Tehnike vizuelnog programiranja – C#

Osnove GDI+

Deo 2.

Iscrtavanje teksta

- DC sadrži ***DrawString*** metodu za prikaz teksta u klientskoj oblasti. Četkice se koriste zajedno sa fontovima za prikaz u određenoj boji.

- Kreiranje četkica i fontova:

```
private Brush blackBr = Brushes.Black;
private Brush redBr = Brushes.Red;
private Brush royalBlueBr = Brushes.RoyalBlue;

private Font boldTRFont = new Font("Times New Roman",14,FontStyle.Bold);
private Font italicCOFont = new Font("Courier",12,FontStyle.Italic);
private Font ARFont = new Font("Arial",10,FontStyle.Regular);
```

```
private void Form1_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
    Graphics dc = e.Graphics;
    String s1 = "This is a test of Fonts" ;
    String s2 = "This line is in italics";
    String s3 = "This line is in Arial, regular style";

    dc.DrawString(s1, boldTRFont, blackBr, new Point(0,20));
    dc.DrawString(s1, italicCOFont, redBr, new Point(0,40));
    dc.DrawString(s1, ARFont, royalBlueBr, new Point(0,60));
}
```

Region

- Region je kolekcija piksela unutar prozora gde operativni sistem dozvoljava iscrtavanje. Van regiona se crtanje ne obavlja.
- Koordinate regiona su relativne u odnosu na gornji levi ugao kontrole – ne klientskog dela kontrole.

```
System.Drawing.Drawing2D.GraphicsPath gpath =
                            new System.Drawing.Drawing2D.GraphicsPath();
           this.Size = new Size(200, 200);
           gpath.AddEllipse(0, 0, 200, 50);
           gpath.AddRectangle(new Rectangle(10, 30, 190, 170));
           this.Region = new Region(gpath);
```

Štampanje

- Klase
  - PrintDocument
  - PrintPreviewDialog
  - PrintDialog

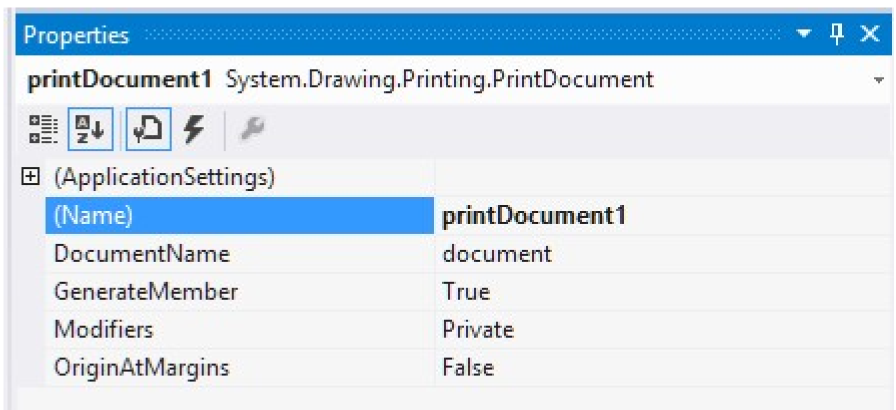
  - Za dokument je vezano štampanje, dok se klase *PrintDialog* i *PrintPreviewDialog* vezuju za dokument.

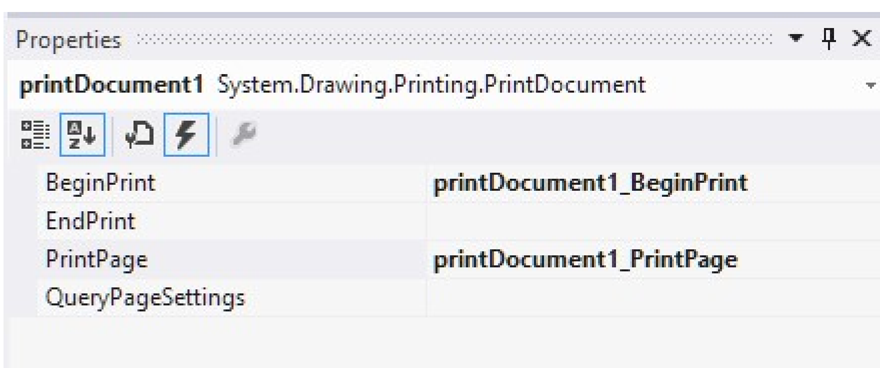
PrintDocument

- PrintDocument printDocument1;

- private void printDocument1_PrintPage(object sender, PrintPageEventArgs e) {

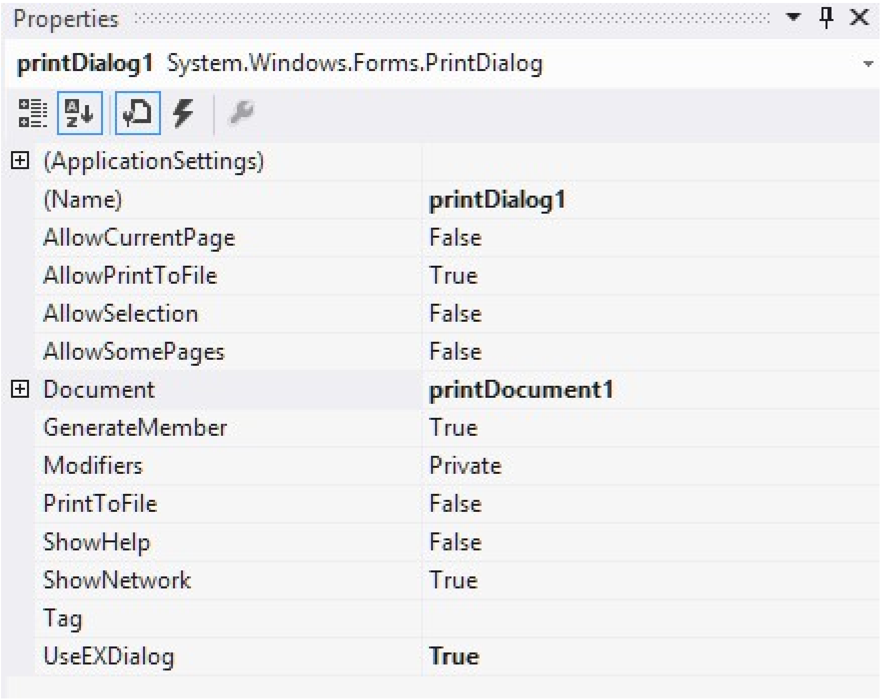
```
    Graphics dc = e.Graphics;
    crtanje(dc);
}
```





PrintDialog

- PrintDialog printDialog;



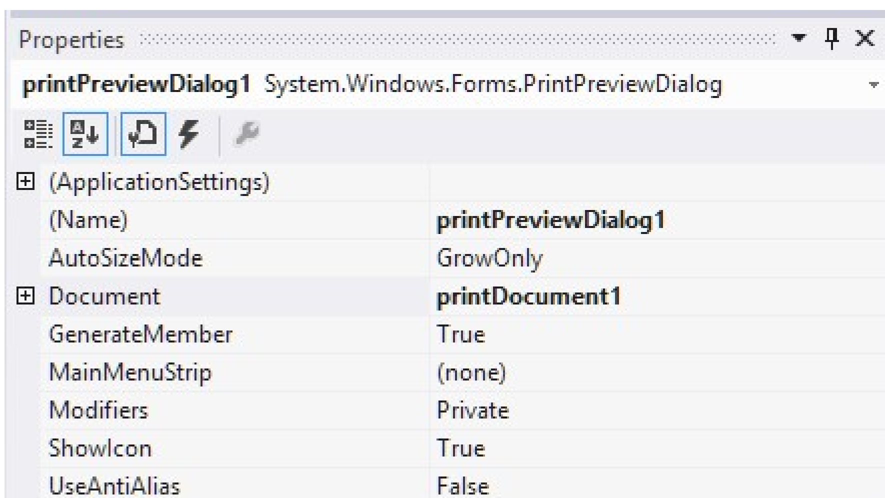
PrintPreviewDialog

- PrintPreviewDialog printPreviewDialog1;

- void printPreviewDialog1_Click(object sender, EventArgs e) {

  printPreviewDialog1.ShowDialog();

  }



Standardni dijalozi za rad sa grafičkim objektima

Common Font Dialog

```
FontDialog fontDlg = new FontDialog();
//fontDlg.Color = Color.Red ;  // default color
if (fontDlg.ShowDialog() == DialogResult.OK)
{
    font = fontDlg.Font;
    Invalidate();
}
```

ColorDialog

```
ColorDialog colorDlg = new ColorDialog();
if (colorDlg.ShowDialog() == DialogResult.OK)
{
  fontColor = colorDlg.Color;
  Invalidate();
}
```

Odabrane teme

Deo 1.

Nullable tipovi

- T? isto što i System.Nullable<T>

```
int n = 2;
double f = 1.2;
n = null; //Greska
f = null; //Greska

int? n2 = 2;
double? f2 = 1.2;
System.Nullable<int> n3;
n2 = null;
f2 = null;
n3 = null;
```

- Konverzije

```
int i = 123;
int? x = i;          // int --> int?
i = x;               // greska
double? y = x;       // int? --> double?
int? z = (int?)y;    // double? --> int?
int j = (int)z;      // int? --> int
```

Provera null vrednosti

- Ako je vrednost prvog operanda nula, onda operator vraća vrednost drugog operanda, u suprotnom vraća vrijednost prvog operanda.

```
static void Main(string[] args)
{
    double? num1 = null;
    double? num2 = 3.14157;
    double num3;
    num3 = num1 ?? 5.34;
    Console.WriteLine(" Value of num3: {0}", num3);
    num3 = num2 ?? 5.34;
    Console.WriteLine(" Value of num3: {0}", num3);
    Console.ReadLine();
}
```

```
Value of num3: 5.34
Value of num3: 3.14157
```

Indekseri

- Indekser dozvoljava objektu da bude indeksiran kao što je niz. Kada definišete indekser za klasu, ova klasa se ponaša slično virtuelnom nizu. Tada možete pristupiti instanci ove klase pomoću operatora pristupa nizu ([]).
- Indekeri mogu biti preopterećeni. Mogu deklarisati više parametara i svaki parametar može biti drugačiji tip. Nije neophodno da indeksi moraju biti celi brojevi. C# dozvoljava indeksima da budu od drugih tipova, na primer, nizova.

```
class IndImena
{
    private string[] arrImena = new string[size];
    static public int size = 10;

    public IndImena()
    {
        for (int i = 0; i < size; i++)
            arrImena[i] = "...";
    }
    public string this[int index]
    {
        get
        {
            string tmp;

            if (index >= 0 && index <= size - 1)
            {
                tmp = arrImena[index];
            }
            else
            {
                tmp = "";
            }

            return (tmp);
        }
        set
        {
            if (index >= 0 && index <= size - 1)
            {
                arrImena[index] = value;
            }
        }
    }
    public int this[string name]
    {
        get
        {
            int index = 0;

            while (index < size)
            {
                if (arrImena[index] == name)
                {
                    return index;
                }
                index++;
            }
            return index;
        }
    }
}
```

Iteratori

- Iterator vrši prilagođenu iteraciju dobavljanja vrednosti kolekcije. Metoda iteratora koristi **yield return** kako bi vratila element kolekcije - jedan po jedan. Kada se izvrši **yield return**, zapamti se trenutna lokacija u kodu. Izvršenje se ponovo pokreće sa te lokacije sledeći put kada se pozove funkcija iteratora.
- Upotreba iteratora se vrši koristeći foreach-a ili pomoću LINQ upita.
- Primer: Prva iteracija foreach-petlje dovodi do izvršenja u metodu iteratora *SomeNumbers* sve dok se ne dostigne prva **yield return**. Ova iteracija vraća vrednost 3, a trenutna lokacija u metodi iteratora je zadržana. Na sledećoj iteraciji petlje, izvršenje u iteratorskoj metodi se nastavlja od mesta gde je stalo, ponovo se zaustavlja kada dostigne **yield return**. Ova iteracija vraća vrednost 5, a trenutna lokacija u metodi iteratora se ponovo zadržava. Krug se završava kada se dostigne kraj metoda iteratora.

```
static void Main()
{
    foreach (int number in SomeNumbers())
    {
        Console.ReadKey();
        Console.Write(number.ToString() + " ");
    }

    Console.ReadKey();
}

public static System.Collections.IEnumerable
SomeNumbers()
{
    yield return 3;
    yield return 5;
    yield return 8;
}
```

- Povratna vrednost nekog iteratora mora biti IEnumerable, IEnumerable<T>, IEnumerator, or IEnumerator<T>.
- Može se koristiti **yield break** naredba za iznenadni kraj iteracija.

Kreiranje klase koja je kolekcija

- Napravićemo klasu DaniNedelje koja implementira interfejs IEnumerable, koji zahteva metodu **GetEnumerator**. Kompajler implicitno poziva tu metodu koja vraća vrednost tipa IEnumerator.
- Metod GetEnumerator vraća string svaki put kada se koristi naredba **yield return**.

```
static void Main()
{
    DaniNedelje days = new DaniNedelje();

    foreach (string day in days)
    {
        Console.ReadKey();
        Console.Write(day + " ");
    }
    // izlaz: Pon Uto Sre Čet Pet Sub Ned

}

public class DaniNedelje : IEnumerable
{
    private string[] days = { "Pon", "Uto", "Sre", "Čet",
"Pet", "Sub", "Ned" };

    public IEnumerator GetEnumerator()
    {
        for (int index = 0; index < days.Length; index++)
        {
            yield return days[index];
        }
    }
}
```

Primer:
Formiranje svojstva koje je kolekcija uz primenu iteratora

```
public static IEnumerable<Products> GetProducts
{
    get{
        yield return new Products() { Id = 1, Name = "Product1" };
        yield return new Products() { Id = 2, Name = "Product2" };
        yield return new Products() { Id = 3, Name = "Product3" };
    }

}
public static void Main(string[] args)
{
    foreach (Products p in GetProducts)
    {
        Console.WriteLine(String.Format("Product Id: {0}, Name: {1}", p.Id, p.Name));
    }
}
public class Products
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Iterators

```
public class List<T>
{
   public IEnumerator<T> GetEnumerator() {
      for (int i = 0; i < count; i++)
         yield return elements[i];
   }

   public IEnumerable<T> Descending() {
      for (int i = count - 1; i >= 0; i--)
         yield return elements[i];
   }

   public IEnumerable<T> Subrange(int index, int n) {
      for (int i = 0; i < n; i++)
         yield return elements[index + i];
   }
}
```

```
List<Item> items = GetItemList();
foreach (Item x in items) {...}
foreach (Item x in items.Descending()) {...}
foreach (Item x in Items.Subrange(10, 20)) {...}
```

Regularni izrazi

- Regularni izraz je obrazac koji se može upariti sa ulaznim tekstom. .Net pruža mehanizam testiranja primenom regularnih izraza.
- Prostor imena:
- using System.Text.RegularExpressions;

```
using System.Text.RegularExpressions;

private static void showMatch(string text, string
expr)
{
    MatchCollection mc = Regex.Matches(text, expr);
    foreach (Match m in mc)
        Console.WriteLine("" + m.Index+": " + m);
}

static void Main(string[] args)
{
    string str = "Ovo je samO jedna 1 proba!";

    Console.WriteLine("Izdvajanje reci: ");
    showMatch(str, @"\b\w{1,}\b");
    Console.ReadKey();

    str = "OvojesamOjedna1proba!";
    var words = Regex.Split(str, @"(je|1)").ToList();

    str = Regex.Replace(str, @"\d", "jedan");
}
```