OVERLOADING OPERATORA

U ovom odeljku su dati primeri koji prikazuju način na koji je moguće eksplicitno i implicitno kastovanje jednog tipa u drugi, kao i overloading operatora +, -, *, /, ==, !=...

U nastavku su data bliža objašnjenja o primerima u nastavku.

- Eksplicitno kastovanje:

  ```
  Tip1 t1 = new Tip1();
  Tip2 t2 = (Tip2)t1;
  ```

- Implicitno kastovanje:

  ```
  Tip1 t1;
  Int i= 1;
  t1 = i; // vrednost promenljive i se implicitnim kastovanjem dodeljuje promenljivoj t1 tipa Tip1.
  ```

- Operator overloading

1.
   ```
   Tip1 t1 = new Tip1();
   Tip1 t2 = new Tip1();
   Tip1 t3 = t1 + t2;
   ```
2.
   ```
   Tacka2D t1 = new Tacka2D();
   Vektor2D t2 = new Vektor2D();
   Tacka2D t = t1 + t2;
   ```

EXPLICIT OPERATOR

PRIMER 1

```
class Celzijus
{
    private float stepeni;

    public float Stepeni
    {
        get { return stepeni; }
    }

    public Celzijus(float temp)
    {
        stepeni = temp;
    }

    public static explicit operator Farenhajt(Celzijus c)
    {
        return new Farenhajt((9.0f / 5.0f) * c.stepeni + 32);
    }
```

```
    }

    class Farenhajt
    {
        private float stepeni;

        public float Stepeni
        {
            get { return stepeni; }
        }

        public Farenhajt(float temp)
        {
            stepeni = temp;
        }

        public static explicit operator Celzijus(Farenhajt fahr)
        {
            return new Celzijus((5.0f / 9.0f) * (fahr.stepeni - 32));
        }


    }

    class Program
    {
        static void Main(string[] args)
        {

            // Primer sa temperaturama
            Farenhajt fahr = new Farenhajt(100.0f);
            Console.Write("{0} Farenhajt", fahr.Stepeni);
            Celzijus c = (Celzijus)fahr;

            Console.Write(" = {0} Celzijus", c.Stepeni);
            Farenhajt fahr2 = (Farenhajt)c;
            Console.WriteLine(" = {0} Farenhajt", fahr2.Stepeni);

        }
    }
```

PRIMER 2

```
    /// <summary>
    /// Reprezentacija cifre koja se cuva kao tip byte
    /// </summary>
    public struct Cifra
    {
        byte vrednost;

        public Cifra(byte vrednost)
        {
            // Ako je vrednost veca od 9 podigni gresku da argument nije u redu.
            if (vrednost > 9)
            {
                throw new ArgumentException("Vrednost " + vrednost + " prevazilazi
cifru.");
```

```
            }

            this.vrednost = vrednost;
        }

        // Definisi operator za eksplicitnu byte-to-Cifra konverziju.
        public static explicit operator Cifra(byte b)
        {
            Cifra d = new Cifra(b);
            Console.WriteLine("conversion occurred");
            return d;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Primer sa cifrom
            try
            {
                byte b = 3;
                // byte b = 10; // Primer u kome se podize greska
                Cifra d = (Cifra)b; // explicit conversion
            }
            catch (Exception e)
            {
                Console.WriteLine("{0} Exception caught.", e);
            }
        }
    }
```

IMPLICIT OPERATOR

```
    class Cifra
    {
        public double val;

        public Cifra(double d) { val = d; }

        // User-defined conversion from Digit to double
        public static implicit operator double(Cifra d)
        {
            return d.val;
        }

        //  User-defined conversion from double to Digit
        public static implicit operator Cifra(double d)
        {
            return new Cifra(d);
        }
    }

    class Program
    {
        static void Main(string[] args)
```

```
        {
             Cifra dig = new Cifra(7);

            //This call invokes the implicit "double" operator
            double num = dig;
            //This call invokes the implicit "Digit" operator
            Cifra dig2 = 12;
            Console.WriteLine("num = {0} dig2 = {1}", num, dig2.val);
            Console.ReadLine();
        }
    }
```

OPERATOR OVERLOADING

```
using System;

public struct Complex
{
   public int real;
   public int imaginary;

   public Complex(int real, int imaginary)
   {
      this.real = real;
      this.imaginary = imaginary;
   }

   // Declare which operator to overload (+), the types
   // that can be added (two Complex objects), and the
   // return type (Complex):
   public static Complex operator +(Complex c1, Complex c2)
   {
      return new Complex(c1.real + c2.real, c1.imaginary + c2.imaginary);
   }

   // Override the ToString method to display an complex number in the suitable format:
   public override string ToString()
   {
      return(String.Format("{0} + {1}i", real, imaginary));
   }

   public static void Main()
   {
      Complex num1 = new Complex(2,3);
      Complex num2 = new Complex(3,4);

      // Add two Complex objects (num1 and num2) through the
      // overloaded plus operator:
      Complex sum = num1 + num2;

     // Print the numbers and the sum using the overriden ToString method:
      Console.WriteLine("First complex number:  {0}",num1);
      Console.WriteLine("Second complex number: {0}",num2);
      Console.WriteLine("The sum of the two numbers: {0}",sum);

   }
}
```