INDEKSER

Sledeći primer pokazuje način definisanja privatnog niza, temps, i indeksera. Indexer omogućava direktan pristup privatnom nizu temp, preko reference kao tempRecord[i].

```
class TempRecord
{
    // Array of temperature values
    private double[] temps = new double[5] { 30, 30, 30, 30, 30 };

    // To enable client code to validate input
    // when accessing your indexer.
    public int Length
    {
        get { return temps.Length; }
    }

    // Indexer declaration.
    // If index is out of range, the temps array will throw the exception.
    public double this[int index]
    {
        get
        {
            return temps[index];
        }

        set
        {
            temps[index] = value;
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        TempRecord tempRecord = new TempRecord();

        // Use the indexer's set accessor
        tempRecord[1] = 25;
        tempRecord[2] = 25.5;

        // Use the indexer's get accessor
        for (int i = 0; i < tempRecord.Length; i++)
        {
            System.Console.WriteLine("Element #{0} = {1}", i, tempRecord[i]);
        }

        // Keep the console window open in debug mode.
        System.Console.WriteLine("Press any key to exit.");
        System.Console.ReadKey();
    }
}
```

PODRŠKA ZA FOREACH PETLJU

U prethodnom primeru smo za štampanje svih elemenata niza ***temp***, u klasi **TempRecord**, koristili ***for*** petlju i indekser.

```
            // Use the indexer's get accessor
            for (int i = 0; i < tempRecord.Length; i++)
            {
                System.Console.WriteLine("Element #{0} = {1}", i, tempRecord[i]);
            }
```

Sledeći primeri pokazuju način na koji možemo omogućiti iterisanje kroz vrednosti niza temp uz pomoć ***foreach*** petlje:

```
            foreach (double vrednostIzTempNiza in tempRecord)
            {
                System.Console.WriteLine("Vrednost = {0}", vrednostIzTempNiza);
            }
```

PRIMER 1 – IENUMERABLE & IENUMERATOR

```
    class TempRecord : IEnumerable
    {
        // Array of temperature values
        private double[] temps = new double[5] { 30, 30, 30, 30, 30 };

        // To enable client code to validate input
        // when accessing your indexer.
        public int Length
        {
            get { return temps.Length; }
        }

        // Indexer declaration.
        // If index is out of range, the temps array will throw the exception.
        public double this[int index]
        {
            get
            {
                return temps[index];
            }

            set
            {
                temps[index] = value;
            }
        }

        public IEnumerator GetEnumerator()
        {
            return new EnumeratorTempRecord(this);
        }
    }
```

```
class EnumeratorTempRecord : IEnumerator
{
    TempRecord tempRecord;
    int trenutnaPozicijaUNizu;

    public EnumeratorTempRecord(TempRecord tempRecord)
    {
        this.tempRecord = tempRecord;
        trenutnaPozicijaUNizu = -1;
    }

    public object Current
    {
        get { return tempRecord[trenutnaPozicijaUNizu]; }
    }

    public bool MoveNext()
    {
        trenutnaPozicijaUNizu++;
        if (trenutnaPozicijaUNizu < tempRecord.Length)
            return true;
        else return false;
    }

    public void Reset()
    {
        trenutnaPozicijaUNizu = -1;
    }
}

class Program
{
    static void Main(string[] args)
    {
        TempRecord tempRecord = new TempRecord();

        // Use the indexer's set accessor
        tempRecord[1] = 25;
        tempRecord[2] = 25.5;

        foreach (double vrednostIzTempNiza in tempRecord)
        {
            System.Console.WriteLine("Vrednost = {0}", vrednostIzTempNiza);
        }

        // Keep the console window open in debug mode.
        System.Console.WriteLine("Press any key to exit.");
        System.Console.ReadKey();
    }
}
```

PRIMER 2 - **YIELD RETURN**

```
class TempRecord
{
    // Array of temperature values
    private double[] temps = new double[5] { 30, 30, 30, 30, 30 };

    // To enable client code to validate input
    // when accessing your indexer.
    public int Length
    {
        get { return temps.Length; }
    }

    // Indexer declaration.
    // If index is out of range, the temps array will throw the exception.
    public double this[int index]
    {
        get
        {
            return temps[index];
        }

        set
        {
            temps[index] = value;
        }
    }

    public IEnumerable<double> MojEnumerator()
    {
        for (int i = 0; i < Length; i++)
        {
            yield return temps[i];
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        TempRecord tempRecord = new TempRecord();

        // Use the indexer's set accessor
        tempRecord[1] = 25;
        tempRecord[2] = 25.5;

        foreach (double vrednostIzTempNiza in tempRecord.MojEnumerator())
        {
            System.Console.WriteLine("Vrednost = {0}", vrednostIzTempNiza);
        }

        // Keep the console window open in debug mode.
        System.Console.WriteLine("Press any key to exit.");
        System.Console.ReadKey();
    }
```

```
}
```

DELEGATI (POKAZIVAČI NA FUNKCIJE)

Jedna bitna stvar koju Java nema je mogućnost da promenljive pokazuju na funkcije i da budu izvršene. Ovakve pokazivačke promenljive postoje u još nekim jezicima iz C familije. U C# jeziku pokazivači na funkcije se nazivaju delegati. Primer delegata je prikazan u sledećem listingu:

```
public delegate double AritmetickaFunkcija(double x, double y);

public static double Saberi(double x, double y)
{
    return x + y;
}

public static int Oduzmi(int x, int y)
{
    return x - y;
}
```

U ovom kodu je definisan jedan tip delegata (`AritmetickaFunkcija`) koji predstavlja pokazivač na bilo koju funkciju koja prima dva celobrojna argumenta i vraća celobrojni rezultat. Primer takvih funkcija je prikazan u istom listingu. Sada možemo da definišemo promenljive delegatskog tipa koje će biti pokazivači na ove dve funkcije. Primer je prikazan u sledećem listingu:

```
AritmetickaFunkcija pokazivac = Saberi;
Console.WriteLine(pokazivac(3, 5));
pokazivac = Oduzmi;
Console.WriteLine(pokazivac(3, 5));
```

Prvo je definisana promenljiva pokazivac tipa `AritmetickaFunkcija` koja pokazuje na funkciju Saberi. Pošto je ova promenljiva funkcija, ona se može "izvršiti" tako što joj se proslede dva argumenta i ona će delegirati poziv funkciji na koju pokazuje. Kao rezultat će se izvršiti funkcija Saberi sa parametrima 3 i 5. Potom pokazivac pokazuje na funkciju Oduzmi i ponovo se izvršava sa istim argumentima. Kao rezultat će se poziv delegirati funkciji Oduzmi.

Delegati se mogu simulirati u javi kao interfejsi koji sadrže jedan metod medutim ovo je malo elegantniji način rada ako je to potrebno.

EVENTS

Događaji u .NET okviru su bazirani na modelu delegata. Model delegata prati *observer pattern*, koji omogućava pretplatniku da se registruje i prima obaveštenja od provajdera.

Događaj predstavlja poruku koja se šalje od strane objekta (provajdera) kada se desi određena akcija, razlog za podizanje događaja može biti klik na dugme, ili promenom vrednosti neke promenljive. Objekat koji podiže događaj se zove *event sender*.

Događaji predstavljaju posebnu vrstu višeznačnih delegata. Kod događaja uvek imamo:

a) Klasa koja generiše događaj (**generator događaja**)
b) Klasa koja želi da bude obaveštena o nekom događaju (**potrošač**)

PRIMER 1

DEFINISANJE PROBLEMA

Neka je data definicija klase koja prati promenu temperature u vazduhu:

```
public class Termometar
{
    float trenutnaTemperatura = 10;

    void MeriTemperaturu()
    {
        Random r = new Random();

        while (true)
        {
            Thread.Sleep(r.Next(4000));

            if (r.Next(2) == 1)
                trenutnaTemperatura += 1;
            else
                trenutnaTemperatura -= 1;

        }
    }

}
```

Instanciranjem klase `Termometar t = new Termometar();` temperature će se menjati na svakih x sekundi, gde se x bira slučajnim izborom iz intervala [0-4].

```
class Program
{
    static void Main(string[] args)
    {
        Termometar t = new Termometar();
        t.MeriTemperaturu();
    }
}
```

Postavlja se pitanje na koji način možemo da ispisujemo u konzoli promenu temperature, jer ne znamo u kom trenutku je doslo do njene promene. U ovom slučaju je konzola potrošač, dok je `Termometar` proizvođač.

Prvo ćemo da napišemo funkciju koja će prosleđenu temperaturu ispisivati u konzoli:

```
class Program
{
    public static void PromenaTemperature(double temperatura)
    {
        Console.WriteLine("Temperatura je: " + temperatura);
    }


    static void Main(string[] args)
    {
        Termometar t = new Termometar();
        t.MeriTemperaturu();
    }
}
```

KREIRANJE MEHANIZMA DOGAĐAJA

Idealno je da se funkcija `PromenaTemperature` poziva samo kada se promena i desi. U nastavku je prikazana klasa `Termometar` koja ima ugrađen mehanizam događaja.

```
    public delegate void TemperaturaJePromenjenaHandler(double temperature);

    public class Termometar
    {
        public event TemperaturaJePromenjenaHandler TemperaturaJePromenjena;

        float trenutnaTemperatura = 10;

        public void MeriTemperaturu()
        {
            Random r = new Random();

            while (true)
            {
                Thread.Sleep(r.Next(4000));

                if (r.Next(2) == 1)
                    trenutnaTemperatura += 1;
                else
                    trenutnaTemperatura -= 1;

                //podigni dogadjaj promene temperature
                if (TemperaturaJePromenjena != null)
                    TemperaturaJePromenjena(trenutnaTemperatura);
            }
        }

    }
```

Prvo je kreiran delegat:

```
public delegate void TemperaturaJePromenjenaHandler(double temperature);
```

a zatim je kreiran **event** koji je tipa `TemperaturaJePromenjenaHandler:`

```
public event TemperaturaJePromenjenaHandler TemperaturaJePromenjena;
```

Sledeći korak je podizanje događaja, u slučaju da je neko na njega prijavljen:

```
//podigni dogadjaj promene temperature
if (TemperaturaJePromenjena != null)
      TemperaturaJePromenjena(trenutnaTemperatura);
```

Drugim rečima, na event se mogu prijaviti samo funkcije koje imaju povratni tip ***void*** i jedan ***double*** ulazni parametar.

PRIJAVA NA DOGAĐAJ

```
class Program
{
    public static void PromenaTemperature(double temperatura)
    {
        Console.WriteLine("Temperatura je: " + temperatura);
    }


    static void Main(string[] args)
    {
        Termometar t = new Termometar();

        //Prijava metode PromenaTemperature na event TemperaturaJePromenjena
        t.TemperaturaJePromenjena += PromenaTemperature;

        t.MeriTemperaturu();
    }
}
```

Metod `PromenaTemperature` je prijavljen na događaj naredbom:

```
t.TemperaturaJePromenjena += PromenaTemperature;
```

Ovo ima za rezultat poziv funkcije `PromenaTemperature` kada god se izvrši podizanje događaja u klasi `Termometar`:

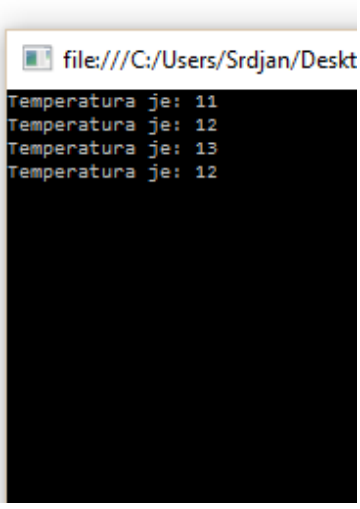
```
TemperaturaJePromenjena(trenutnaTemperatura);
```

```
class Program
{
    public static void PromenaTemperature(double temperatura)
    {
        Console.WriteLine("Temperatura je: " + temperatura);
    }

    static void Main(string[] args)
    {
        Termometar t = new Termometar();

        t.TemperaturaJePromenjena += PromenaTemperature;

        t.MeriTemperaturu();
    }
}
```



Slika 1 – Rezltat pokretanja programa sa jednom prijavljenom metodom

DVE PRIJAVLJENE FUNKCIJE

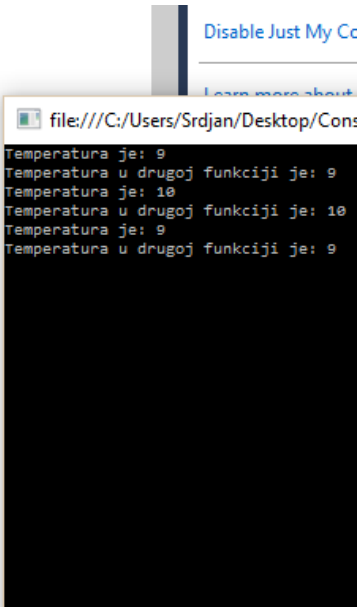
```
class Program
{
    public static void PromenaTemperature(double temperatura)
    {
        Console.WriteLine("Temperatura je: " + temperatura);
    }

    public static void PromenaTemperature_2(double temperatura)
    {
        Console.WriteLine("Temperatura u drugoj funkciji je: " + temperatura);
    }


    static void Main(string[] args)
    {
        Termometar t = new Termometar();

        t.TemperaturaJePromenjena += PromenaTemperature;
        t.TemperaturaJePromenjena += PromenaTemperature_2;

        t.MeriTemperaturu();
    }
}
```



Slika 2 – Rezultat sa dve prijavljene funkcije