C#

dodaci

Parcijalne klase

- Postoji mogućnost da se definicija klase podeli u više različitih file-ova
- Definicija dela klase se označava sa ključnom reči partial

```
partial class Circle
{
        public Circle() // default constructor
        {
                radius = 0;
        }
                public Circle(int initialRadius) // overloaded constructor
        {
                radius = initialRadius;
        }
}
partial class Circle
{
        public double Area()
        {
                return Math.PI * radius * radius;
        }
        private int radius;
        }
```

Implicitna deklaracija

```
var myVariable = 99; //implicitna deklaracija u int
var myOtherVariable = “Hello”; //implicitna deklaracija u string
var yetAnotherVariable; // Error - compiler cannot infer type
```

Anonimne klase

- Anonimne klase su klase bez imena
- Deklarišu se implicitno

```
var myAnonymousObject = new { Name = “John”, Age = 42 };
```

Novoj klasi se ne zna ime, ima dva polja Name i Age
Var rešava problem nepoznavanja imena klase

- Korišćenje objekta anonimne klase:

```
Console.WriteLine(“Name: {0} Age: {1}”, myAnonymousObject.Name, myAnonymousObject.Age);
var anotherAnonymousObject = new { Name = “Diana”, Age = 43 );
anotherAnonymousObject = myAnonymousObject;
```

Allijas

alias

```
using System;

using MyConsole = System.Console;

class UsingExample
{
  static void Main()
  {
    Console.WriteLine("Using the System.Console namespace.");
    MyConsole.WriteLine("So is this.");
   }
}
```

Korisne kolekcije

C#

Dictionary

```
var accounts = new Dictionary<string, double>();

// Initialise to zero...

accounts["Fred"] = 0;
accounts["George"] = 0;
accounts["Fred"] = 0;

// Add cash.
accounts["Fred"] += 4.56;
accounts["George"] += 1.00;
accounts["Fred"] += 1.00;

Console.WriteLine("Fred owes me ${0}", accounts["Fred"]);
```

Dictionary

```
public class Example
{
    public static void Main()
    {
           Dictionary<string, string> openWith = new Dictionary<string, string>();

// Add some elements to the dictionary. There are no duplicate keys, but some of the values are duplicates.
        openWith.Add("txt", "notepad.exe");
        openWith.Add("bmp", "paint.exe");
        openWith.Add("dib", "paint.exe");
        openWith.Add("rtf", "wordpad.exe");

// The Add method throws an exception if the new key is already in the dictionary.
        try
           {
                openWith.Add("txt", "winword.exe");
           }
        catch (ArgumentException)
           {
            Console.WriteLine("An element with Key = \"txt\" already exists.");
           }

         Console.WriteLine("For key = \"rtf\", value = {0}.", openWith["rtf"]);

// The indexer can be used to change the value associated with a key.
        openWith["rtf"] = "winword.exe";
        Console.WriteLine("For key = \"rtf\", value = {0}.", openWith["rtf"]);

// If a key does not exist, setting the indexer for that key adds a new key/value pair.
        openWith["doc"] = "winword.exe";
```

Dictionary

```
// The indexer throws an exception if the requested key is not in the dictionary.
        try
        {
            Console.WriteLine("For key = \"tif\", value = {0}.", openWith["tif"]);
        }
        catch (KeyNotFoundException)
        {
            Console.WriteLine("Key = \"tif\" is not found.");
        }

// When a program often has to try keys that turn out not to be in the dictionary, TryGetValue can be a more
efficient way to retrieve values.

         string value = "";
        if (openWith.TryGetValue("tif", out value))
        {
            Console.WriteLine("For key = \"tif\", value = {0}.", value);
        }
        else
        {
            Console.WriteLine("Key = \"tif\" is not found.");
        }

// ContainsKey can be used to test keys before inserting them.
        if (!openWith.ContainsKey("ht"))
        {
            openWith.Add("ht", "hypertrm.exe");
            Console.WriteLine("Value added for key = \"ht\": {0}",
                openWith["ht"]);
        }
```

Dictionary

```
// When you use foreach to enumerate dictionary elements, the elements are retrieved as KeyValuePair objects.
        Console.WriteLine();
        foreach( KeyValuePair<string, string> kvp in openWith )
        {
            Console.WriteLine("Key = {0}, Value = {1}", kvp.Key, kvp.Value);
        }


// The elements of the ValueCollection are strongly typed with the type that was specified for dictionary values.
        Console.WriteLine();
        foreach( string s in valueColl )
        {
            Console.WriteLine("Value = {0}", s);
        }

// To get the keys alone, use the Keys property.
        Dictionary<string, string>.KeyCollection keyColl = openWith.Keys;

// The elements of the KeyCollection are strongly typed with the type that was specified for dictionary keys.
        Console.WriteLine();
        foreach( string s in keyColl )
        {
            Console.WriteLine("Key = {0}", s);
        }

// Use the Remove method to remove a key/value pair.
        Console.WriteLine("\nRemove(\"doc\")");
        openWith.Remove("doc");
}
}
```

System.Collections Classes

Class	Description
ArrayList	Represents an array of objects whose size is dynamically increased as required.
Hashtable	Represents a collection of key/value pairs that are organized based on the hash code of the key.
Queue	Represents a first in, first out (FIFO) collection of objects.
Stack	Represents a last in, first out (LIFO) collection of objects.

System.Collections.Generic

Class	Description
Dictionary<TKey, TValue>	Represents a collection of key/value pairs that are organized based on the key.
List<T>	Represents a list of objects that can be accessed by index. Provides methods to search, sort, and modify lists.
Queue<T>	Represents a first in, first out (FIFO) collection of objects.
SortedList<TKey, TValue>	Represents a collection of key/value pairs that are sorted by key based on the associated IComparer<T> implementation.
Stack<T>	Represents a last in, first out (LIFO) collection of objects.

List

```
 List<string> colors = new List<string>();
    colors.Add("Red");
    colors.Add("Blue");
    colors.Add("Green");

foreach (string color in colors)
    {
        MessageBox.Show(color);
    }

 for (int i = 0; i < colors.Count; i++)
    {
        MessageBox.Show(colors[i]);
    }

colors.Insert(1, "violet");
colors.Sort();
colors.Remove("violet");

if (colors.Contains("Blue"))
    {
        MessageBox.Show("Blue color exist in the list");
    }

arrlist.Clear ();

string[] arr = colors.ToArray();
```

```
string combindedString = string.Join(",", colors);
```