ДРУШТВО МАТЕМАТИЧА СРБИЈЕ

АКРЕДИТОВАНИ СЕМИНАР:

345

ДРЖАВНИ СЕМИНАР О НАСТАВИ МАТЕМАТИКЕ И РАЧУНАРСТВА ДРУШТВА МАТЕМАТИЧАРА СРБИЈЕ

Компетенција: К1
Приоритети: 3

ТЕМА:

НАСТАВА ОБЈЕКТНО-ОРИЈЕНТАСАНОГ ПРОГРАМИРАЊА

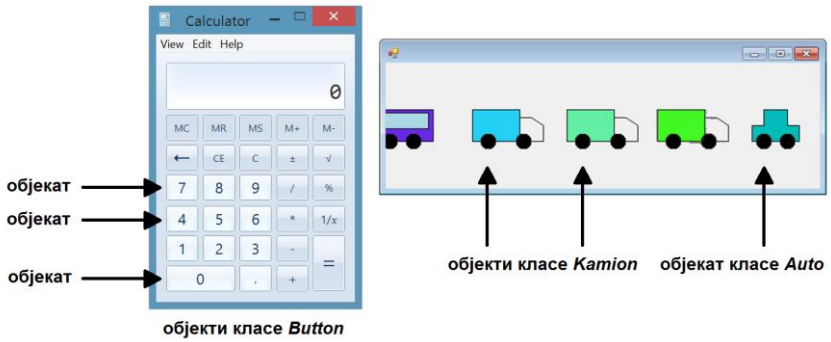
РЕАЛИЗАТОРИ СЕМИНАРА:

ДУША ВУКОВИЋ

БЕОГРАД,
09. – 10. 02. 2019.

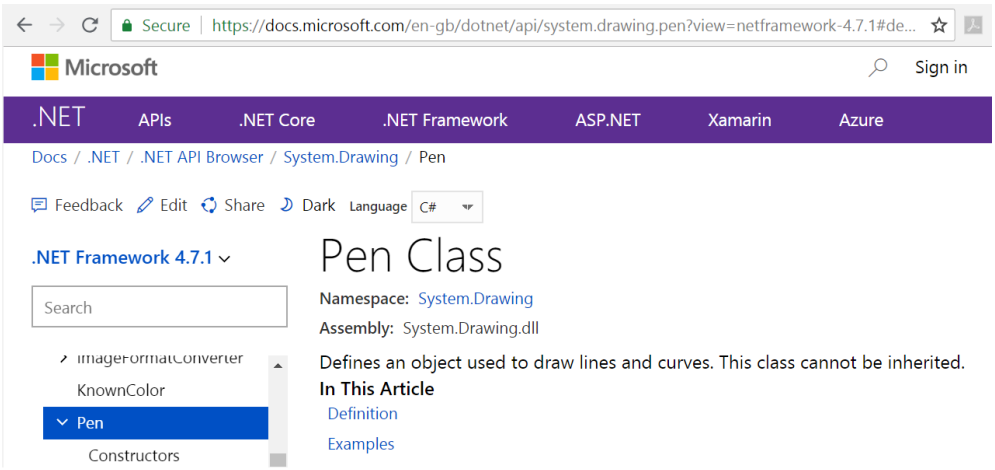
Програмски језик C# је објекно-оријентисани програмски језик који је јако добар за упознавање ученика са објектно-оријентисаним програмирањем и то са процесом грађења класа и креирањем апликација у којима ће се користити објекти креираних класа. Књига „Програмирање - класе и објекти“ издавачке куће CET се бави овом темом на начин прилагођен средњошколском узрасту.

Данашње апликације су креиране употребом објеката. Свако дугме, свако поље за унос текста, као и сваки други елемент корисничког инферфејса апликације је објекат. Сваки учесник у некој игрици је такође објекат, на пример: фудбалер, кошаркаш, аутомобил, маца, хамбургер, балон и сл.



Објекат обједињује у једну целину податке и функције (шта и на који начин може да се ради са тим подацима). Дефиницијом класе су описани сви објекти те класе. Неко опште правило је да су сви подаци, тј. атрибути, приватни и на тај начин заштићени, док су све функције, тј. методе, јавне и помоћу њих се контролисано управља објектом. Ово правило се назива **енкапсулација**. Поред метода, класе имају и друге јавне елементе којима се дефинише рад са објектима те класе, на пример конструктор који се користи за креирање објекта.

Постоји велики број већ креираних класа чије објекте можемо да користимо приликом креирања наших апликација. Тако, на пример, не морамо ми да креирамо класу која ће да опише дугме на екрану зато што већ постоји класа *Button* и ми у наше апликације можемо да додајемо колико год желимо објеката ове класе. Уколико, на пример, желимо да радимо са графиком, можемо да користимо класе: *Pen, SolidBrush, Graphics*...



Поред употребе већ постојећих класа, често постоји потреба и да креирамо нове класе које ћемо користити у апликацијама. Када се говори о настави објектно-оријентисаног програмирања, говори се управо о томе да се са ученицима ради на томе да савладају процес креирања класа.

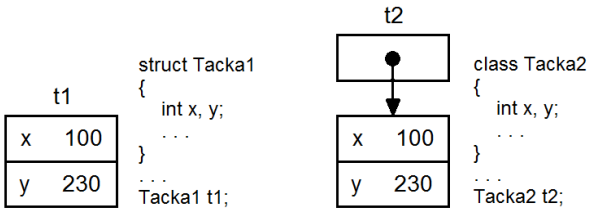
Права предност објектно оријентисаног програмирања се види тек када исту класу употребимо у неколико апликација, зато што се онда далеко смањује количина програмирања у свим следећим апликацијама где користимо објекте једном креиране класе.

Механизам који такође олакшава програмирање и доводи до мањег простора за грешке је **наслеђивање**. Све оно што је заједничко за неколико класа се пише само једном у надкласи и тиме се избегава непотребно понављање. На пример, свако возило може да се покрене, креће, заустави, па ће се те функционалности издвојити у надкласу, док ће се изведене класе разликовати по приказу: ауто, аутобус и камион не изгледају исто.

Програмски језик C# је креиран као чист објектно-оријентисани програмски језик што може да се види и у томе да су чак и основни типови података заправо сложени конструкти. За једну, на пример, целобројну промељиву, можемо да позовемо методу за конверзију у текстуални податак, тј. радимо са њом као са објектом.

int a = 15; textBox1.Text = a.ToString();

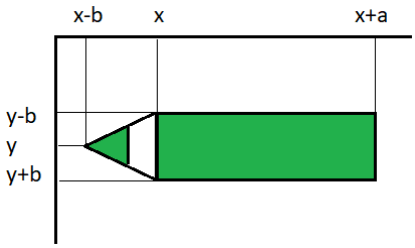
Основни типови података су реализовани као структуре. Тако је целобројни тип *int* заправо структура *Int32*. За разлику од класа, структуре спадају у вредносне типове. Класе спадају у референтне, што значи да уз сваки податак имамо и референцу, показивач. Уколико бисмо имали задатак да дефинишемо тип података чији ће се објекти користити у апликацијама, можемо да изаберемо класу или структуру. Уколико одаберемо класу, уз сваки објекат ћемо у меморији имати и показивач на њега.



Најчешће се креирају класе, док је за неке једноставније ситуације згодније употребити структуре. Тако су, од готових типова које користимо у апликацијама, *Point* и *Color* структуре. Постоји много сличности у раду са објектима структура и класе, али постоје и разлике. Када са ученицима пролазимо кроз концепте објектно-оријентисаног програмирања, можемо, без улажења у разлике, да се концентиршемо на опште концепте рада са објектима, од којих се многи односе и на класе и на структуре.

Дефиниција класе *Olovka*

Креираћемо класу *Olovka* са свим потребним атрибутима, методама и конструкторима за апликације које следе.



Унутар конструктора ћемо проверити да ли су димензије позитивни бројеви. Потребно је имати у виду да је увек добро вршити провере у јавним методама. У класи је дефинисан један конструктор који узима онолико параметара колико има приватних атрибута. Уколико је бар један број прослеђен за димензију оловке негативан, долази до испаљивања изузетка. Уколико су димензије позитивне, сви прослеђени подаци ће се наредбама доделе уписати у приватне атрибуте.

Следећи покушај креирања објекта ће бити неуспешан и испалиће изузетак зато што је један од бројева који се односи на димензије оловке негативан.

Olovka olovka = new Olovka(e.X, e.Y, -100, 15, Color.Green);

Након што се позивом конструктора креира објекат, више није могуће променити ни његов положај, ни димензију, ни боју. Следи комплетан код класе.

```
class Olovka
{
   int x, y, a, b;
   Color boja;

   public Olovka(int x, int y, int a, int b, Color boja)
   {
      if (a < 0 || b < 0)
         throw new Exception("Dimenzije moraju da budu >=0");
      else
      {
         this.x = x;
         this.y = y;
         this.a = a;
         this.b = b;
         this.boja = boja;
      }
   }
   public void Crtaj(Graphics g)
   {
      Point t1 = new Point(x - b, y);
      Point t2 = new Point(x - b / 2, y - b / 2);
      Point t3 = new Point(x - b / 2, y + b / 2);
      Point[] vrh = { t1, t2, t3 };
      SolidBrush cetka = new SolidBrush(boja);
      g.FillPolygon(cetka, vrh);
      g.FillRectangle(cetka, x, y - b, a, 2 * b);

      Pen olovka = new Pen(Color.Black, 2);
      g.DrawRectangle(olovka, x, y - b, a, 2 * b);
      g.DrawLine(olovka, t1, new Point(x, y - b));
      g.DrawLine(olovka, t1, new Point(x, y + b));
      g.DrawLine(olovka, t2, t3);
   }
```

Прва апликација

Креирати апликацију у којој корисник црта оловке тако што двоструко кликне на жељену позицију. Све оловке су исте величине, али су различитих (насумичних) боја.

```
   Random r = new Random();
   private void Form1_MouseDoubleClick(object sender, MouseEventArgs e)
   {
      Color boja = Color.FromArgb(r.Next(256), r.Next(256), r.Next(256));
      Olovka olovka = new Olovka(e.X, e.Y, 100, 15, boja);
      Graphics g = CreateGraphics();
      olovka.Crtaj(g);
   }
```

Друга апликација

Креирати апликацију у којој се исцртава оловка на основу унетих података. Следи комплетан код апликације.

```
Color boja = Color.Black;
private void pictureBox1_Click(object sender, EventArgs e)
{
   colorDialog1.ShowDialog(); boja = colorDialog1.Color; pictureBox1.BackColor = boja;
}
private void button1_Click(object sender, EventArgs e)
{
   try
   {
      int x = Convert.ToInt32(textBox1.Text);
      int y = Convert.ToInt32(textBox2.Text);
      int a = Convert.ToInt32(textBox3.Text);
      int b = Convert.ToInt32(textBox4.Text);
      Olovka olovka = new Olovka(x, y, a, b, boja);
      Graphics g = CreateGraphics();
      olovka.Crtaj(g);
   }
   catch (Exception izuzetak)
   {
       MessageBox.Show(izuzetak.Message, "Greska");
   }
}
```

Дефиниција класе Proizvod

Креираћемо класу *Proizvod* са свим потребним атрибутима (бар код, назив, цена), методама и конструкторима за апликацију која следи. Унутар конструктора треба да се проверава исправност података који треба да се упишу у приватне атрибуте. За цену треба проверити да није негативна, а за текстуалне податке да имају неки садржај.

Производ можемо успешно да креирамо позивом овог конструктора на следећи начин.

```
Proizvod p = new Proizvod("111111222222","hleb", 45);
```

Како ћемо имати у дефиницији класе и подразумевани конструктор који изоставља вредности, објекат можемо да креирамо и употребом њега након чега би се вредности попуниле из фајла.

```
StreamReader f = new StreamReader("spisak.txt");
Proizvod p = new Proizvod();
p.Citaj(f);
```

Унутар дефиниције класе је могуће реализовати и неке предефинисане методе, као што је метода *ToString.* Тај процес се назива редефиниција методе (енг. override). Следи комплетан код класе.

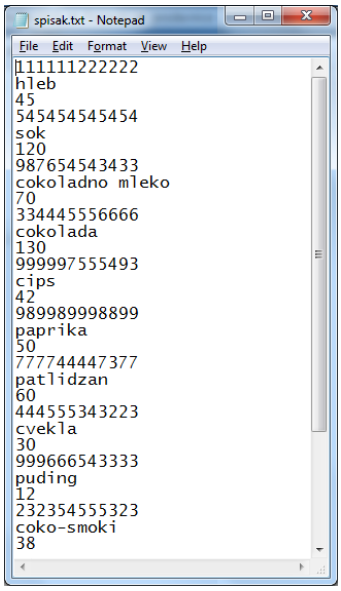
```
class Proizvod
{
   string barKod, naziv;
   double cena;

   public Proizvod()
   {
      barKod = naziv = String.Empty;
      cena = 0;
   }
```

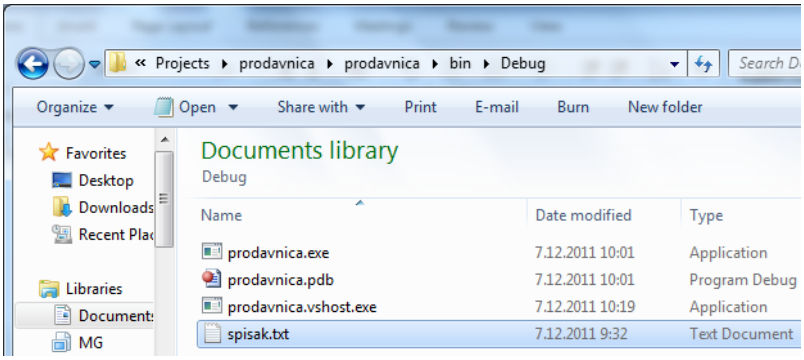
```
    public Proizvod(string barKod, string naziv, double cena)
    {
       if (cena < 0 || naziv.Equals(String.Empty) || barKod.Equals(String.Empty))
          throw new Exception("Neispravni podaci");
       else
       {
          this.barKod = barKod;
          this.naziv = naziv;
          this.cena = cena;
       }
    }

    public void Citaj(StreamReader f)
    {
       barKod = f.ReadLine();
       naziv = f.ReadLine();
       cena = Convert.ToDouble(f.ReadLine());
    }

    public bool NazivPocinjeNa(string s)
    {
       return naziv.StartsWith(s);
    }

    public override string ToString()
    {
       return barKod+": "+naziv+", "+cena.ToString()+"din";
    }
 }
```

Апликација

Креирати апликацију у којој корисник може да претражује по називу списак производа уписаних у текстуалном фајлу. У текстуалном фајлу се налази највише 100 производа.

Пре покретања апликације, креирати текстуални фајл са списком производа. Сачувати га у *bin\Debug* фолдеру пројекту.





Следи комплетан код апликације.

```
    Proizvod[] p = new Proizvod[100];
    int n = 0;

    private void Form1_Load(object sender, EventArgs e)
    {
       try
       {
          StreamReader f = new StreamReader("spisak.txt");
          while (!f.EndOfStream)
          {
             p[n] = new Proizvod();
             p[n].Citaj(f);
             listBox1.Items.Add(p[n].ToString());
             n++;
          }
          f.Close();
       }
       catch (Exception izuzetak)
       {
          MessageBox.Show(izuzetak.Message, "GRESKA");
       }
    }

    private void textBox1_TextChanged(object sender, EventArgs e)
    {
       listBox1.Items.Clear();
       string s = textBox1.Text;
       for (int i = 0; i < n; i++)
       {
          if (p[i].NazivPocinjeNa(s))
             listBox1.Items.Add(p[i].ToString());
       }
    }
  }
}
```