Protected pravo pristupa

```
class ClassA
{
    public int a1;
    private int a2;

    public int A2
    {
        get { return a2; }
        set { a2 = value; }
    }
    protected int a3;

    public int A3
    {
        get { return a3; }
        set { a3 = value; }
    }
    public ClassA()
    {
        Console.WriteLine("Konstruktor bazne klase");
        a1 = 1;
        a2 = 2;
        a3 = 3;
        Console.WriteLine("----------------------------");
    }
    public virtual void Stampaj()
    {
        Console.WriteLine($"a1={a1} a2={a2} i a3={a3}");
        Console.WriteLine("----------------------------");
    }

}
```

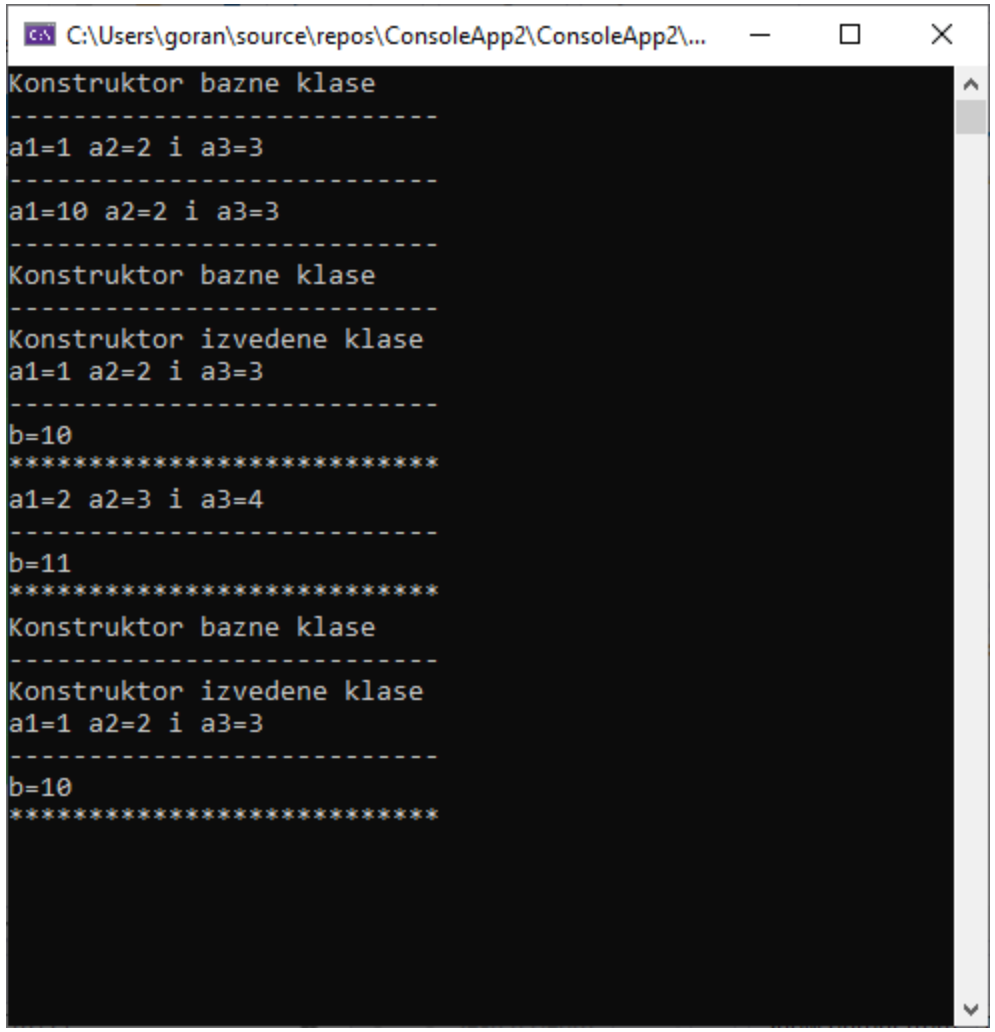
Bazna klasa

Izvedena klasa

```
class ClassB : ClassA
{
    public int B { get; set; }

    public ClassB()
    {
        Console.WriteLine("Konstruktor izvedene klase");
        B = 10;
    }

    public void Uvecaj()
    {
        a1++;
        A2++; // a2 je privatni clan bazne klase
        a3++; // a3 je protected
        B++;

    }
    public override void Stampaj()
    {
        base.Stampaj();
        Console.WriteLine($"b={B}");
        Console.WriteLine("**************************");
    }
}
```

Kreiranje objekata bazne i izvedene klase

```
static void Main(string[] args)
{
    ClassA inst1 = new ClassA();
    inst1.Stampaj();
    inst1.a1 = 10;
    inst1.Stampaj();


    ClassB inst2 = new ClassB();
    inst2.Stampaj();
    inst2.Uvecaj();
    inst2.Stampaj();

    // 1. pravilo polimorfizma
    ClassA inst3 = new ClassB();
    inst3.Stampaj();
    Console.ReadLine();
}
```



Apstraktne klase

Apstraktne klase

- **Apstraktna metoda** je metoda bez implementacije (nema telo metode)
- Klasa koja ima bar jednu apstraktnu metodu je **apstraktna klasa**
- Ispred apstraktnih metoda i apstraktnih klasa piše se ključna reč **abstract**
- Apstraktna klasa ne može da se instancira
- Apstraktna klasa može imati i metode koje nisu apstraktne(koje imaju implementaciju)
- Klasa izvedena iz apstraktne klase mora da realizuje (implementira) sve apstraktne metode
- Primer apstraktne klase je klasa DbConnection

Nasleđivanje apstraktne klase

```
abstract class A
{

    // apstraktna metoda
    public abstract void metoda1();
}
```

```
class B : A
{
    public override void metoda1()
    {
        throw new NotImplementedException();
    }
}
```

Apstraktna klasa Oblik

```
abstract class Oblik
{
    public string Ime { get; set; }

    public Oblik(string s)
    {
        Ime = s;
    }

    // apstraktna metoda
    public abstract double Povrsina();

    public override string ToString()
    {
        return $"{Ime} : Povrsina: {Povrsina()}";
    }
}
```

Klasa Kvadrat

```
class Kvadrat : Oblik
{
    private double a;
    public Kvadrat(string naziv, double a) : base(naziv)
    {
        this.a = a;
    }
    public override double Povrsina()
    {
        return a * a;
    }
}
```

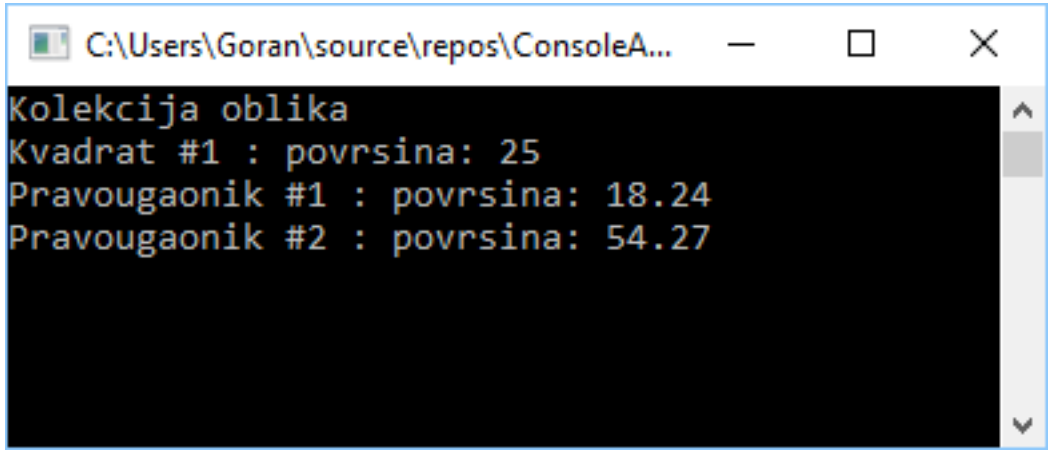
Klasa Pravougaonik

```
class Pravougaonik : Oblik
{
    private double a;
    private double b;
    public Pravougaonik(string naziv, double a, double b) : base(naziv)
    {
        this.a = a;
        this.b = b;
    }
   public override double Povrsina()
    {
          return a * b;
     }

}
```

Ilustracija pravila polimorfizma

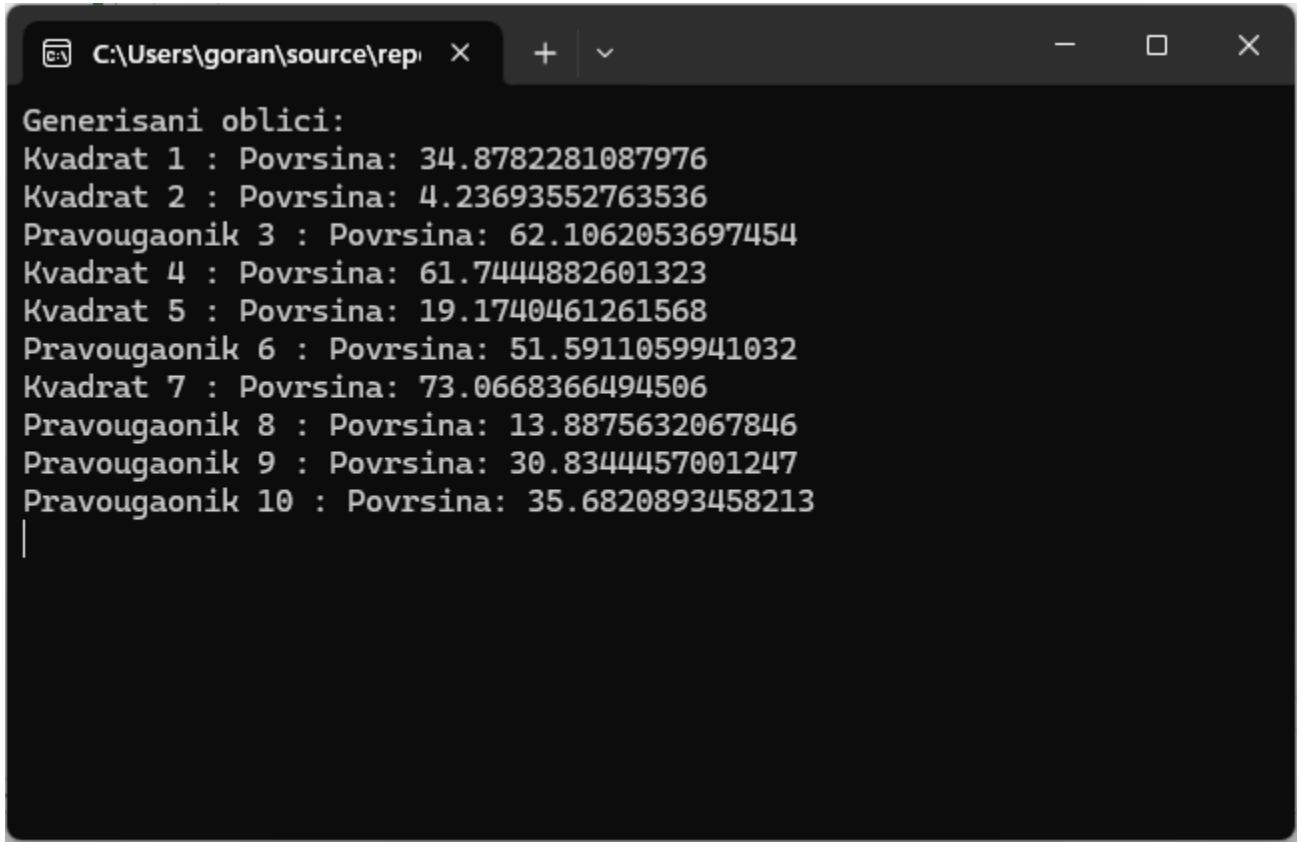
```
static void Main(string[] args)
{
    Oblik[] oblici ={
    new Kvadrat("Kvadrat #1",5),
    new Pravougaonik("Pravougaonik #1",3.2, 5.7),
    new Pravougaonik("Pravougaonik #2",6.7,8.1)
    };

    Console.WriteLine("Kolekcija oblika");
    foreach (Oblik s in oblici)
    {
        Console.WriteLine(s.ToString());
    }

    Console.ReadLine();
}
```

Rezultat izvršavanja koda



```
static void Main(string[] args)
{
    Random rnd = new Random();

    Oblik[] oblici = new Oblik[10];

    for (int i = 0; i < oblici.Length; i++)
    {
        int tip = rnd.Next(1, 3); // daje 1 ili 2

        if (tip == 1)
        {
            // Kvadrat sa slucajnom stranicom od 1 do 10
            double a = rnd.NextDouble() * 9 + 1;
            oblici[i] = new Kvadrat($"Kvadrat {i + 1}", a);
        }
        else
        {
            // Pravougaonik sa slucajnim stranicama
            double a = rnd.NextDouble() * 9 + 1;
            double b = rnd.NextDouble() * 9 + 1;
            oblici[i] = new Pravougaonik($"Pravougaonik {i + 1}", a, b);
        }
    }

    Console.WriteLine("Generisani oblici:");
    foreach (Oblik o in oblici)
    {
        Console.WriteLine(o.ToString());
    }

    Console.ReadLine();
}
```

Rezultat izvršavanja koda



Interfejsi

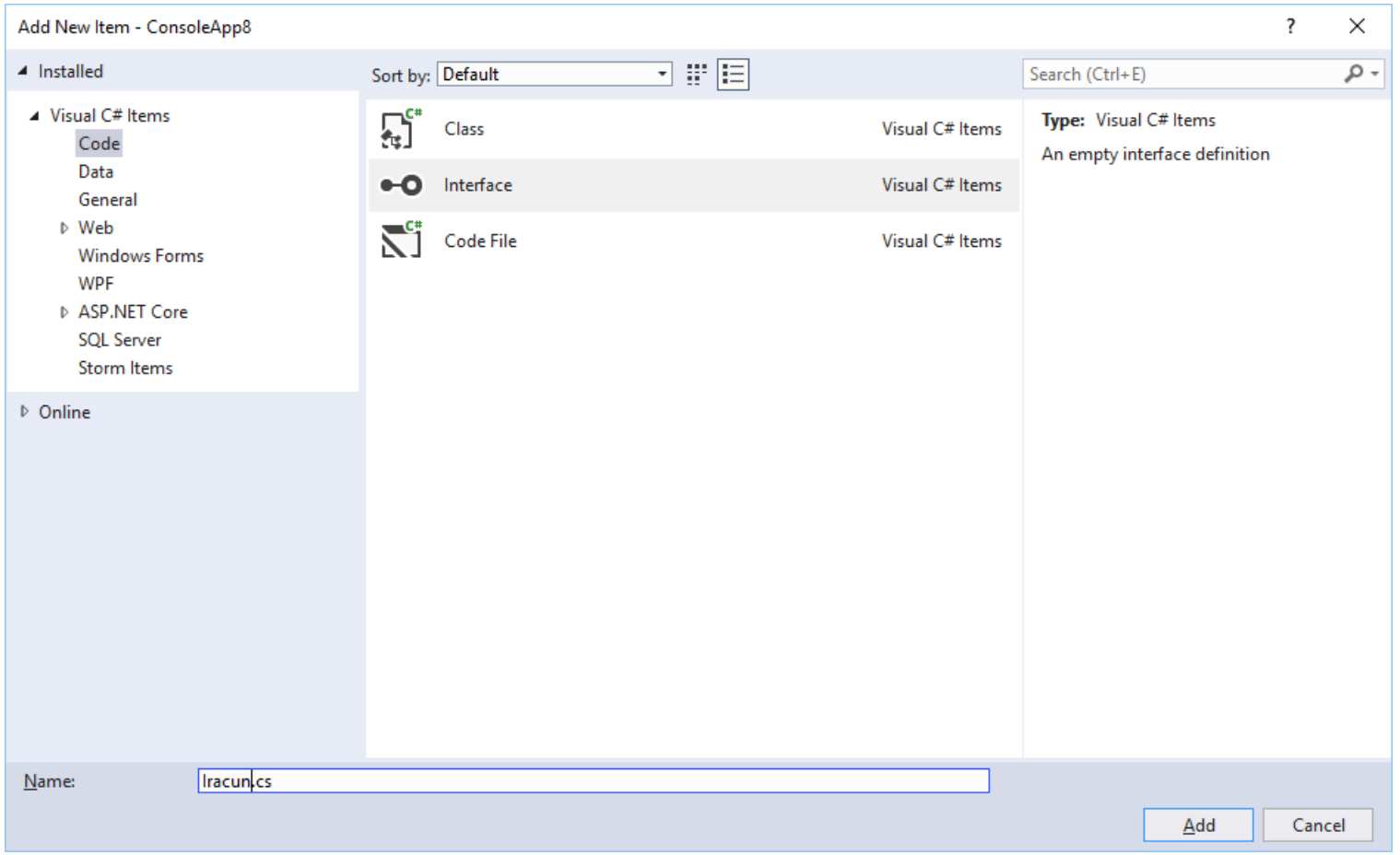
Pojam interfejsa

- Interfejs predstavlja ugovor: definiše koje metode neka klasa mora da ima
- Interfejs sadrži samo potpis metode (naziv, povratni tip, parametri), ali nema implementaciju
- Klasa koja implementira interfejs mora da napiše sve metode koje interfejs definiše
- Interfejs omogućava da različite klase imaju isti skup metoda, ali ih svaka implementira na svoj način
- Interfejs se ne može instancirati

Karakteristike interfejsa

- Različite klase mogu implementirati interfejs na različite načine
- Koristi se ključna reč ***interface*** pri definiciji interfejsa
- Interfejs može da nasledi više interfejsa
- Klasa može da implementira više interfejsa
- Metoda koja implementira metodu iz interfejsa mora biti javna
- Kod koji koristi interfejs ne mora da zna koja se konkretna klasa nalazi iza njega
- Primer interjfesa IDbConnection

Dodavanje interfejsa u VisualStudio okruženju



Primer interfejsa

```
interface IRacun
{
    void Uplata(double iznos);
    bool Isplata(double iznos);
    double Stanje { get; }
}
```

Implementacija interfejsa -1

```
class Racun : IRacun
{
    private string ime;
    private string prezime;
    private double stanje;

    public Racun(string ime, string prezime, double stanje)
    {
        this.ime = ime;
        this.prezime = prezime;
        this.stanje = stanje;
    }
    public double Stanje
    {
        get
        {
            return stanje;
        }
    }
```

Implementacija interfejsa -2

```
 public bool Isplata(double iznos)
    {
        if (iznos <= stanje)
        {
            stanje -= iznos;
            return true;
        }
        return false;
    }

    public void Uplata(double iznos)
    {
        stanje += iznos;
    }

    public void Stampaj()
    {
        Console.WriteLine($"{ime} {prezime}, Stanje: {stanje}");
    }
}
```

Pristup objektu klase posredstvom interfejsa

```
static void Main(string[] args)
{
    IRacun r1 = new Racun("Marko", "Markovic", 34567);

    r1.Uplata(23456);
    Console.WriteLine(r1.Stanje);

    bool b1 = r1.Isplata(56789);
    if (!b1)
    {
        Console.WriteLine("Isplata nije uspela – nedovoljno sredstava.");
    }

    Console.WriteLine(r1.Stanje);
    Console.WriteLine(new string('*', 30));

}
```

Kada koristimo referencu tipa interfejsa, pristupamo samo onome što je definisano u interfejsu

Pristup objektu klase preko reference

```
Racun r2 = new Racun("Jovan", "Markovic", 12345);
r2.Stampaj();
r2.Uplata(13456.1);
r2.Stampaj();
```

Kada koristimo referencu tipa klase, možemo pristupati svim metodama i poljima te klase

Eksplicitna implementacija interfejsa

- Kod eksplicitne implementacije metoda se implementira bez modifikatora pristupa (bez public, private…)
- Eksplicitno implementirane metode nisu dostupne preko naziva klase — dostupne su samo kroz interfejs.
- Metodama se pristupa isključivo preko reference tipa interfejsa

Eksplicitna implementacija interfejsa

```
double IRacun.Stanje
{
    get
    {
        return stanje;
    }
}

bool IRacun.Isplata(double iznos)
{
    if (iznos <= stanje)
    {
        stanje -= iznos;
        return true;
    }
    return false;
}

void IRacun.Uplata(double iznos)
{
    stanje += iznos;
}
```

Poziv ekplicitno implementirane metode interfejsa

```
static void Main(string[] args)
{
    Racun r1 = new Racun("Petar", "Markovic", 12345);
    //r1.Uplata(); // ne moze

    IRacun r2 = new Racun1("Petar", "Markovic", 12345);
    r2.Uplata(45678.1);
}
```

Implementacija više interfejsa

```
internal interface IEngleskeDimenzije
{
    float Duzina();
    float Sirina();
}
```

```
internal interface IMetrickeDimenzije
{
    float Duzina();
    float Sirina();
}
```

```
 class Pravougaonik : IEngleskeDimenzije, IMetrickeDimenzije
{
    float duzinaInch;
    float sirinaInch;

    public Pravougaonik(float duzinaInch, float sirinaInch)
    {
        this.duzinaInch = duzinaInch;
        this.sirinaInch = sirinaInch;
    }

    // IMetrickeDimenzije - cm
    float IMetrickeDimenzije.Duzina()
    {
        return duzinaInch * 2.54f;
    }

    float IMetrickeDimenzije.Sirina()
    {
        return sirinaInch * 2.54f;
    }

    // IEngleskeDimenzije - inch
    float IEngleskeDimenzije.Duzina()
    {
        return duzinaInch;
    }

    float IEngleskeDimenzije.Sirina()
    {
        return sirinaInch;
    }
}
```

Poziv metoda iz interfejsa

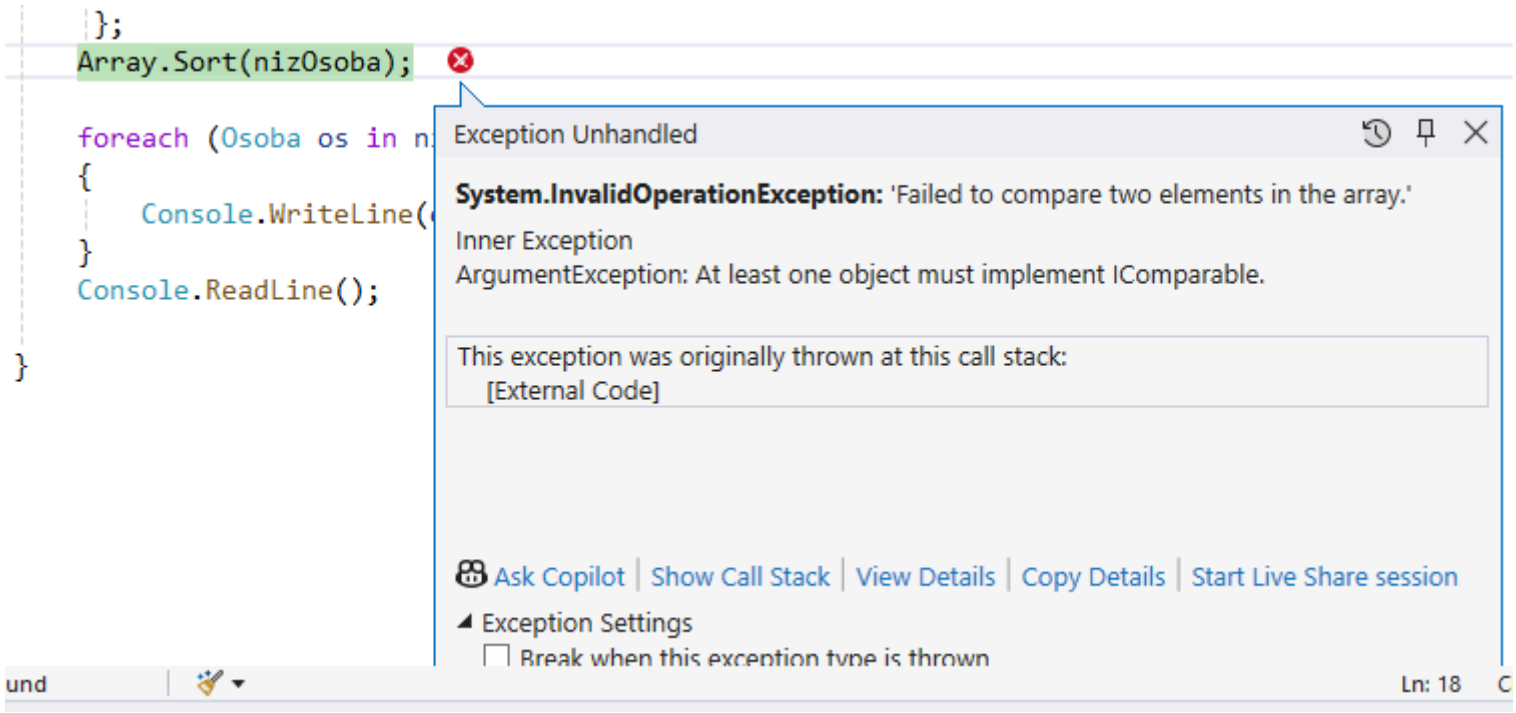
```
static void Main(string[] args)
{
    Pravougaonik p = new Pravougaonik(2.1f, 3.2f);

    IEngleskeDimenzije ieng = p;
    Console.WriteLine($"Engleske dimenzije: sirina {ieng.Sirina()} duzina {ieng.Duzina()}");

    IMetrickeDimenzije imet = p;
    Console.WriteLine($"Metricke dimenzije: sirina {imet.Sirina()} duzina {imet.Duzina()}");

}
```

Sortiranje niza objekata

```
class Osoba
{
    public string Ime { get; set; }
    public string Prezime { get; set; }
}
```

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

    Osoba[] nizOsoba = {
        new Osoba { Ime = "Pera", Prezime = "Peric" },
        new Osoba { Ime = "Mika", Prezime = "Mikic" },
        new Osoba { Ime = "Laza", Prezime = "Lazic" }
    };
    Array.Sort(nizOsoba);

    foreach (Osoba os in nizOsoba)
    {
        Console.WriteLine(os.Ime + " " + os.Prezime);
    }
    Console.ReadLine();

}
```

Izuzetak

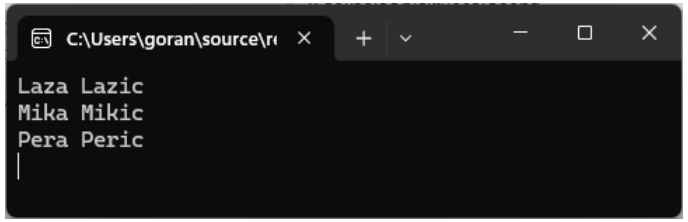


Pokušaj sortiranja niza objekata klase Osoba izaziva izuzetak, jer klasa ne implementira sistemski interfejs IComparable
Array.Sort može da poredi samo objekte koji implementiraju ovaj interfejs

Implementacija IComparable interfejsa

```
internal class Osoba : IComparable<Osoba>
{
    public string Ime { get; set; }
    public string Prezime { get; set; }

    public int CompareTo(Osoba other)
    {
        throw new NotImplementedException();
    }
}
```

```
internal class Osoba : IComparable<Osoba>
{
    public string Ime { get; set; }
    public string Prezime { get; set; }

    public int CompareTo(Osoba other)
    {
        return this.Prezime.CompareTo(other.Prezime);
    }
}
```



Pitanje 1

Abstraktna metoda klase :

a. nema telo

b. ima telo ali prazno

c. ima telo u kome se može nalaziti kod

Odgovor: a

Pitanje 2

Klasa **Racun** ima konstruktor bez parametara i implementira interfejs **IRacun**.
Šta je dozvoljeno pisati:

a. Iracun i = new Iracun();
b. Iracun i = new Racun();
c. Racun r = new Iracun();

Odgovor: b

Pitanje 3

Kako se pristupa metodi klase iz interfejsa ako klasa eksplicitno implementira interfejs?

a. Preko reference tipa klase
b. Preko reference tipa interfejsa
c. Preko reference tipa Explicit

Odgovor: b

Pitanje 4

Šta je tačno u vezi sa protected modifikatorom pristupa u C#?

a. Članovi sa protected pristupom su dostupni samo unutar klase u kojoj su definisani
b. Članovi sa protected pristupom su dostupni unutar klase u kojoj su definisani i u svim klasama koje nasleđuju tu klasu
c. Članovi sa protected pristupom su dostupni samo unutar istog paketa

Odgovor: b

Pitanje 5

Klasa **Dokument** implementira interfejs **IStampanje**, ali takođe ima i dodatnu metodu **Arhiviraj()** koja nije deo interfejsa.
Data je sledeći kod: IStampanje d = new Dokument();

a. Preko promenljive d može se pristupiti svim metodama klase Dokument
b. Preko promenljive d može se pristupiti samo metodama definisanim u interfejsu IStampanje ali ne i metodi Arhiviraj
c. Preko promenljive d se može pozvati samo metoda Arhiviraj

Odgovor: b