OOP KEELED

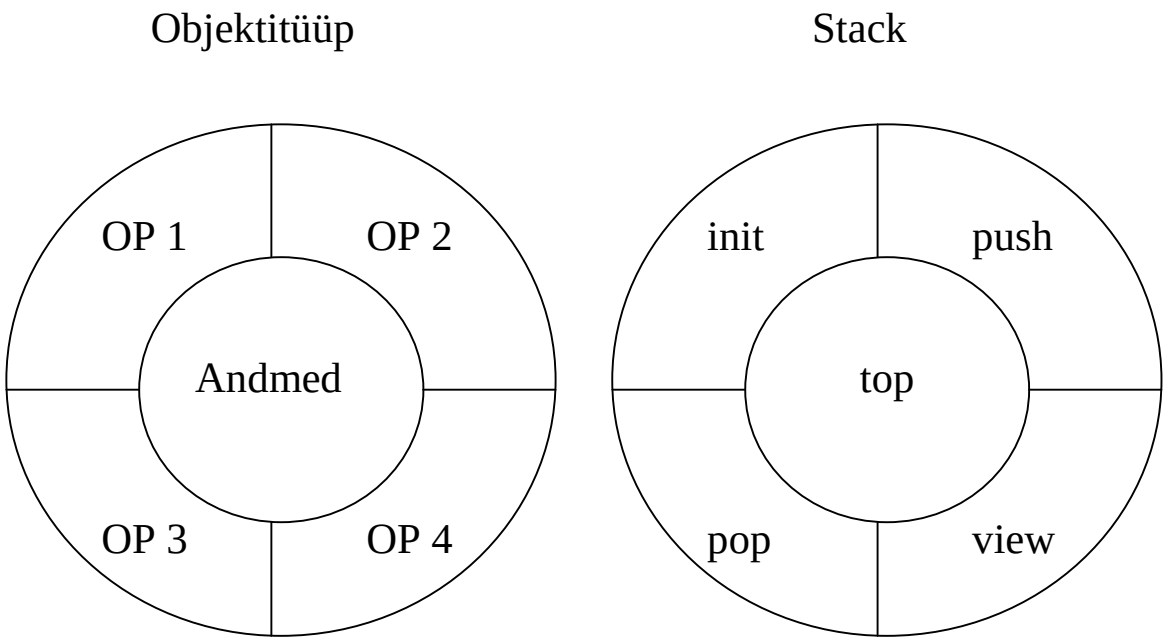
Kolm omadust:

1) **kapseldamine** (*encapsulation*) – keeles on vahendid, mis võimaldavad koondada andmed ja nendega tehtavad operatsioonid (meetodid) ühte programmilisse ühikusse (objektitüüp)

   NB! objekti andmetele saab reeglina ligi ainult meetodite kaudu!

2) **pärandamine** (*inheritance*) – igast objektitüübist saab tuletada uusi objektitüüpe, kusjuures järglased pärivad eellase omadused (nii andmed kui meetodid)

3) **polümorfism** (*polymorphism*) – samanimelised meetodid võivad erinevatel objektidel teostuda erineval viisil



Objektitüübid keeles C++

Kirjeldajad: **struct** **union** **class**

```
<kirjeldaja>  <objektitüübi_identifikaator>
{
        <muutujad_ja_meetodid_=_LIIKMED_(members)>
}
```

```
struct  Elem
{
        int  arv;
        Elem  *next;
}
```

```
class  Stack
{
private:
        Elem  *top;

public:
        void  init( void )  { top = NULL; }

        void  push( int a )
        {
                Elem *uus = new Elem;
                uus -> arv = a;
                uus -> next = top;
                top = uus;
        }
```

```
        int  pop( int &a)
        {
                if( !top )  return  0;      //  FALSE kui tühi
                Elem  *vana = top;
                a = top -> arv;
                top = top -> next;
                delete  vana;               // vabastada mälu
                return  1;                  // TRUE kui võetakse element
        }

        void view( void )
        {
                Elem *jooksev = top;
                int n = 0;
                while( jooksev )
                {
                        cout  <<  jooksev -> arv  <<  '\n';
                        n++;
                };
                cout  <<  "\n Kokku " <<  n  <<  " elementi\n";
        }
}

int  main( void )
{
        Stack  mag;                 // uus ilming
        mag.init();
        for( int i=0; i<10; i++)
                mag.push( i );
        mag.view();

        Stack pinu;                 // uus ilming
        pinu.init();
        while( mag.pop( int num ))
```

```
        pinu.push( num );
    pinu.view();

    pinu.top ->arv = 7;          // Viga!  pinu.top on privaatne!
    mag.top = NULL;              // Viga!  mag.top on privaatne!
    return  0;
}
```

Juurdepääsu määrajad: **public private protected**

1) klassi (objektitüübi) meetodites on juurdepääs **kõigile** selle klassi liikmetele

2) objektitüübi ilmingu kaudu on juurdepääs (liitnime abil) **ainult avalikele** liikmetele

Vaikimisi:

class	-	private
struct	-	public
union	-	public

Konstruktorid ja destruktorid

Konstruktor:

- liikmefunktsioon, mille nimi ühtib klassi nimega
- konstrutoril puudub väärtus (ka mitte *void !*)
- konstruktor käivitatakse objekti ilmingu deklareerimise hetkel (ja ainult siis! – liitnimega ei saa pöörduda!)
- konstruktoril võivad olla parameetrid

Näide:

muudame klassi **Stack** meetodi **init** konstruktoriks:

```
public:
    Stack ( void )   { top = NULL; }

                    // põhiprogrammis nüüd niimoodi:

    Stack mag;   // käivitub konstruktor!
```

Destruktorid:

- liikmefunktsioonid objekti ilmingu “hävitamiseks”
- nimi nagu konstrutoril, aga ees ~ (tilde)
- lubatud väljakutse liitnimega

- objekti ilmingule rakendub destruktor automaatselt, kui programm “väljub” plokist, kus see ilming on deklareeritud

Näide: klassi **Stack** destruktor:

```
public:
        ~Stack ( void )
        {
                int b;
                while( pop( b )) ;
        }


// nüüd võib põhiprogrammis teha nii:

        mag.~Stack()
```

Funktsiooni prototüüp

Funktsiooni pealdis, milles on ära näidatud:

- funktsiooni väärtuse tüüp
- funktsiooni nimi
- funktsiooni parameetrite tüüpide loetelu

Näiteks:

```
int midagi( float, char *, int );
```

Funktsiooni täielik kirjeldus antakse kuskil mujal

Ka objektitüübi kirjelduses võivad olla ainult liikmefunktsioonide prototüübid:

```
class Stack
{
        Elem *top;
public:
        Stack( void );
        void push( int );
        int pop( int & );
        void view( void );
        ~Stack( void );
}
```

// meetodite tegelikud kirjeldused tuleb nüüd anda kujul:

```
Stack :: Stack( void )  { top=NULL; }

void  Stack :: push( int a )  {
                                        // meetodi keha
                                }
```

inline liikmefunktsioonid

Liikmefunktsiooni (meetodi) deklaratsioon võib paikneda:

1) klassi deklaratsiooni “sees”

2) väljaspool klassi deklaratsiooni

Juhul 1 loetakse meetod **inline**-liikmefunktsiooniks

Juhul 2 muutub meetod **inline**-liikmefunktsiooniks ainult siis, kui meetodi deklaratsioon algab võtmesõnaga **inline**

Erinevus kajastub meetodi poole pöördumise realisatsioonis:

if *inline*

then *makrolaiendus*

else *call* <meetod>

Näide:

```
inline  int  Stack :: pop( int &a)
{
      if( !top )  return  0;      //  FALSE kui tühi
      Elem  *vana = top;
      a = top -> arv;
      top = top -> next;
      delete  vana;               // vabastada mälu
      return  1;                  // TRUE kui võetakse element
}
```

// siis pöörumine

mag.pop(int num)

// realiseeritakse pöördumisena “uue” funktsiooni poole:

```
int  mag213( void )          // mag213 – kompilaatori poolt
                             // genereeritud nimi
{
    if( !mag.top )  return  0;
    Elem  *vana = mag.top;
    num = mag.top -> arv;
    mag.top = mag.top -> next;
    delete  vana;
    return  1;
}
```

mag213 - sama, mis meetod Stack::pop, mille kehas kõik formaalsed parameetrid **on asendatud** faktiliste parameetritega

Viidad objektidele

```
Stack  mag, *p;          // p on viitmuutuja Stack-tüüpi objektile
p = &mag;                // nüüd p viitab objektile mag
p -> push( 17 );

Stack tosin[12];         // 12 Stack'i
tosin[4].push( 999 );
p = &tosin[7];
p = tosin;               //  sama mis  p = &tosin[0];
```

this

Võtmesõna **this** on klassi meetodites predefineeritud viitmuutja, mille väärtuseks on viit "jooksvale" objektile:

```
class Ilus
{
        int x;
public:
        Ilus( void )
          {
                this -> x = -1;          // sama mis    x = -1;
          }
        Ilus  *aadress( void )
          {
                return  this;            //  !!!
           }
}
```

Funktsioonide *overload*

Programmis (ka objektitüübis) võib olla mitu sama nimega funktsiooni (ka mitu konstruktorit):

- funktsioonide väärtuse tüüp peab olema sama
- kas erinev arv parameetreid
- või parameetritel erinevad tüübid

Näide:

```
class Midagi
{
public:
        Midagi( int );          //  1
        Midagi( char * );       //  2
        Midagi( int, int );     //  3
}

// kuskil mujal:

Midagi      k( 14, -3 ),        //  3
            l( 56740 ),         //  1
            m("tere");          //  2
```

Objektid ja operaatorid

Olgu antud objektitüüp:

```
class  Paar
{
        int  x, y;
public:
        Paar( int a, int b)  { x = a; y = b; }
        Paar( void )  {  x = 0; y = 0; }
        int  getX( void ) { return x; }
        int  getY( void ) { return y; }
        void muuda( int a = 1, int b = 1 ) { x = a; y = b; }
}
```

Defineerime Paar'ide jaoks:

- liitmise +
- omistamise =
- operaatori ++
- skalaariga korrutamise *
- skalaarkorrutise *

Selleks tuleb klassi Paar deklaratsiooni lisada:

```
        Paar  operator + ( Paar teine )
        {
                Paar  summa( x + teine.x, y + teine.y );
                return  summa;
        }
```

```
        Paar operator = ( Paar parem )
        {
            x = parem.x;  y = parem.y;
            return  *this;      // omistamisel on väärtus!
        }

        Paar operator ++ ( void )
        {
            x++ ;  y++ ;  return  *this;
        }

        Paar operator * ( int kordaja )
        {
            x *= kordaja;  y *= kordaja;  return *this;
        }

        int operator * ( Paar teine )
        {
            return  x * teine.x + y * teine.y;
        }
    }

// kasutamise näide

    int main( void )
    {
        Paar  u( 4, -3),  v( -2, 5),  w;
        w = u + v;    u++;
        v = w * 2;          // skalaariga ainult paremalt!
        int s = u * v;
        w.muuda( 7, -6);
        u.muuda( -4 );                  //  ( -4, 1)
        v.muuda( , 8 );                 //  ( 1, 8)
        w.muuda( );                     //  ( 1, 1)
    }
```

Reeglid:

- ei saa muuta operaatorite aarsust
- ei saa muuta operaatorite prioriteete avaldises
- ++v on lubatud, kuid realiseeritakse kui v++