
    double getP() const {...}
};

class Krug : public Figura { };

class Pravougaonik : public Figura { };
class Kvadrat: public Pravougaonik { };

class Trougao : public Figura { };
class JKTrougao : public Trougao { };
class JSTrougao : public JKTrougao { };
```

?

```
class Figura {
    virtual double getO() const = 0;
    virtual double getP() const = 0;
};

class Krug : public Figura { };

class Pravougaonik : public Figura { };
class Kvadrat: public Pravougaonik { };

class Trougao : public Figura { };
class JKTrougao : public Trougao { };
class JSTrougao : public JKTrougao { };
```

```
class Figura {
    virtual double getO() const = 0;
    virtual double getP() const = 0;
};

class Krug : public Figura { };

class Pravougaonik : public Figura { };
class Kvadrat: public Pravougaonik { };

class Trougao : public Figura { };
class JKTrougao : public Trougao { };
class JSTrougao : public JKTrougao { };
```

```
int main()
{
    Figura f;
}
```

```
error: cannot declare variable 'f'
to be of abstract type 'Figura'
```

```
class Figura {
    virtual double getO() const = 0;
    virtual double getP() const = 0;
};

class Krug : public Figura { };

class Pravougaonik : public Figura { };
class Kvadrat: public Pravougaonik { };

class Trougao : public Figura { };
class JKTrougao : public Trougao { };
class JSTrougao : public JKTrougao { };
```

Ako proglasim metodu za apstraktnu šta se dešava sa podklasama?

Da li se mogu instancirati?

?

```
class Figura {
    virtual double getO() const = 0;
    virtual double getP() const = 0;
};

class Krug : public Figura { };

class Pravougaonik : public Figura { };
class Kvadrat: public Pravougaonik { };

class Trougao : public Figura { };
class JKTrougao : public Trougao { };
class JSTrougao : public JKTrougao { };
```

Ako proglasim metodu za apstraktnu šta se dešava sa podklasama?

Da li se mogu instancirati?

```
error: cannot declare variable 'k' to
be of abstract type 'Krug'
```

```
class Figura {
    virtual double getO() const = 0;
    virtual double getP() const = 0;
};

class Krug : public Figura {
    double getO() const { ... }
    double getP() const { ... }
};

class Pravougaonik : public Figura { ... };
class Kvadrat: public Pravougaonik { ... };

class Trougao : public Figura { ... };
class JKTrougao : public Trougao { ... };
class JSTrougao : public JKTrougao { ... };
```

```
class A {
    virtual ... metod1... = 0;
    virtual ... metod2... = 0;
    virtual ... metod3... = 0;
    virtual ... metod4... = 0;
    virtual ... metod5... = 0;
    virtual ... metod6... = 0;
   ...
}
```

interfejs

Još par napomena...

```
class A {
protected:
    int x;
public:
    A() : x(0) {  }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
}
```

?

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
}
```

```
AK1
AK1
AK1
AK1
AK1
```

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
    niz[2].ispis();
}
```

?

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
    niz[2].ispis();
}
```

```
AK1
AK1
AK1
AK1
AK1
A: x = 0
```

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
    niz[2].ispis();

    B b(3, 5);
    niz[2] = b;

    b.ispis();
    niz[2].ispis();
}
```

?

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
    niz[2].ispis();

    B b(3, 5);
    niz[2] = b;

    b.ispis();
    niz[2].ispis();
}
```

```
AK1
AK1
AK1
AK1
AK1
A: x = 0
B: x = 3 y = 5
A: x = 3
```

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
    niz[2].ispis();

    B b(3, 5);
    niz[2] = b;

    b.ispis();
    niz[2].ispis();

    cout << "-------" << endl;
    A* nizPok[5];

}
```

?

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
    niz[2].ispis();

    B b(3, 5);
    niz[2] = b;

    b.ispis();
    niz[2].ispis();

    cout << "-----
    A* nizPok[5];

}
```

```
AK1
AK1
AK1
AK1
AK1
A: x = 0
B: x = 3 y = 5
A: x = 3
-------
```

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
    niz[2].ispis();

    B b(3, 5);
    niz[2] = b;

    b.ispis();
    niz[2].ispis();

    cout << "-------" << endl;
    A* nizPok[5];
    nizPok[2] -> ispis();

}
```

Opasno, paziti!

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
    niz[2].ispis();

    B b(3, 5);
    niz[2] = b;

    b.ispis();
    niz[2].ispis();

    cout << "-------" << endl;
    A* nizPok[5];
    nizPok[2] -> ispis();

    nizPok[2] = &b;

    b.ispis();
    nizPok[2] -> ispis();
}
```

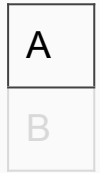
?

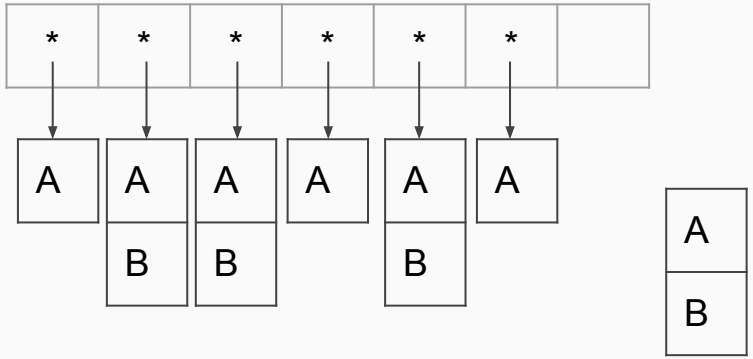
```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
    niz[2].ispis();

    B b(3, 5);
    niz[2] = b;

    b.ispis();
    niz[2].ispis();

    cout << "-------"
    A* nizPok[5];
    nizPok[2] -> ispis

    nizPok[2] = &b;

    b.ispis();
    nizPok[2] -> ispis
}
```

```
AK1
AK1
AK1
AK1
AK1
A: x = 0
B: x = 3 y = 5
A: x = 3
-------
B: x = 3 y = 5
B: x = 3 y = 5
```

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
    niz[2].ispis();

    B b(3, 5);
    niz[2] = b;

    b.ispis();
    niz[2].ispis();

    A* nizPok[5];
    nizPok[2] -> ispis();

    nizPok[2] = &b;

    b.ispis();
    nizPok[2] -> ispis();

    A& nizRef[5];
}
```

?

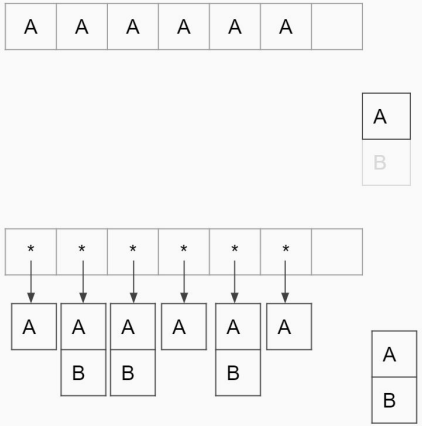
```
class A {
protected:
    int x;
public:
    A() : x(0) { cout << "AK1" << endl; }
    A(int xx) : x(xx) { }
    A(const A &a) : x(a.x) { }
    virtual void ispis() const {
      cout << "A: x = " << x << endl;
    }
};
class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }
    void ispis() const {
       cout << "B: x = " << x << " y = " << y << endl;
    }
};
```

```
int main()
{
    A niz[5];
    niz[2].ispis();

    B b(3, 5);
    niz[2] = b;

    b.ispis();
    niz[2].ispis();

    A* nizPok[5];
    nizPok[2] -> ispis();

    nizPok[2] = &b;

    b.ispis();
    nizPok[2] -> ispis();

    A& nizRef[5];
}
```

Nizovi (vrednost, pokazivač)
Parametri metoda (vrednost, pokazivač, referenca)

Obratiti pažnju koje je željeno ponašanje

virtualni destruktori
set metode
...

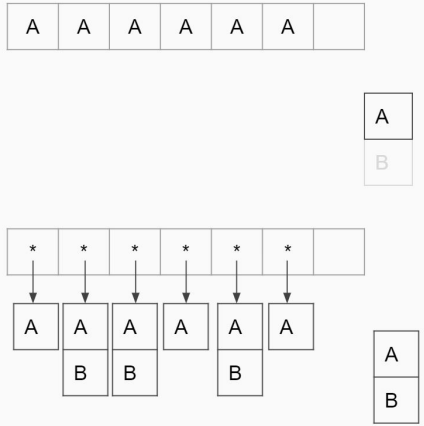


Dinamička alokacija memorije
- konstruktor kopije
- = operator
- destruktor

PP da nasleđena klasa ima dinamičku alokaciju
(ili pristupa nekim spoljnim resursima ili obavlja bilo kakvu drugu "kritičnu" funkcionalnost)

Šta ako se pozove destruktor nadklase (`A* a = new B()`)?
memory leak
(ili nerešena kritična situacija)

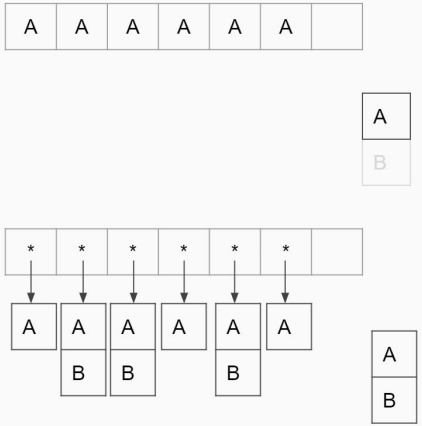
→ virtualni destruktor



# Nasleđena klasa - specijalni slučaj + ograničenja

```
class Trougao {
private:
    int a, b, c;
public:
    konstruktori
    get
    set
};

class JKtrougao : Trougao{
public:
    konstruktori -> ogranicenja
    set -> ogranicenja
};
```

```
class Pravougaonik {
private:
    int a, b;
public:
    konstruktori
    get
    set
};

class Kvadrat : Pravougaonik{
public:
    konstruktori -> ogranicenja
    set -> ogranicenja
};
```




→ virtualni set

set (i get) metode su najobičnije metode, kao i sve druge

Za sve metode treba razmotriti koje je ponašanje poželjno
Uglavnom je, kod nasleđivanja, poželjno da budu virtual
Nasleđivanje je često iz apstraktne klase gde je svakako bar jedan metod virtual

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

    virtual void ispis() const = 0;
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
  A a(1);
    B b(3, 5);

    a.ispis();
    b.ispis();
}
```

pure virtual methods
apstraktne metode
apstraktne klase

*U prethodnoj pp fali const kod ispisa*

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

    virtual void ispis() const = 0;
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
  A a(1);
    B b(3, 5);

    a.ispis();
    b.ispis();
}
```

Ako ne možemo da instanciramo zašto imamo konstruktore?

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

    virtual void ispis() const = 0;
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
void ispisA (A a) { ... }
void ispisB (B b) { ... }
void ispisARef (const A& a) { ... }
void ispisBRef (const B& b) { ... }
void ispisAPok (A* a) { ... }
void ispisBPok (B* b) { ... }


int main()
{
   A a(1);
    B b(3, 5);

    a.ispis();
    b.ispis();
}
```

Šta ne može?

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

    virtual void ispis() const = 0;
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
void ispisA (A a) { ... }
void ispisB (B b) { ... }
void ispisARef (const A& a) { ... }
void ispisBRef (const B& b) { ... }
void ispisAPok (A* a) { ... }
void ispisBPok (B* b) { ... }


int main()
{
  A a(1);
    B b(3, 5);

    a.ispis();
    b.ispis();
}
```

```
error: cannot declare parameter 'a' to be of
abstract type 'A'
```

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

    virtual void ispis() const = 0;
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
  ~~A a(1);~~
    B b(3, 5);

    ~~a.ispis();~~
    b.ispis();

    A nizV[5];
    A* nizP[5];

}
```

Šta ne može?

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

    virtual void ispis() const = 0;
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
    A a(1);
    B b(3, 5);

    a.ispis();
    b.ispis();

    A nizV[5];
    A* nizP[5];

}
```

```
error: invalid abstract type 'A' for 'nizV'
```

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

    virtual void ispis() const = 0;
};

class B : public A {
private:
    int y;
public:
    B() : A(1), y(1){  }
    B(int xx, int yy) : A(xx), y(yy){  }
    B(const B &b) : A(b), y(b.y){  }

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

```
int main()
{
  ~~A a(1);~~
    B b(3, 5);

    ~~a.ispis();~~
    b.ispis();

    ~~A nizV[5];~~
    A* nizP[5];

    nizP[0] = new B(0, 10);
    nizP[1] = new B(1, 11);
    nizP[2] = new B(2, 12);
    nizP[3] = new B(3, 13);
    nizP[4] = new B(4, 14);

    for (int i = 0; i < 5; i++) {
       nizP[i] -> ispis();
    }
}
```

## ZADATAK

*(neobavezno)*

Napisati klase Pravougaonik i Kvadrat tako da Kvadrat nema svoja polja već preko konstruktora obezbeđuje validnost.
Posmatrati sve što smo obradilil. Na primer:

- Razmisliti o pristupu poljima za stranice u klasi Pravougaonik (protected i private?).
- Dodati neki metod (ispis) koji je virtualan.
- Napisati neke, slične, slobodne funkcije koje koriste virtualni metod u kojima se parametar tipa Pravougaonik prenosi preko vrednosti i preko reference. Testirati sa obe figure.
- Napraviti nizove pravougaonika (vrednost, pokazivač) i posmatrati ponašanja virtuelne metode kad se u nizove dodaju i kvadarti i pravougaonici.
- Dodati apstraktnu klasu Figura na vrh ove hijerarhije. Virtualna metoda za ispis može biti apstraktna.
- Napisati slobodne funkcije čiji parametri su Figure (vrednost, referenca). Napraviti nizove pokazivača. Prosleđivati pravougaonike i kvadrate, posmatrati.

Dodati set metode u Pravougaonik i Kvadrat. U Kvadratu obezbediti da stranice budu iste.
Posmatrati razliku ako set metoda u Pravougaoniku jeste ili nije virtualna.

Napisati kod koji dovodi do situacije da objekat tipa Kvadrat zapravo to nije tj ima dve različite stranice.