Enumeracije

Enumeracije

- Enumeracioni tip specificira skup imenovanih konstanti
- Prednosti korišćenja enumeracija:
  - Lakše je održavanje koda jer se promenljivama dodeljuju samo očekivane vrednosti
  - Kod je lakše čitati jer se vrednostima dodeljuju imena koja je lako identifikovati
  - Lakše je pisanje koda jer IntelliSense prikazuje listu mogućih vrednosti koju možemo koristiti

Enumercijski tip

```
class Program
{
    enum Dan
    {
        Ponedeljak,
        Utorak,
        Sreda,
        Cetvrtak,
        Petak,
        Subota,
        Nedelja
    }
    static void Main(string[] args)
    {
    }
}
```

Definiše se unutar klase ili unutar namespace-a.
Ne unutar metode.

```
static void Main(string[] args)
    {
        Dan d = Dan.Petak;
        Console.WriteLine(d);

        int n = (int)d;
        Console.WriteLine(n);
        Console.ReadLine();
    }
```

Enumercijski tip

```
class Program
{
    enum Dan
    {
        Ponedeljak = 1,
        Utorak,
        Sreda,
        Cetvrtak,
        Petak,
        Subota,
        Nedelja
    }

    static void Main(string[] args)
    {
        Console.WriteLine("Unesite redni broj dana u nedelji 1-7");
        int redniBroj = int.Parse(Console.ReadLine());
        Dan unetiDan = (Dan)redniBroj;
        Console.WriteLine(unetiDan);
        Console.ReadLine();
    }
}
```

Pitanje 1

Enumeracioni tip podataka se ne može definisati unutar:

a. klase
b. namespace
c. metode

Odgovor: c

Pitanje 2

Šta se ispisuje kao rezultat izvršavanja Main funkcije:

```
class Program
{
        enum Doba { Prolece, Leto, Jesen, Zima }

        static void Main(string[] args)
        {
            Console.WriteLine((int)Doba.Leto);
            Console.ReadLine();
        }

    }
}
```

a. 0
b. 1
c. 2

Odgovor: b

Pitanje 3

Šta se ispisuje kao rezultat izvršavanja Main funkcije:

```
enum Doba { Prolece, Leto, Jesen, Zima }

static void Main(string[] args)
{
    Doba d = (Doba)1;
    Console.WriteLine(d);
    Console.ReadLine();
}
```

a. 0
b. Leto
c. 1

Odgovor: b

Definisanje klase i kreiranje objekata

Osnovni koncepti OOP-a

- Apstraktni tipovi podataka
- Enkapsulacija
- Nasleđivanje
- Polimorfizam

Apstraktni tipovi podataka

- Objekti iz realnog sveta se modeluju njihovom ponašanjem sa stanovišta korisnika i ovaj koncept se naziva **abstrakcija**
- Pri abstrakciji sakrivaju se nepotrebni detalji objekata sa stanovišta korisnika
- **Abstraktni** tip podataka je tip podataka kojim se predstavljaju objekti
- Ugrađeni (osnovni, primitivni tipovi podataka) npr. **float**, **int**, **double...**
- Ravnopravno se definišu korisnički definisani tipovi – apstraktni tipovi podataka: TekuciRacun, Osoba, Student, …
- Proizvoljan broj primeraka nekog tipa i mogu se vršiti operacije nad njima

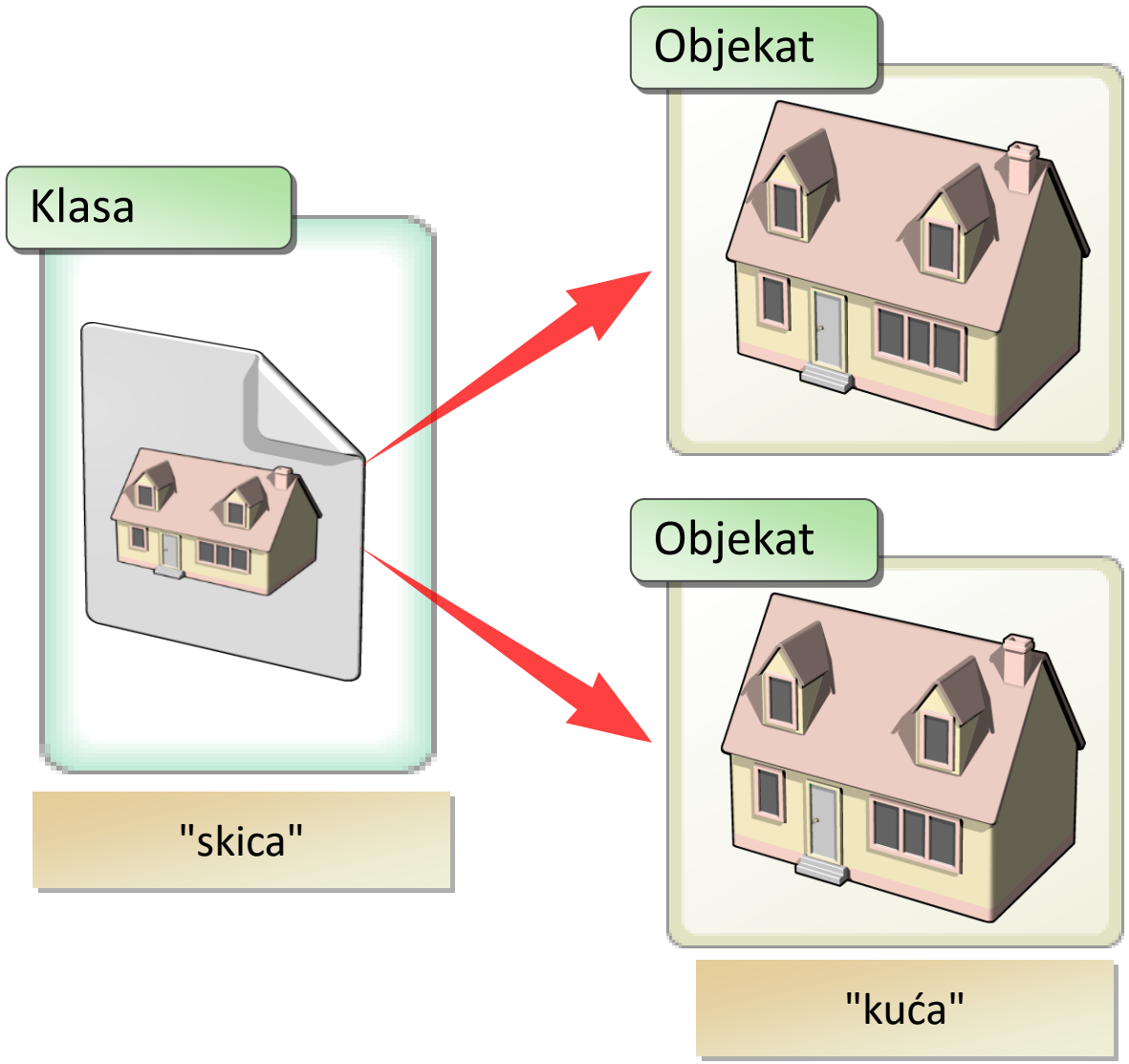
Enkapsulacija

- Enkapsulacija je sposobnost objekta da skriva svoje unutrašnje podatke i implementacione detalje
- Grupisanje podataka i koda koji manipuliše podacima
- Enkapsulacija se ostvaruje korišćenjem klase kao novog tipa podataka
- Realizija nekog tipa podataka može i treba da se sakrije od ostatka sistema tj. onih koji ga koriste
- Korisnicima se definiše šta se sa tipom može uraditi a način na koji se to radi se skriva

Odnos klase i objekata

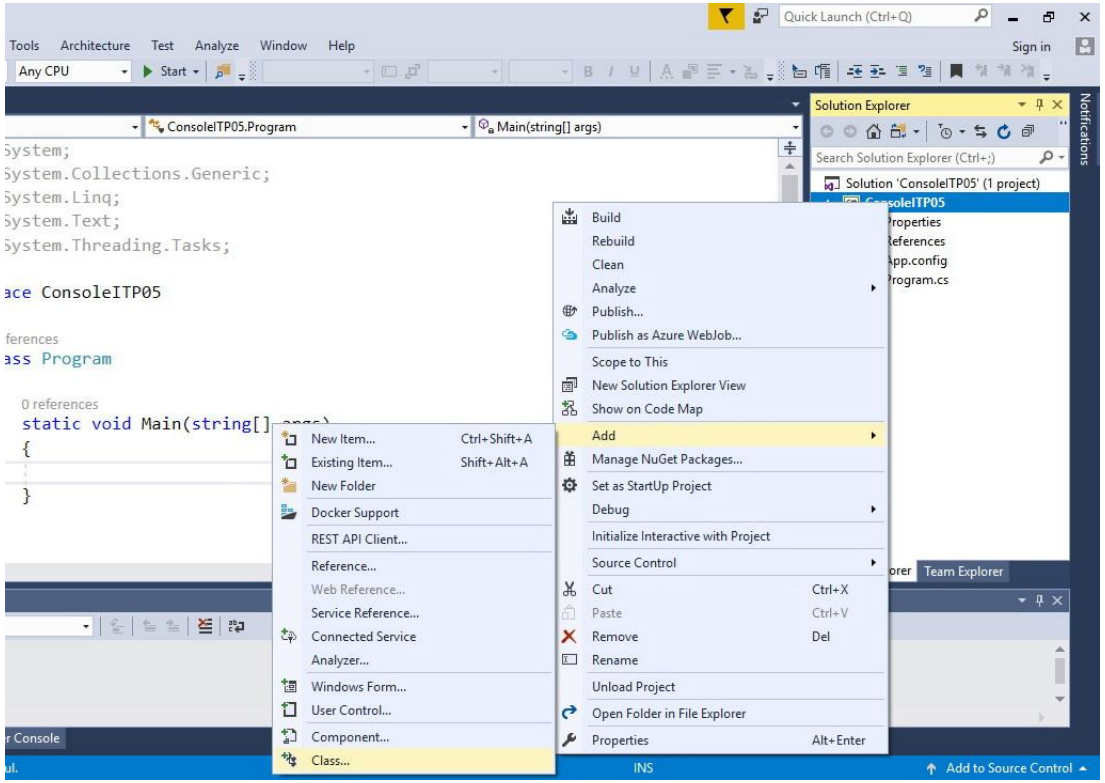
- Klasa :
  - To je model koji opisuje kako kreirati objekat
  - je kao "šematski plan(skica)"
  - Sadrži podatke (polja) i metode
- Objekti:
  - Objekat je predstava nekog entiteta iz realnog sveta
  - Instance klase
  - Može biti više objekata (instanci) klase

Primer klase i objekata



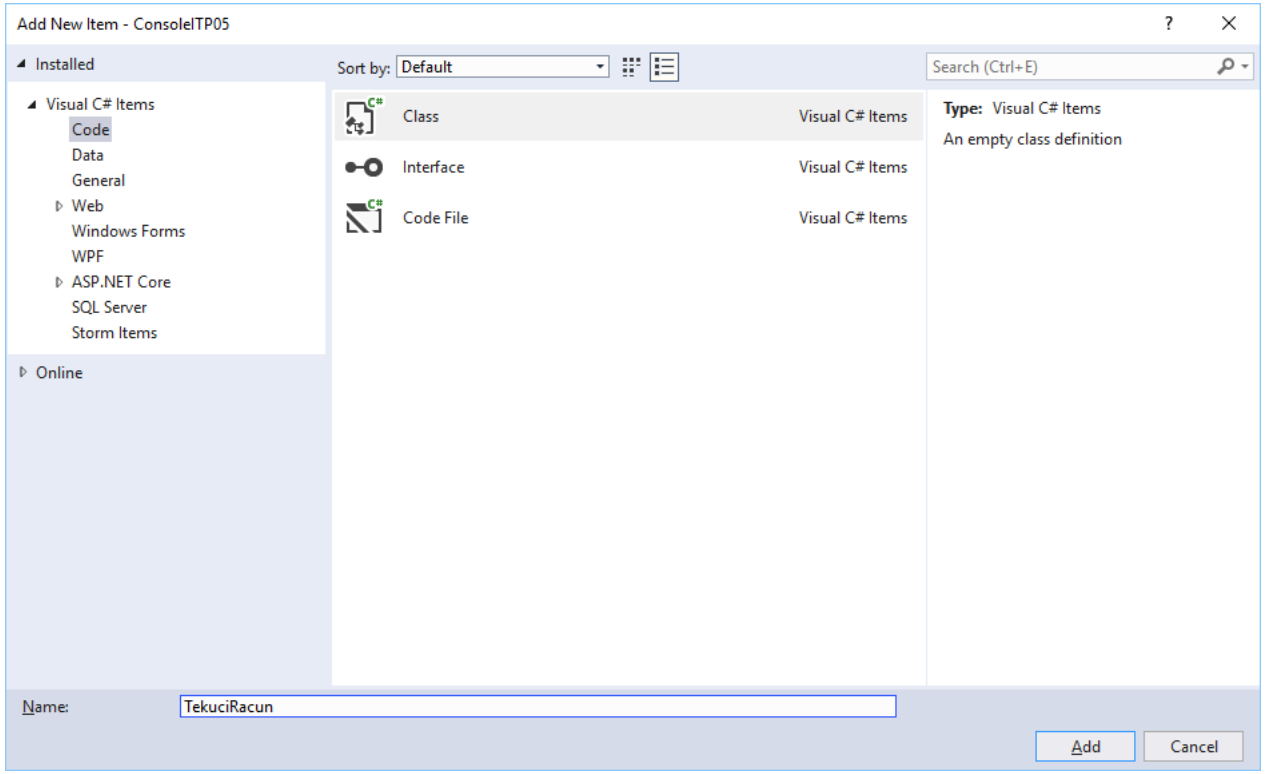
Funkcionalnosti enkapsulirane unutar klase

Dodavanje klase u Visual Studio 2022 okruženju



Desni klik na naziv projekta

Dodavanje klase u Visual Studio 2022 okruženju



Definisanje klase

```
internal class TekuciRacun
{
    public string ime;
    public string prezime;
    public double stanje;
}
```

Kreiranje objekata

- Objekti su inicijalno neoznačeni
- Podrazumevana vrednost objekta je null
- Da bi se koristila promenljiva tipa klase, klasa se mora instancirati
- Nova instanca klase kreira se korišćenjem operatora **new**

Kreiranje objekata

- Operator new prouzrokuje da:
  - CLR alocira memoriju za objekat
  - poziva konstruktor da inicijalizuje objekat
- Da bi se pristupilo javnom članu klase koristi se ime instance iza koga sledi operator tačka

Instanciranje klase (kreiranje objekata)

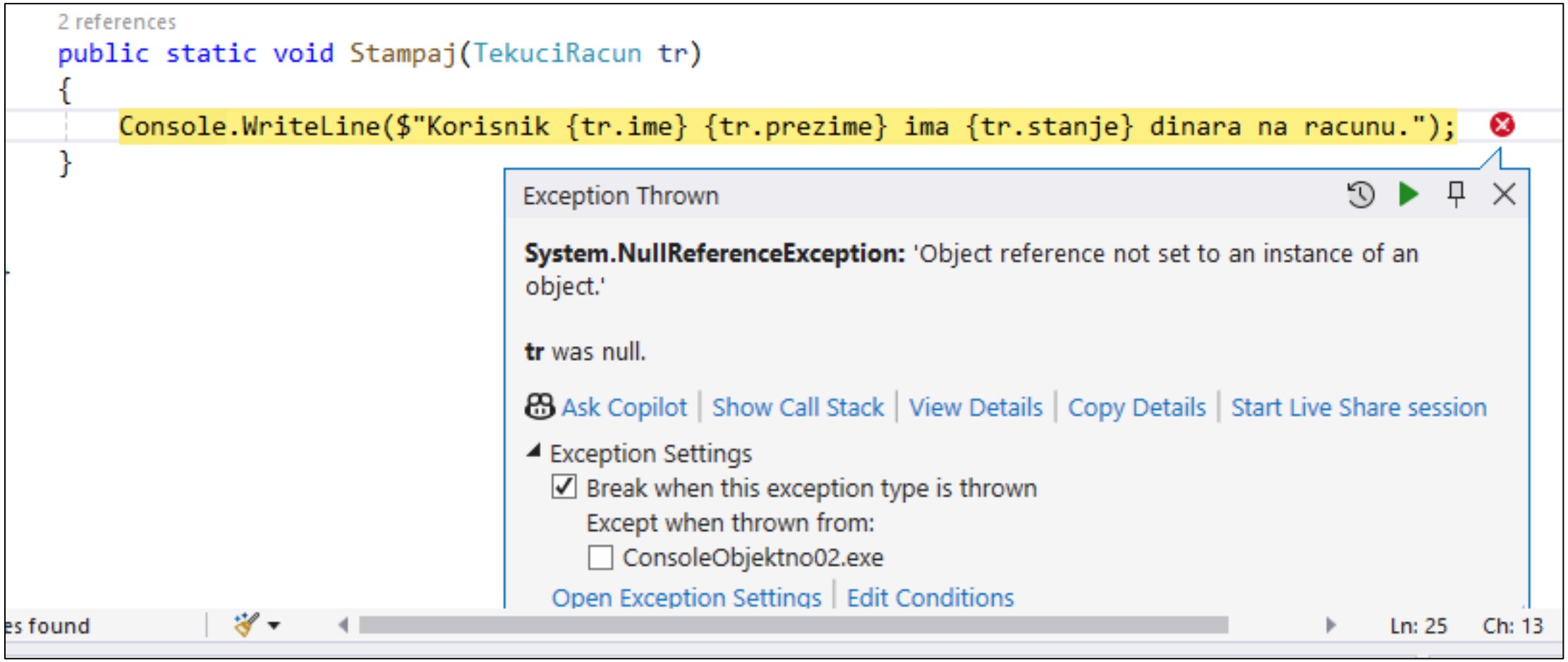
```
static void Main(string[] args)
{
    TekuciRacun tr1 = new TekuciRacun();
    tr1.ime = "Pera";
    tr1.prezime = "Peric";
    tr1.stanje = 12345.34;
    Console.WriteLine($"Korisnik {tr1.ime} {tr1.prezime} ima {tr1.stanje}
    dinara na racunu.");
    Console.ReadLine();
}
```

Dodeljivanje null reference

```
public static void Stampaj(TekuciRacun tr)
{
    Console.WriteLine($"Korisnik {tr.ime} {tr.prezime} ima {tr.stanje}
    dinara na racunu.");
}
```

```
static void Main(string[] args)
{
    TekuciRacun tr1 = new TekuciRacun();
    tr1.ime = "Pera";
    tr1.prezime = "Peric";
    tr1.stanje = 12345.34;
    Stampaj(tr1);
    tr1 = null;
    Stampaj(tr1);
    Console.ReadLine();
}
```

Generisanje izuzetka od strane okruženja



Metode klase

- Metode klase predstavljaju funkcije – članice klase.
- Svaka metoda sadrži :
  - tip podataka koga metoda vraća (ili void ukoliko ne vraća podatke)
  - naziv(ime) metode
  - listu parametara
  - telo metode
- Ukoliko metoda vraća neku vrednost, onda se unutar tela metode mora pozvati naredba **return**.

```
public T nazivMetode(T1 param1, ..., TN paramN)
{
    // telo metode
    return rezultat;
}
```

Primeri metoda

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

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

```
public void Isplata(double iznos)
{
    stanje -= iznos;
}
```

Klasa sa poljima i metodama

```
internal class TekuciRacun
{
    public string ime;
    public string prezime;
    public double stanje;

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

    public void Isplata(double iznos)
    {
        stanje -= iznos;
    }
}
```

Pozivanje metoda klase

```
static void Main(string[] args)
{
    TekuciRacun tr1 = new TekuciRacun();
    tr1.ime = "Pera";
    tr1.prezime = "Peric";
    tr1.stanje = 12345.34;
    Stampaj(tr1);

    tr1.Uplata(45678.12);
    Stampaj(tr1);

    tr1.Isplata(12345.45);
    Stampaj(tr1);
    Console.ReadLine();
}
```

Klasa sa podrazumevanim konstruktorom

```
internal class TekuciRacun
{
    public string ime;
    public string prezime;
    public double stanje;
    public void Uplata(double iznos)
    {
        stanje += iznos;
    }
    public void Isplata(double iznos)
    {
        stanje -= iznos;
    }
}
```

Definisanje parametarskog konstruktora

```
public TekuciRacun(string ime1, string prezime1, double stanje1)
{
    ime = ime1;
    prezime = prezime1;
    stanje = stanje1;
}
```

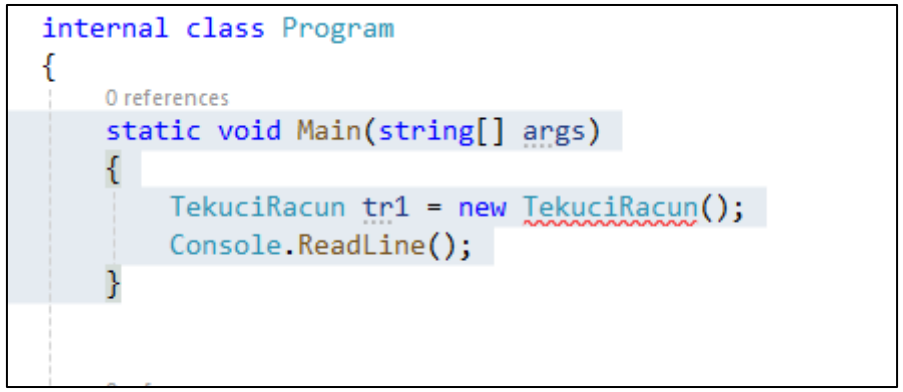
```
public TekuciRacun(string ime, string prezime, double stanje)
{
    this.ime = ime;
    this.prezime = prezime;
    this.stanje = stanje;
}
```

Klasa sa parametarskim konstruktorom

```
internal class TekuciRacun
{
    public string ime;
    public string prezime;
    public double stanje;
    public TekuciRacun(string ime, string prezime, double stanje)
    {
        this.ime = ime;
        this.prezime = prezime;
        this.stanje = stanje;
    }
    public void Uplata(double iznos)
    {
        stanje += iznos;
    }
    public void Isplata(double iznos)
    {
        stanje -= iznos;
    }
}
```

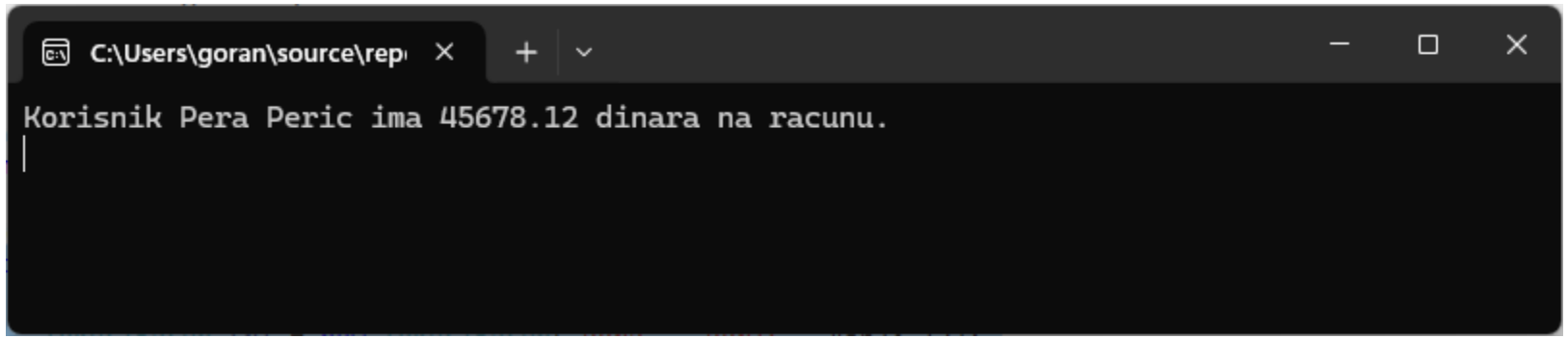
Poziv podrazumevanog konstruktora

```
static void Main(string[] args)
{
    TekuciRacun tr1 = new TekuciRacun();
    Console.ReadLine();
}
```



Poziv parametarskog konstruktora

```
static void Main(string[] args)
{
    TekuciRacun tr1 = new TekuciRacun("Pera", "Peric", 45678.12);
    Stampaj(tr1);
    Console.ReadLine();
}
```



Definisanje konstruktora bez parametara

```
public TekuciRacun()
{
}
```

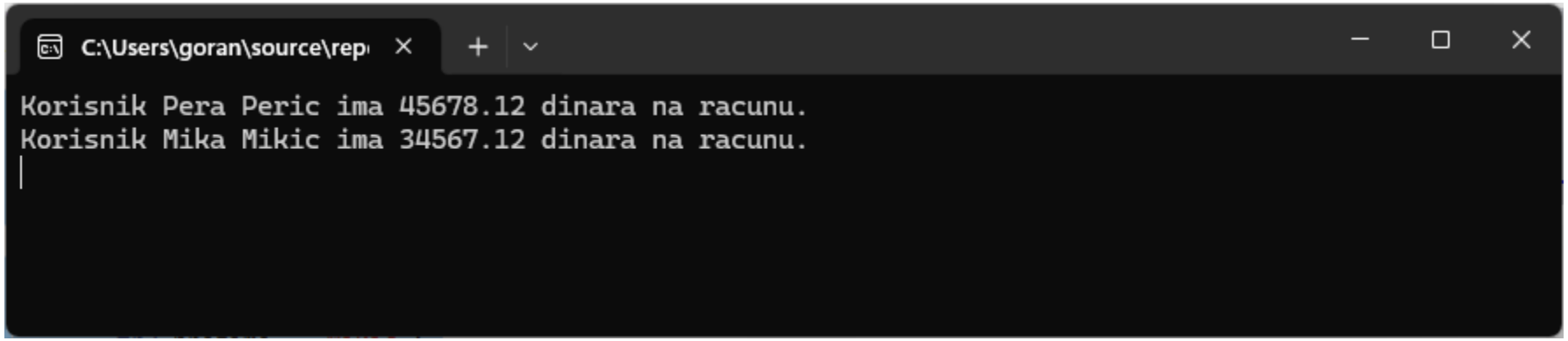
Poziv konstruktora

```
static void Main(string[] args)
{
    TekuciRacun tr1 = new TekuciRacun("Pera", "Peric", 45678.12);
    Stampaj(tr1);

    TekuciRacun tr2 = new TekuciRacun();
    tr2.ime = "Mika";
    tr2.prezime = "Mikic";
    tr2.stanje = 34567.12;

    Stampaj(tr2);
    Console.ReadLine();
}
```

Poziv različitih konstruktora - rezultat



Metoda Stampaj() u klasi TekuciRacun

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

public void Stampaj()
{
    Console.WriteLine($"Korisnik {ime} {prezime} ima {stanje}
    dinara na racunu.");
}
```

Poziv nove metode Stampaj()

```
static void Main(string[] args)
{
    TekuciRacun tr1 = new TekuciRacun("Pera", "Peric", 45678.12);
    tr1.Stampaj();

    TekuciRacun tr2 = new TekuciRacun();
    tr2.ime = "Mika";
    tr2.prezime = "Mikic";
    tr2.stanje = 34567.12;

    tr2.Stampaj();
    Console.ReadLine();
}
```

Pitanje 1

Sposobnost objekta da skriva svoje podatke i implementacione detalje naziva se:

a. abstrakcija
b. enkapsulacija
c. nasleđivanje

Odgovor: b

Pitanje 2

Klasa može sadržati:

a. samo podatke ili polja klase
b. samo metode ili funkcije klase
c. i polja i metode

Odgovor: c

Pitanje 3

Ako metoda klase ne vraća vrednost onda je njen povratni tip:

a. object
b. void
c. int

Odgovor: b

Pitanje 4

Neka je kreirana klasa Student i unutar nje konstruktor koji ima dva ulazna parametra. Povratna vrednost konstruktora je tipa ?

a. int
b. objekat klase Student
c. void
d. konstruktor klase Student ne vraća nikakvu vrednost

Odgovor: d

Pitanje 5

Kada se unutar klase definiše konstruktor tada nastupa sledeća situacija:

a. klasa pored definisanog konstruktora ima I podrazumevani konstruktor
b. vrši se prebrisavanje podrazumevanog konstruktora definisanim konstruktorom
c. generiše se greška jer klasa već ima podrazumevani konstruktor

Odgovor: b

Pitanje 6

Unutar klase moguće je definisati:

a. samo jedan konstruktor
b. najviše dva konstruktora
c. neograničeni broj konstruktora

Odgovor: c