Vežba 4

Nasleđivanje je mehanizam uz pomoć kog klasa naslednica može da poprimi sve atribute roditeljske klase i doda nove, kao i da poprimi sve funkcije roditeljske klase, proširi ih, redefiniše i doda nove.

Na slici 1 se može videti primer 2 slične klase koje se razlikuju samo u jednom atirbutu.

```python
class PrehrambeniProizvod:

    @property
    def sifra(self):
        return self.__sifra

    @sifra.setter
    def sifra(self, sifra):
        self.__sifra = sifra

    @property
    def naziv(self):
        return self.__naziv

    @naziv.setter
    def naziv(self, naziv):
        self.__naziv = naziv

    @property
    def cena(self):
        return self.__cena

    @cena.setter
    def cena(self, cena):
        self.__cena = cena

    @property
    def rok_trajanja(self):
        return self.__rok_trajanja

    @rok_trajanja.setter
    def rok_trajanja(self, rok_trajanja):
        self.__rok_trajanja = rok_trajanja

    def __init__(self, sifra, naziv, cena, rok_trajanja):
        self.__sifra = sifra
        self.__naziv = naziv
        self.__cena = cena
        self.__rok_trajanja = rok_trajanja

    def __str__(self):
        return "\n".join([
            "",
            "{:>12}: {}".format("Šifra", self.__sifra),
            "{:>12}: {}".format("Naziv", self.__naziv),
            "{:>12}: {}".format("Cena", self.__cena),
            "{:>12}: {} dana".format("Rok trajanja", self.__rok_t
        ])
```

```python
class TehnickiProizvod:

    @property
    def sifra(self):
        return self.__sifra

    @sifra.setter
    def sifra(self, sifra):
        self.__sifra = sifra

    @property
    def naziv(self):
        return self.__naziv

    @naziv.setter
    def naziv(self, naziv):
        self.__naziv = naziv

    @property
    def cena(self):
        return self.__cena

    @cena.setter
    def cena(self, cena):
        self.__cena = cena

    @property
    def garancija(self):
        return self.__garancija

    @garancija.setter
    def garancija(self, garancija):
        self.__garancija = garancija

    def __init__(self, sifra, naziv, cena, garancija):
        self.__sifra = sifra
        self.__naziv = naziv
        self.__cena = cena
        self.__garancija = garancija

    def __str__(self):
        return "\n".join([
            "",
            "{:>12}: {}".format("Šifra", self.__sifra),
            "{:>12}: {}".format("Naziv", self.__naziv),
            "{:>12}: {}".format("Cena", self.__cena),
            "{:>12}: {} dana".format("Garancija", self.__gar
        ])
```

Slika 1. Slične klase

Moguće je izbeći ponovljenu implementaciju definisanjem roditeljske klase koja će implementirati zajedničke osobine obeju klasa naslednica. U klasama naslednicama je potrebno implementirati samo razlike (slika 2).

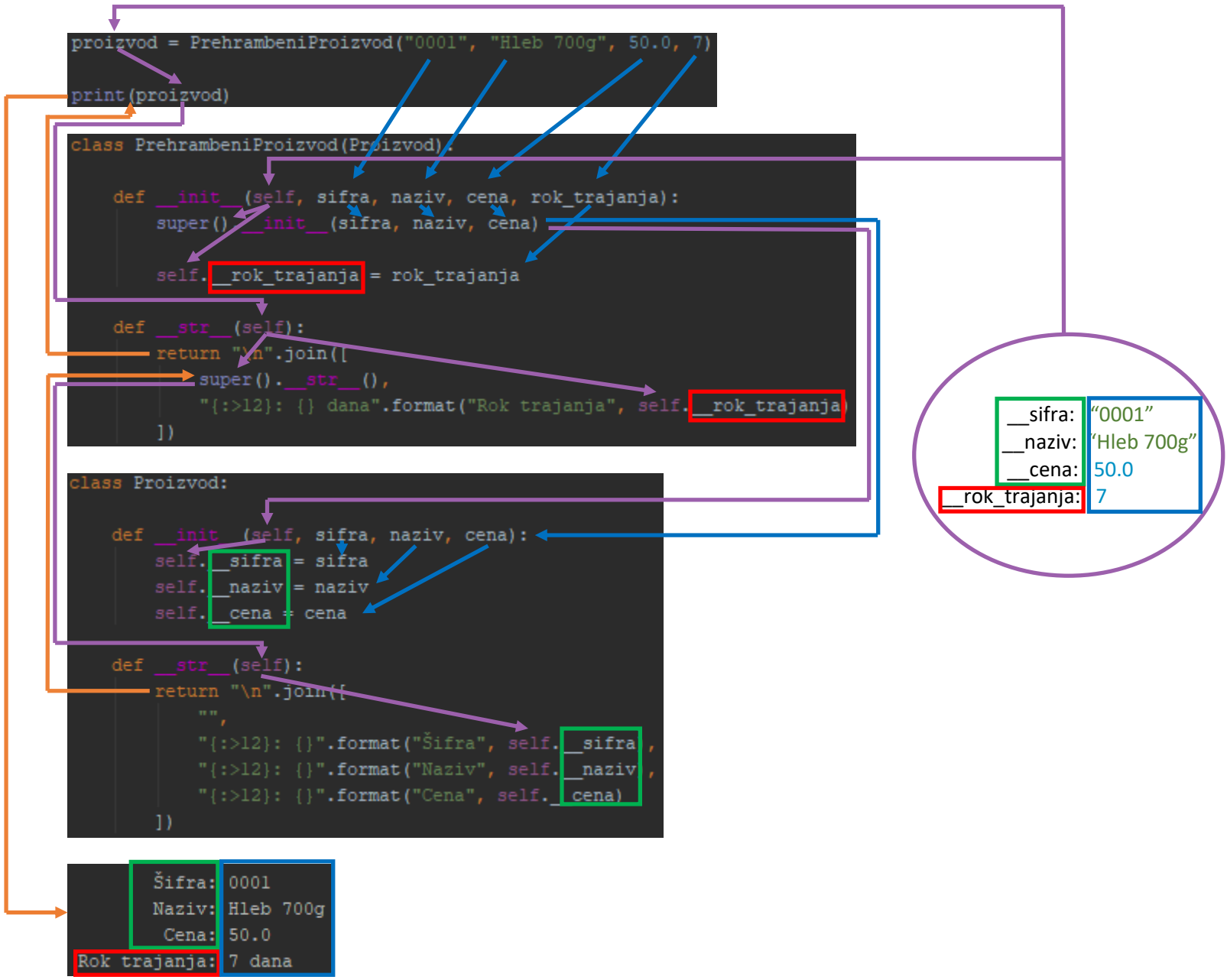
```python
class Proizvod:

    @property
    def sifra(self):
        return self.__sifra

    @sifra.setter
    def sifra(self, sifra):
        self.__sifra = sifra

    @property
    def naziv(self):
        return self.__naziv

    @naziv.setter
    def naziv(self, naziv):
        self.__naziv = naziv

    @property
    def cena(self):
        return self.__cena

    @cena.setter
    def cena(self, cena):
        self.__cena = cena

    def __init__(self, sifra, naziv, cena):
        self.__sifra = sifra
        self.__naziv = naziv
        self.__cena = cena

    def __str__(self):
        return "\n".join([
            "",
            "{:>12}: {}".format("Šifra", self.__sifra),
            "{:>12}: {}".format("Naziv", self.__naziv),
            "{:>12}: {}".format("Cena", self.__cena)
        ])
```

```python
class PrehrambeniProizvod(Proizvod):

    @property
    def rok_trajanja(self):
        return self.__rok_trajanja

    @rok_trajanja.setter
    def rok_trajanja(self, rok_trajanja):
        self.__rok_trajanja = rok_trajanja

    def __init__(self, sifra, naziv, cena, rok_trajanja):
        super().__init__(sifra, naziv, cena)

        self.__rok_trajanja = rok_trajanja

    def __str__(self):
        return "\n".join([
            super().__str__(),
            "{:>12}: {} dana".format("Rok trajanja", self.__rok_trajanj
        ])


class TehnickiProizvod(Proizvod):

    @property
    def garancija(self):
        return self.__garancija

    @garancija.setter
    def garancija(self, garancija):
        self.__garancija = garancija

    def __init__(self, sifra, naziv, cena, garancija):
        super().__init__(sifra, naziv, cena)

        self.__garancija = garancija

    def __str__(self):
        return "\n".join([
            super().__str__(),
            "{:>12}: {} dana".format("Garancija", self.__garancija)
        ])
```

Slika 2. Nasleđivanje

Implementaciju funkcija klasa naslednica je moguće izvesti redefinisanjem, odnosno proširivanjem. Čim se u klasi naslednici navede **funkcija sa istim nazivom** kakav postoji u roditeljskoj klasi, pri pozivu, biće pozvana funkcija klase naslednice. Ako se implementaciji te funkcije u klasi naslednici pozove implementacija istoimene funkcije iz roditeljske klase, posredstvom funkcije *super()*, tada se radi o proširivanju funkcije. U suprotnom se radi o redefinisanju funkcije, pri čemu se potpuno premošćava implementacija u roditeljskoj klasi.

I konstruktor i __str__ funkcija sa slike 2 su dobijene proširivanjem. Na slici 3 se može videti način funkcionisanja **proširivanja**.



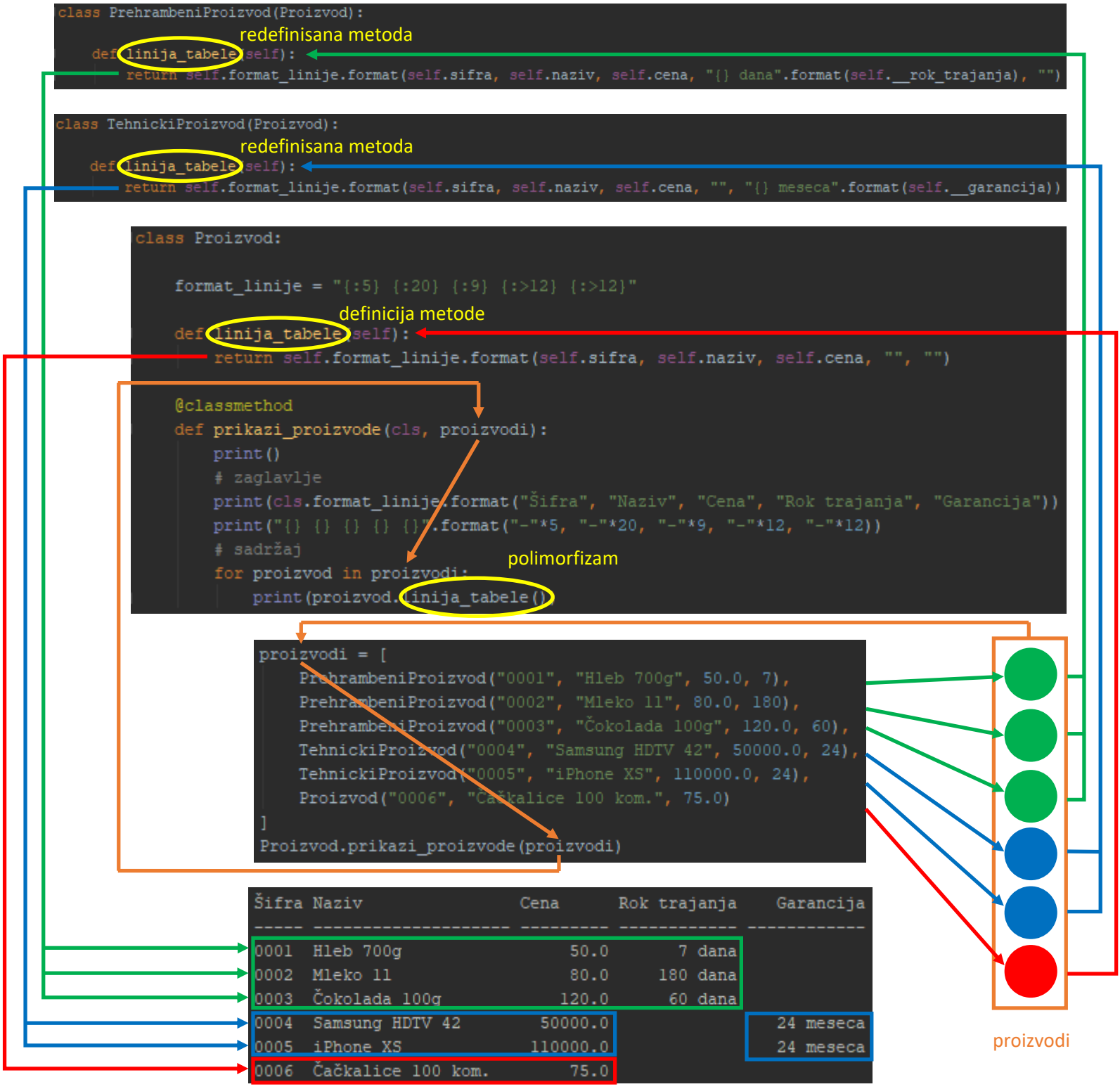
Slika 3. Proširivanje funkcija

Na slici 4 se može videti primer **redefinisanja** metode *linija_tabele*. Primetiti da nigde u klasama naslednicama nema poziva *super()* funkcije.

Polimorfizam podrazumeva sposobnost objekata različitih klasa da ispolje različito ponašanje pozivom metode sa istim nazivom. Samo metode mogu biti polimorfne. Na slici 4 se takođe može videti način funkcionisanja polimorfizma na primeru metode *linija_tabele*.

Na slici 4 se još može videti i način upotrebe **statičkog atributa klase** *format_linije*. Njemu se iz metoda može pristupiti kroz referencu ***self***, a iz funkcija klase mu se može pristupiti kroz referencu ***cls***.

Primetiti na slici 4 i da se **atributima roditeljske klase u funkcijama klasa naslednica mora pristupati preko *property*-a**, za razliku od atributa koji su definisani u tim klasama.



Slika 4. Polimorfizam

Komplementarno polimorfizmu, različito ponašanje programa u zavisnosti od tipova podataka (klasa) sa kojima radi, može se postići **ispitivanjem tipova podataka za vreme izvršavanja** programa posredstvom funkcije *isinstance*. Na slici 5 se može videti primer upotrebe ove funkcije.

```python
class Proizvod:

    @classmethod
    def prikazi_prebrojavanje(cls, proizvodi):
        prehrambeni = 0
        tehnicki = 0
        ostali = 0
        for proizvod in proizvodi:
            if isinstance(proizvod, PrehrambeniProizvod):
                prehrambeni += 1
            elif isinstance(proizvod, TehnickiProizvod):
                tehnicki += 1
            else:
                ostali += 1

        print("\n".join([
            "",
            "{:>11}: {}".format("Prehrambeni", prehrambeni),
            "{:>11}: {}".format("Tehnički", tehnicki),
            "{:>11}: {}".format("Ostali", ostali)
        ]))
```

da li je *proizvod PrehrambeniProizvod*?

da li je *proizvod TehnickiProizvod*?

```python
proizvodi = [
    PrehrambeniProizvod("0001", "Hleb 700g", 50.0, 7),
    PrehrambeniProizvod("0002", "Mleko 1l", 80.0, 180),
    PrehrambeniProizvod("0003", "Čokolada 100g", 120.0, 60),
    TehnickiProizvod("0004", "Samsung HDTV 42", 50000.0, 24),
    TehnickiProizvod("0005", "iPhone XS", 110000.0, 24),
    Proizvod("0006", "Čačkalice 100 kom.", 75.0)
]
Proizvod.prikazi_prebrojavanje(proizvodi)
```

```
Prehrambeni: 3
   Tehnički: 2
     Ostali: 1
```

Slika 5. Funkcija *isinstance*

Zadatak 1

Napisati modul *zaposlen.py*. U modulu je najpre potrebno uvesti klasu *Osoba* iz modula *osoba.py*.
Klasa ***Zaposlen*** nasleđuje klasu *Osoba*. Klasa *Zaposlen* nema dodatne atribute, konstruktor, ni *__str__* metodu, a definiše:

1. metodu *mesecna_zarada*, bez implementacije
2. funkciju klase *prikazi*
3. funkciju klase *ukupna_mesecna_zarada*

Početna implementacija klase *Zaposlen* se može videti na slici 1.

```python
from osoba import Osoba


class Zaposlen(Osoba):

    def mesecna_zarada(self):
        pass

    @classmethod
    def prikazi(cls, zaposleni):
        pass

    @classmethod
    def ukupna_mesecna_zarada(cls, zaposleni):
        pass
```

Slika 1. Klasa zaposleni

Potrebno je definisati klasu ***Lekar*** koja nasleđuje klasu *Zaposlen*, a:

1. definiše atribut *__specijalizacija* (tipa *string*) i odgovarajući *property*
2. proširuje konstruktor, tako da inicijalizuje i novi atribut
3. proširuje *__str__* metodu tako da prikazuje i novi atribut

Potrebno je definisati i klasu ***Tehnicar*** koja nasleđuje klasu *Zaposlen*, a:

1. definiše atribut *__strucna_sprema* (tipa *string*) i odgovarajući *property*
2. proširuje konstruktor, tako da inicijalizuje i novi atribut
3. proširuje *__str__* metodu tako da prikazuje i novi atribut

Test funkcija koja odgovara ovakvom programu je na slici 2.

```python
def test():
    zaposleni = [
        Lekar("1111111111111", "Aaa", "Aaa", 1981, "opsti"),
        Lekar("2222222222222", "Bbb", "Bbb", 1982, "internista"),
        Tehnicar("3333333333333", "Ccc", "Ccc", 1983, "visoka"),
        Tehnicar("4444444444444", "Ddd", "Ddd", 1984, "visa"),
        Tehnicar("5555555555555", "Eee", "Eee", 1985, "srednja")
    ]
    Zaposlen.prikazi(zaposleni)
    Zaposlen.ukupna_mesecna_zarada(zaposleni)
```

Slika 2. Test funkcija modula *zaposlen.py*

Potrebno je implementirati funkciju ***prikazi*** klase *Zaposlen*, tako da od korisnika zatraži izbor na osnovu kog će iz liste zaposlenih ispisati samo lekare, samo tehničare ili sve zaposlene (osloniti se na funkciju *isinstance*). Primer rezultata 3 različita poziva ove funkcije je na slici 3.

```
1. prikaži lekare
2. prikaži tehničare
3. prikaži sve
--------------------
Uneste izbor: 1

        JMBG: 1111111111111
         Ime: Aaa
     Prezime: Aaa
God. rođenja: 1981
     Starost: 38
       Spec.: opsti

        JMBG: 2222222222222
         Ime: Bbb
     Prezime: Bbb
God. rođenja: 1982
     Starost: 37
       Spec.: internista
```

```
1. prikaži lekare
2. prikaži tehničare
3. prikaži sve
--------------------
Uneste izbor: 2

        JMBG: 3333333333333
         Ime: Ccc
     Prezime: Ccc
God. rođenja: 1983
     Starost: 36
   Str. spr.: visoka

        JMBG: 4444444444444
         Ime: Ddd
     Prezime: Ddd
God. rođenja: 1984
     Starost: 35
   Str. spr.: visa

        JMBG: 5555555555555
         Ime: Eee
     Prezime: Eee
God. rođenja: 1985
     Starost: 34
   Str. spr.: srednja
```

```
1. prikaži lekare
2. prikaži tehničare
3. prikaži sve
--------------------
Uneste izbor: 3

        JMBG: 1111111111111
         Ime: Aaa
     Prezime: Aaa
God. rođenja: 1981
     Starost: 38
       Spec.: opsti

        JMBG: 2222222222222
         Ime: Bbb
     Prezime: Bbb
God. rođenja: 1982
     Starost: 37
       Spec.: internista

        JMBG: 3333333333333
         Ime: Ccc
     Prezime: Ccc
God. rođenja: 1983
     Starost: 36
   Str. spr.: visoka

        JMBG: 4444444444444
         Ime: Ddd
     Prezime: Ddd
God. rođenja: 1984
     Starost: 35
   Str. spr.: visa

        JMBG: 5555555555555
         Ime: Eee
     Prezime: Eee
God. rođenja: 1985
     Starost: 34
   Str. spr.: srednja
```

Slika 3. Poziv funkcije *prikazi* klase *Zaposlen*

Potrebno je redefinisati metodu ***mesecna_zarada*** u klasama:

1. *Lekar*, tako da vraća vrednost 65000.0 ako je specijalizacija lekara “opsti”, odnosno 75000.0 u suprotnom
2. *Tehnicar*, tako da vraća vrednost 60000.0 ako je stručna sprema tehničara “visoka”, 50000.0 ako je stručna sprema tehničara “visa”, odnosno 35000.0, ako je stručna sprema tehničara “srednja”

Potrebno je implementirati funkciju ***ukupna_mesecna_zarada*** klase *Zaposlen* tako da upotrebom metode *mesecna_zarada* svakog zaposlenog izračuna ukupnu mesečnu zaradu svih zaposlenih. Primer rezultata poziva ove funkcije je na slici 4.

```
Ukupna mesečna zarada: 285000.0
```

Slika 4. Poziv funkcije *ukupna_mesecna_zarada* klase *Zaposlen*

Katedra za informatiku
Novi Sad, 2019.