Примена области важења

Промењиве се користе да би се запамтиле вредности и могу се креирати на разним местима у коду апликације:

```
private void SabiranjeKlikova(int a)
{
      int x = 0;
}
```

Метода SabiranjeKlikova креира промењиву x типа int и додељује јој иницијалну вредност.
Ова промењива настаје на месту дефинисања и све линије кода унутар методе могу да користе ову промењиву; завршетком методе, промењива нестаје и више се нигде не може користити.
Када се промењивој негде у програму може приступити, каже се да је промењива у области важења (досег, scope) на том месту.
Промењива x има област важења унутар методе, изван методе јој се не може приступити.
Промењива се може дефинисати изван методе а унутар класе и тој промењивој могу приступити све методе унутар те класе; таква промењива има област важења унутар класе.

Витичасте заграде које ограничавају тело методе, дефинишу област важења за ту методу; све промењиве дефинисане унутра су локалне промењиве за ту методу.
То такође значи да се ове методе не могу користити за дељење информација између метода.
Витичасте заградекоје дефинишу тело класе, дефинишу област важења за ту класу.
Све промењиве које се декларишу унутар тела класе (али не и унутар метода класе) имају ту област важења, такве промењиве се називају пољима; могу да се користе за дељење информација између метода.
Разлика између поља и локалних промењивих је у томе што локалну промењиву прво морамо дефинисати да би смо је користили; у случају поља компајлер уређује неке ствари сам по себи.

Преклапање метода

Ако два идентификатора имају исти назив и декларисани су у истој области важења, за њих се кађе да су преклопљени (overloaded).
Постоји начин да се преклопи идентификатор за неку методу, ако је то корисно и важно за рад.
Метода WriteLine() нуди велики број верзија са различитим скуповима параметара а компајлер током процеса компајлирања тражи типове параметара и на основу тога одређује која ће се верзија методе позвати.
Преклапање је корисно када треба исту операцију обавити са различитим типовима података или групама информација.
Метода се преклапа са истим именом али различитим бројем параметара или различитим типовима параметара; типови разултата које метода враћа се не преклапају.

Преклапање оператора

Највећи број уграђених оператора у C# се може предефинисати или преклопити, па зато програмер може користити операторе и са кориснички дефинисаним типовима.
Преклопљени оператори су функције са посебним именима, службеном речи operator и са самим симболом операције.
Сваки преклопљени оператор има повратни тип и листу параметара:

```
class Kutija
    {
        private double duzina;
        private double sirina;
        private double visina;
        public double RacunanjeZapremine()
        { return duzina * sirina * visina; }
        public void PostavljanjeDuzine(double duz)
        { duzina = duz; }
        public void PostavljanjeSirine(double sir)
        { sirina = sir; }
        public void PostavljanjeVisine(double vis)
        { visina = vis; }

        public static Kutija operator+ (Kutija b, Kutija c)
        {
            Kutija kutija = new Kutija();
            kutija.duzina = b.duzina + c.duzina;
            kutija.sirina = b.sirina + c.sirina;
            kutija.visina = b.visina + c.visina;
            return kutija;
        }
    }
```

Парцијалне (делимичне) класе

Класе у C# имају посебну способност да имплементирају функционалност једне класе у више фајлова а сви ти фајлови се комбинују у једну класу када се апликација компајлира.

Службена реч partial се користи и за поделу функционалности метода, интерфејса или структура у више фајлова.

Сваки део дефиниције парцијалне класе треба бити у истом пројекту и именском простору, али се може користити различити називи за сорс фајлове.

Ако је било који део парцијалне класе декларисан као апстрактан, запечаћен или основни, онда је цела класа декларисана у истом типу.

У примеру се користе два фајла која садрже једну парцијалну класу Klasa; сорс фајлови се називају Klasa1.cs, Klasa2.cs:

```
public partial class Klasa
{
     private string imeAutora;
     private int ukupnoClanaka;
     public Klasa(string a, int t)
     {
          this.imeAutora = a;
          this.ukupnoClanaka = t;
     }
}
```

Фајл Klasa2.cs:

```
public partial class Klasa
{
     public void Prikaz()
     {
          Console.WriteLine("Ime autora: " + imeAutora);
          Console.WriteLine("Ukupan broj clanaka: " + ukupnoClanaka);
     }
}
```

Динамичке библиотеке класа

Библиотеке класа су засебни пројекти у C#.

Треба изабрати C#, Windows, Library и тип ClassLibrary (.NET Framework).

Класе из ових библиотека имају код који се може прикључити другим типовима пројеката (нпр Console Application, WPF Application...).

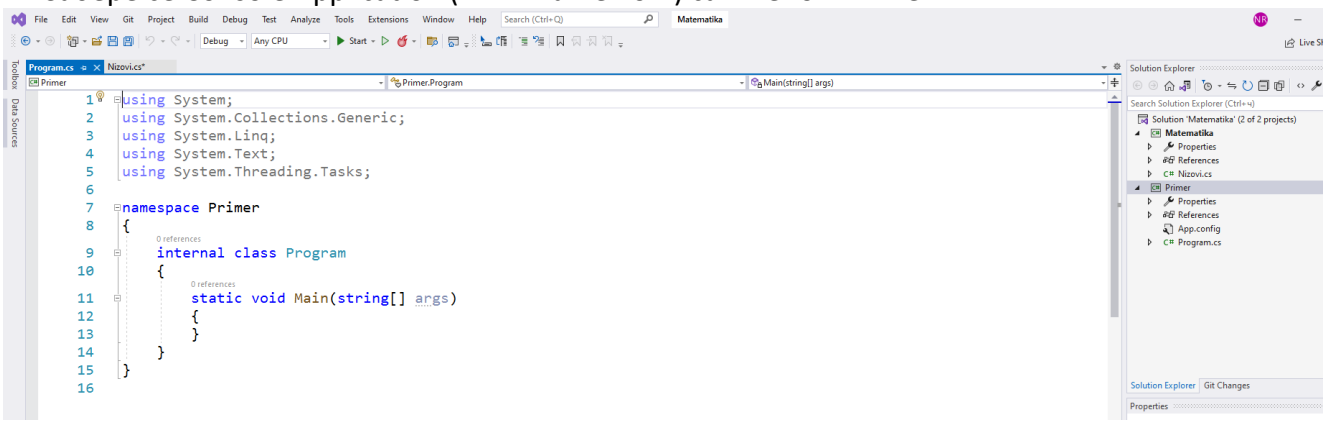
Једна библиотека класа се може користити у више пројеката.

Дати назив именског простора Математика, фајл Nizovi.cs:

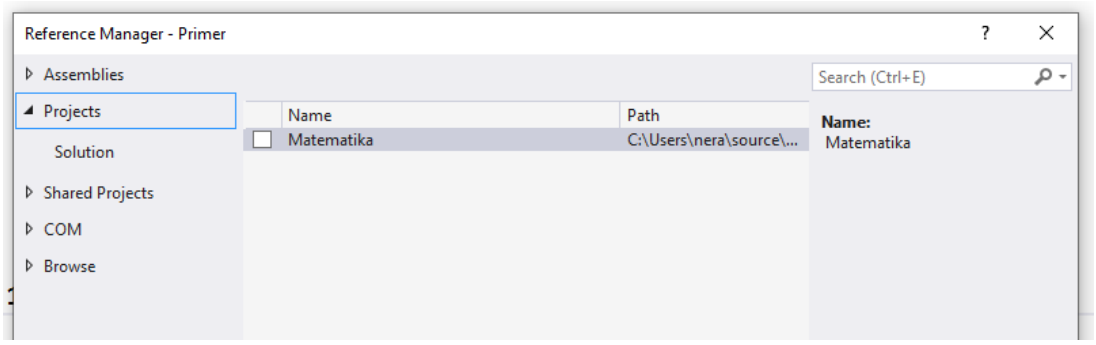
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Matematika
{
    public class Nizovi
    {
        public static double Max(double[] niz)
        {
            double m = niz[0];
            foreach (double x in niz)
                if (x > m)
                    m = x;
            return m;
        }
        public static double Min(double[] niz)
        {
            double m = niz[0];
            foreach (double x in niz)
                if (x < m)
                    m = x;
            return m;
        }
        public static double Prosek(double[] niz)
```

```
        {
            double zbir = 0;
            foreach (double x in niz)
                zbir += x;
            return zbir / niz.Length;
        }
    }
}
```

Класа Nizovi у себи има три методе које су статичке и јавне и све три враћају резултат
Пројекат Matematika је библиотека класа и ако се компајлира добија се порука о успеху али ако се стартује добија се порука о грешци, да се пројекат овог типа не може директно стартовати.
Да би се он искористио, мора се направити нови пројекат, десни клик на Solution->Add New Project и изабере се Console Application (.NET Framework) са именом Primer.



Десни клик на Primer у SE и клик на падајућем менију Set As Startup Project.
Десни клик на References испод Primer и избор Add Reference:



Клик на чек бокс и тако се прави референца на пројекат Математика и сада су доступне све класе из библиотеке и њихове јавне методе.
Када се у коду откуца Nizovi, са леве стране икона са сијалицом клик на стрелицу и изабрати using Matematika; чиме се појављује линија кода у заглављу.

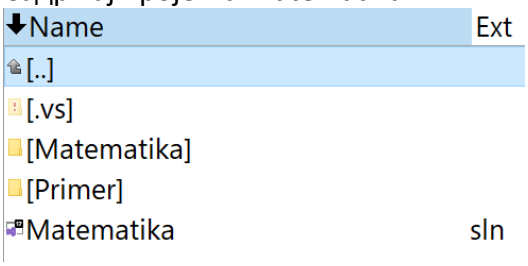
```
static void Main(string[] args)
{
    double[] brojevi = { 1, 2, 3, 4, -6, 0, 15, -39 };
    Console.WriteLine($"Max: {Nizovi.Max(brojevi)}, " +
                      $"Min: {Nizovi.Min(brojevi)},
                     Prosek: {Nizovi.Prosek(brojevi)}");
```

}

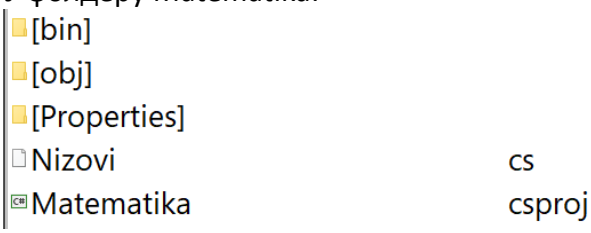
Даје: Max: 15, Min: -39, Prosek: -2.5

Шта је резултат компајлирања библиотеке класа?

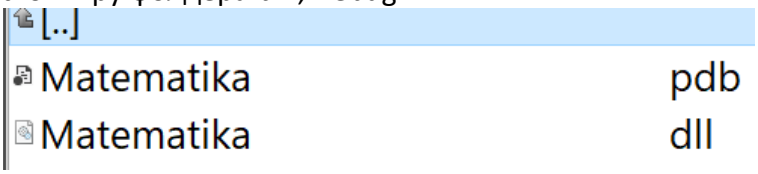
Садржај пројекта Matematika:



У фолдеру Matematika:



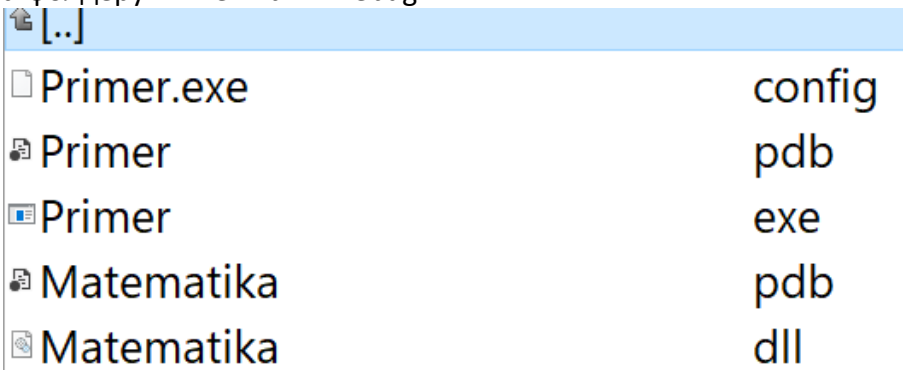
У оквиру фолдера bin, Debug:



Фајл Matematika.dll је типа application extension (dynamic link library) која се динамички повезује са главним програмом

Фајл .pdb је потребан само за дебаговање.

У фолдеру Primer->bin->Debug:



<u>Задаци за самосталан рад</u>

1. Написати програм којим се креира преклапање методе Sabiranje у зависности од броја аргумената (два и три аргумента) који се сабирају.

2. Написати програм којим се креира преклапање чланова методе Sabiranje у зависности од типа аргумената (int и float) који се сабирају.

3. Написати програм којим се дефинише класа Zaposleni у којој се налази приватна промењива без иницијализације, дифолтни конструктор којим се повећава вредност промењиве за један, својство којим се само чита број инстанцираних радника.