2.7. Klassid ja objektid

Objektorienteeritud kood vastandub protseduurilisele programmeerimisele, mille kood on ühes kompaktses osas. Objektorienteeritud programmeerimise peamised eelised on klasside korduvas kasutuses ja vigade lokaliseerimises. Objektorienteeritud programmeerimisel luuakse tarkvara omavahel seotud objektidest. Objekt on objektiklassi esinemisjuht (*instance*). Objekt tuleb üldiselt enne kasumist üksikobjektina moodustada ehk instantsseerida (*instantiate*) sõnaga `new`, välja arvatud staatiliste klasside puhul, millel on vaid üks ühine esinemisjuht. Vt ka lõik Instantsseerimine.

Klassidel ja objektidel on liikmed, mis määravad nende omadusi ja käitumist (vt lähemalt). Klassi liikmete hulgas võivad olla näiteks väljad (*fields*), konstandid, omadused (*properties*), meetodid, sündmused (*events*) ja moodustajad (*constructors*). Neid käsitletakse lähemalt järgnevates alapeatükkides. Objektid pärivad oma omadused klassilt ja klassid endast kõrgema taseme klassidelt. Klassi liikmetel on määratud nähtavus (*accessibility*) (ptk Objektide nähtavus).

Klassid on objektide mudelid, mille järgi objekt luuakse.

Meetodid on toimingud, mida mingisse klassi kuuluvad objektid saavad teha. VS keeled pakuvad tuhandeid toiminguid ning kasutaja saab neid ka ise juurde luua.

Klassid, andmetüübid ja toimingud on hierarhiliselt organiseeritud nimeruumidesse (*namespace*), et vältida samanimelisust. Näiteks `System.Double` tähendab, et topelt-täpsusega reaalarvu tüüp kuulub nimeruumi `System`. Veebilehel kasutatav nupp kuulub nimeruumi `System.Web.UI.WebControls.Button`. Uue klassi sidumiseks nimeruumiga tuleb klassi lähtekood paigutada nimeruumi algus- ja lõpurea vahele. Klasside ja toimingute kompileeritud kood on põhiliselt *dll* failides, kusjuures üks *dll* fail võib sisaldada mitme nimeruumi klasse ja üks nimeruum võib paikneda mitmes *dll* failis.

Kui kasutatav nimeruum on koodi alguses osutatud käsuga `using`, siis ei ole seda tarvis klassi kasutamisel enam kirjutada. Projekti klasse ja nende liikmeid saab ülevaatlikult vaadata VS klassivaate aknas (*Class View Window*). Kui klassi nimi esineb rohkem kui ühes projektis kasutatavas nimeruumis, siis tuleb objektiklassi täisnime kasutada ka siis, kui see on koodi päises viidatud. Näiteks klass `Axis` on nii nimeruumis `Microsoft.Office.Interop.Excel` kui ka nimeruumis `System.Windows.Forms.DataVisualization.Charting`.

Järgneb pikslit esindava klassi näide ja klassi komponentide lühikirjeldus. Pane järgnevast koodist tähele, et klassi välju ja omadusi saab deklareerida mitme moel, enamasti piisab `{ get; set; }` aktsessoritega lühideklaratsioonidest.

```
public class Pixel
{
    public int ID;
    // See on avatud (public) väli, kuid älju ei soovitata otse avada.

    public float Value { get; set; }
    public short Col { get; set; }
    // Need on omaduse lühideklaratsioonid. Omaduse välja moodustab kompilaator.

    private short r;
    public short Row
    { // See on omaduse detailsem määratlus.
    // Omadusele saab lisada tingimusi ja funktsioone,
    // mis teevad selle kasutamise turvalisemaks.

        get => r;// See on omaduse lühike deklaratsioon privaatse välja kaudu.
```

```
        set
        {
            if (value < 0) _ = 0; // Rea number ei saa olla negatiivne
        }
    // get ja set on aktsessorid, st nad näevad ja vahendavad privaatseid välju.
    // Kui set aktsessor puuduks, siis oleks omadus vaid lugemiseks.
    }

    public Pixel(short row, short col)
    //See on moodustaja. Moodustaja nimi on sama, mis klassil.
    //Moodustaja argumendid on selle klassi objekti moodustamisel kohustuslikud.
    {
        r = row; // Siin kasutatakse privaatset vahendajat.
    // Eespool on välja Row määratluses: get => r;
        Col = col;
    // Välju ID ja Value siin ei ole. Neid ei pea objekti moodustamisel määrama.
    }

    // Järgnevad on meetodid.
    public static Pixel Random(short maxR, short maxC)
    {// Meetod väljastab juhusliku piksli.
        Random rnd = new Random();
        short row = Convert.ToInt16(rnd.Next(maxR));
        short col = Convert.ToInt16(rnd.Next(maxC));
        return new Pixel(row, col);
    }

    public bool Equals(Pixel p2)
    {// Meetod objektide võrdlemiseks, mis määrab võrdsuse tingimused.
        return Row == p2.Row && Col == p2.Col;
    }

    public static float EuclDistance(Pixel p1, Pixel p2)
    {// Meetod väljastab vahemaa kahe ruudukujulise piksli vahel.
     // Ühikuks on siin piksli külje pikkus.
        float distance = 0;
        if (!Equals(p1, p2))
        {
            float diffX = p2.Col - p1.Col;
            float diffY = p2.Row - p1.Row;
            distance = (float)Math.Sqrt(diffX * diffX + diffY * diffY);
        }
        return distance;
    }
}
```

Mõned näited selliselt moodustatud klassi `Pixel` kasutustest.

```
public class Program
{
    static void Main()
    {
        Pixel piksel1 = new Pixel(100, 10);
        // Deklareeritakse piksel real 100, veerus 10.
        piksel1.Value = 20;
        // piksel1 saab väärtuse 20.
        Pixel piksel2 = Pixel.Random(100, 100);
        // Deklareeritakse 100 x 100 pinnal juhuslikult paiknev piksel.
        int currentRow = piksel1.Row;// Esimese piksli rida.
```

```
        float distanceBetweenPixels = Pixel.EuclDistance(piksel1, piksel2);
        Console.WriteLine("Real " + currentRow +" veerus " + piksel1.Col + "
oleva piksli kaugus juhupikslini = " + distanceBetweenPixels);
        Console.ReadKey();// Vajalik, et konsooliaken ei sulguks.
    }
}
```

2.7.1. Väli

Väli (*field*) on klassiga seotud muutuja, mida reeglina kasutatakse vaid klassi piires. Klassis oleva välja adresseerimisel lisatakse klassi nimele punkt ja välja nimi. Igal klassil, väljal ja meetodil on nähtavus (reeglina kas `public` või `private`). Kui väli või meetod on staatiline, siis on see sama väärtusega klassi kõigis üksikobjektides.

2.7.2. Konstant

Konstant on muutumatu väärtusega väli, mis deklareeritakse sõnaga `const`. Näiteks järgmiselt: `public const int months = 12;`.

2.7.3. Omadus

Omadus (*property*) vahendab välju, sellele saab lisada tingimusi ja funktsioone, mis teevad omaduse kasutamise turvalisemaks kui sama välja vahendamata kasutus. Omaduse iserakenduva määratluse (*auto-implemented property*) puhul kasutatakse deklareerimisel vaid `{ get; set; }` aktsessoreid (*accessories*), mis vahendavad privaatsete väljade väärtusi.

```
public class Person
{
    public string Name { get; set; }
}
```

Kui `set` käsk puudub, on omadus vaid lugemiseks (*read only*). Aktsessorite abil saab klassi omadusi teisendada ja omistatavate väärtuste lubatavust kontrollida.

```
class TimePeriod
{
    private double seconds;

    public double Hours
    {
        get { return seconds / 3600; }
        set
        {
            if (value < 0 || value > 24)
               throw new ArgumentOutOfRangeException(
                          $"{nameof(value)} must be between 0 and 24.");

            seconds = value * 3600;
        }
    }
}
```

2.7.4. Moodustaja

Moodustaja (*constructor*) moodustab klassi — moodustaja nimi on sama, mis klassil. Moodustaja atribuudid on seda klassi objekti moodustamisel kohustuslikud. Vaikimisi moodustaja on ilma atribuutideta. Samal klassil võib olla mitu moodustajat, mis erinevad atribuutide komplekti poolest. Kuid sama atribuutide komplektiga moodustajaid saab olla vaid üks. Sel juhul on sellesse klassi kuuluva objekti instantsseerimiseks mitu võimalust. Kui moodustajas kasutatavad väljade nimed on samad, mis klassil, siis tuleb lisada määratlus `this`.

Iga atribuutide komplekti jaoks on omaette moodustaja.

```
public class Kasutaja
{
    private int id; // Klassi Kasutaja tunnus ehk väli
    public Kasutaja() { } // Minimaalne moodustaja
    public Kasutaja(int ID) // Atribuudiga moodustaja
    {
        id = ID;
    }
}
public class Kasutaja
{
     private int id; // Kasutaja tunnus ehk väli
     private string name;

     // Moodustaja, mis määrab välja vaikimisi väärtuse
     public Kasutaja()
     {
           id = 0;
     }

     // Kui atribuudi nimi on identne klassi välja nimega
     public Kasutaja(int id, string name)
     {
           this.id = id;
           this.name = name;
     }

     // Moodustaja lambda funktsioonina
     public Kasutaja(int id) => this.id = id;
}
```

Siin esitatud klassi saab instantsseerida nii koos omadusega `ID` kui ka ilma selleta või iD ja nimega, kuid mitte ainult nimega.

```
Kasutaja keegi = new Kasutaja();
Kasutaja teineKeegi = new Kasutaja(134); // ID koodiga
Kasutaja keegiOskar = new Kasutaja(134, "Oskar"); // ID koodi ja nimega
```

2.7.5. Meetod

Funktsioon on nimega koodijupp, mis käivitatakse selle kutsumisel. Objektorienteeritud keeltes on funktsioon alati mingi klassi osa ja seda nimetatakse **meetodiks**, kui jututeemaks on meetod ja klass, millesse see kuulub, ei ole parajasti oluline, siis kasutatakse ka üldisemat terminit funktsioon.

Meetod on objektiklassiga seotud funktsioon.

Meetodi parameetrid on sulgudes meetodi nime järel. Parameetrite puudumisel on sulud tühjad, kuid neid ära jätta ei või. Meetodi iga parameetri ees peab olema parameetri tüüp, parameetrid on eraldatud komaga. Mittekohustuslikel parameetritel peab olema vaikimisi väärtus ja need peavad olema parameetrite loetelu lõpus. Meetodi kutsumisel on parameetrite kohal konkreetse väärtusega muutujad, mida nimetatakse **argumentideks**. Argumendi tüüp peab olema sama, mis parameetril; argumendi nimi võib erineda parameetri nimest. Kui meetodi parameetril on vaikimisi väärtus, siis võib vastav argument puududa.

Vaikimisi väärtusega parameetrid peavad olema paremeetrite loetelu lõpus.

Meetod deklareeritakse järgmisel kujul.

```
<nähtavus> <väljundi tüüp> <nimi>(<parameetrid>)
{
    <meetodi kood>
}
```

Näiteks järgmine meetod liidab kaks ujukomaarvu ja väljastab summa samas vormingus.

```
public float AddNumbers(float liidetav1, float liidetav2)
{
      return liidetav1 + liidetav1;
}
```

Selle meetodi nähtavus (*accessibility*) on klassist `public`, see tähendab, et meetodit saab kutsuda ka teistest klassidest. Kui nähtavus on määratlemata, kuulub see klassi `private`, see tähendab on kasutatavad vaid sama klassi piires. Nähtavustest on põhjalikumalt kirjas eraldi peatükis.

Väljundi tüüp on meetodi poolt väljastatava muutuja või objekti tüüp. Kui meetodil ei ole otsest väljundit, siis kasutatakse sõna `void`. Kui meetodi väljastatava muutuja tüüp ei ole `void`, siis peaks meetod sisaldama käsku `return` ja sellele järgnevat väljastatavat muutujat. Käsku `return` saab kasutada ka meetodi töö lõpetamiseks.

Järgnevas näites võrreldakse kahte stringi meetodi abil, mida saab kasutada nii kahe kui ka kolme parameetriga. Kui kolmas parameeter on kas määratlemata või on `false`, siis suurtähti eristatakse. Meetodi kolmandas kasutuses on esitatud ka parameetrite nimed, mille kasutamine mõnikord parandab koodi loetavust. Parameetri nime võib kasutada kas kõikide või ka ainult osade parameetrite esitamisel. Kui kõik parameetrid on esitatud nimega, siis võib neid esitada suvalises järjekorras. Nimega parameetrid muudavad programmi aeglasemaks.

```
bool CompareStrings(string s1, string s2, bool capitalize = false)
{
     bool equal = false;
     if (capitalize)
     {
          if (s1.ToUpper() == s2.ToUpper()) equal = true;
     }
     else
     {
          if (s1 == s2) equal = true;
     }

     return equal;
}

    string lause1 = "c# kood";
    string lause2 = "C# kood";
    bool samaLause;

    samaLause = CompareStrings(lause1, lause2); // samaLause = false
    samaLause = CompareStrings(lause1, lause2, true); // samaLause = true
    samaLause = CompareStrings(s1: lause1, s2: lause2); // Siin on parameetrite nimed
```

Ümarsuslud meetodi nime järel on kohustuslikud nii meetodi deklareerimisel kui ka kasutamisel. Ka siis, kui meetodil ei ole parameetreid.

Vaikimisi algab C keeltes koodi täitmine meetodist `Main()`, kuid atribuudiga `[STAThread]` saab koodi käivitamise kohaks märkida ka mõne teise objekti.

Kõik C# keele klassid sisaldavad meetodit `Equals()`, mis kontrollib objektide võrdsust. Enamikel juhtudest on meetod `Equals()` samaväärne C# keele tehtega `==`. Seejuures kaks eraldi objektideks instantsseeritud sama väärtusega muutujat on võrdsed nii tehte `==` järgi kui ka meetodi `Equals()` abil võrreldes.

Kaks erandit on `string` tüüpi muutujate võrdlemisel. Esiteks, kui võrreldavad sama sisuga tekstid ei ole mõlemad deklareeritud `string` tüüpi muutujaks, näitab `==` tehe ebavõrdsust. Teiseks — kui üks võrreldavatest `string` objektidest on tühi, siis `==` annab tulemuseks *false*, `Equals()` annab vea (*NullReferenceException*). Proovi näiteks järgmist koodi.

```
object nimi = "Kalle";
char[] tähed = { 'K', 'a', 'l', 'l', 'e' };
object munimi = new string(tähed);
Console.WriteLine(nimi + "==" + munimi + " tulemus on {0}", nimi == munimi);
Console.WriteLine(nimi + ".Equals(" + munimi + ") tulemus on {0}",
munimi.Equals(nimi));
```

Delegaat

Delegaat koondab samade parameetritega ja sama väljunditüübiga funktsioone. Järgnevas näites deklareeritakse delegaat ja kaks sama tüüpi parameetritega lambda väljenditena esitatud funktsiooni, kus kasutatakse delegaati. Delegaate kasutavad muuhulgas sündmused.

```
delegate float ProcessDelegate(float p1, float p2);
static float Multiply(float p1, float p2) => p1 * p2;
static float Divide(float p1, float p2) => p1 / p2;

static float MultiplyOrDivide(float f1, float f2, string operation)
{
      ProcessDelegate process;
      if (operation == "M")
          process = Multiply;
      else
          process = Divide;

      return process(f1, f2);
}
```

Delegaat hoiab viiteid funktsioonidele.

Anonüümne meetod

Koodireal kasutatuna võib meetod olla anonüümne, st ilma nimeta. Anonüümne meetod defineeritud delegaadina ja tulemus omistatakse kas delegaadile või sündmuse haldajale. Näites toodud rida lisab nupu `button1` vajutusele funktsiooni, mis omistab `textBox1` tekstiks "0". Anonüümsed meetodid on peaaegu täielikult asendatavad lambda väljenditega.

```
button1.Click += delegate (object s, EventArgs e) { textBox1.Text = "0"; };
```

Action ja Func

Nimeruum `System` sisaldab toimingutüüpe `Action`, mis ei väljasta väärtust ja `Func`, mis on väljundiga. Need võimaldavad lihtsaid meetodeid lühemalt kirja panna. Toimingul võib olla sisend, nagu on järgmise koodi esimeses näites. Sõnale `Action` järgnevates nurksulgudes `< >` on sisendparameetri tüüp, võrdusmärgile järgnevates ümarsulgudes on väljendis kasutatud muutuja nimi. Näidetes on kasutatud Lambda väljendeid.

```
Action<int> toiming1 = (int x) => Console.WriteLine("Write {0}", x);
Action toiming_Valmis = () => Console.WriteLine("Valmis");
toiming1(1);
toiming_Valmis();
```

Üldistatud funktsioon `Func` on alati väljundiga. Nurksulgudes `< >` on sisendparameetrite tüübid ja viimasena väljundi tüüp.

```
Func<int, string> func1 = (x) => string.Format("See tunnus = {0}", x);
Console.WriteLine(func1(5));
```

Tüübil `Action` on üks parameeter ja ei ole väljundväärtust. Tüübil `Func` on üks parameeter ja on väljundväärtus.

Asünkroonne meetod

Asünkroonne meetod vabastab protsessori lõime (*thread*) kuniks märgisega `await` tähistatud tegum (`Task`) tehtud saab. Asünkroonne meetod aitab protsessori tööd tõhusamaks muuta pikemate iseseisvate arvutuste, failide üles- ja allalaadimise ja aega nõudvate andmebaasioperatsioonide ajal ning suure koormusega veebilahenduste puhul. Kiirelt toimiva arvutuse puhul pole asünkroonse meetodi järele olulist vajadust. Tasub siiski silmas pidada (veebi)lahenduse võimekust vastu pidada laiendamisele ja märksa intensiivsemale kasutusele (*scalability*). Asünkroonsete meetodite eelised ilmnevad eelkõige veebiserveri ülekoormuse korral. Asünkroonseid meetodeid käsitletakse ka peatükis 5.6.3.

2.7.6. Funktsioonide kattumine

Funktsioone eristatakse nende signatuuride järgi. Signatuuri moodustab funktsiooni nimi ja parameetrid, väljundi tüüp ei ole signatuuri osa. Näiteks järgmises koodis on kaks sama nimega funktsiooni maksimumväärtuse leidmiseks, üks täisarvudest ja teine ujukoma arvudest. Kattuva nimega funktsiooni adresseerimisel ei ole tarvis täpsustada, millist sama nimega varianti soovitakse kasutada — käivitatav funktsioon sõltub parameetriks oleva muutuja tüübist.

```
static int MaxValue(int[] intArray)
{
    int MaxVal = intArray[0];
    foreach (int val in intArray)
    {
        if (val > MaxVal) MaxVal = val;
    }
        return MaxVal;
}

static double MaxValue(double[] doubleArray)
{
    double MaxVal = doubleArray[0];
    foreach (double val in doubleArray)
        if (val > MaxVal) MaxVal = val;
    return MaxVal;
}
```

Sama klass võib sisaldada sama nimega meetodeid, kui nende meetodite parameetrid on erinevad.

2.7.7. Struktuurid

Struktuurid on muutujate ja funktsioonide konteinerid nagu klassidki. Järgmises näites luuakse struktuur `Klient`, mis sisaldab eesnime, perenime ning ees- ja perenime ühendamise funktsiooni. Moodulis `Main` on struktuuri kasutamise näide.

```
struct Klient
{
    public string Eesnimi;
    public string Perenimi;
    public string Nimi() => eesnimi + " " + perenimi;
}

static void Main()
{
    Klient minaIse;
    minaIse.Eesnimi = "Kalle";
    minaIse.Perenimi = "Remm";

    Console.WriteLine(minaIse.Nimi());
}
```

Struktuurid on sarnased klassidele. Klass on mudel, mille alusel luuakse objekte, struktuur on objektitüüp. Struktuuri moodustamisel ei ole tarvis kasutada sõna `new`, sest struktuur instantsseeritakse deklareerimisel alati. Klassid toetavad omaduste pärimist, struktuurid mitte. Klasse saab deklareerida staatilistena, struktuure mitte. Struktuuri väärtus võib olla `null`, klassil mitte. Vt ka ptk Loend ja struktuur.