

Zadaća br. 3

Izveštaj o *white box* testiranju

Uputstvo za izradu zadaće

Izrada zadaće vrši se u formi izvještaja koja je data u nastavku. Potrebno je popuniti sva polja data u izvještaju, odgovoriti na pitanja i dodati tražene slike. Nije dozvoljeno brisati postojeća, niti dodavati nova polja.

Zadaća se radi u timovima od po tri studenta. Svi studenti iz istog tima popunjavaju isti izvještaj u jednom dokumentu, s tim da popunjavaju različite dijelove dokumenta ovisno o postavkama zadataka. Dovoljno je da jedan član tima pošalje izvještaj preko Zamgera.

Informacije o timu

Popuniti informacije o studentima koji vrše izradu zadaće.

Dodijeljeno programsko rješenje: eParking

Ime i prezime: Amina Šiljak
Broj indexa: 18496

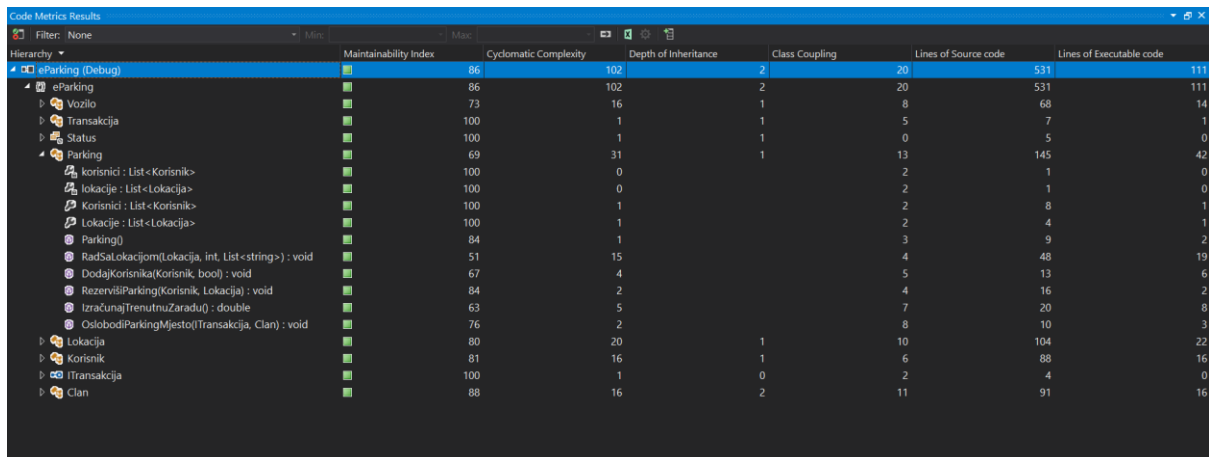
Ime i prezime: Elma Šeremet
Broj indexa: 18318

Ime i prezime: Berina Suljić
Broj indexa: 18385

Zadatak 1. (Metrike i refaktoring)

Potrebno je, koristeći alat *Visual Studio Code Metrics*, izračunati vrijednost najvažnijih metrika i pronaći metodu koja ima najveću vrijednost McCabe metrike. Ukoliko ima više takvih metoda, potrebno je odabrati bilo koju od njih.

Prikaz rezultata analize metrika koda putem alata *Visual Studio Code Metrics*:



	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
eParking (Debug)	86	102	2	20	531	111
eParking	86	102	2	20	531	111
Vozilo	73	16	1	8	68	14
Transakcija	100	1	1	5	7	1
Status	100	1	1	0	5	0
Parking	69	31	1	13	145	42
Korisnici : List<Korisnik>	100	0	2	2	1	0
Lokacije : List<Lokacija>	100	0	2	1	0	0
Korisnici : List<Korisnik>	100	1	2	8	1	1
Lokacije : List<Lokacija>	100	1	2	4	1	1
Parking()	84	1	3	9	2	2
RadSaLokacijom(Lokacija, int, List<string>) : void	51	15	4	48	19	19
DodajKorisnika(Korisnik, bool) : void	67	4	5	13	6	6
RezervisiParking(Korisnik, Lokacija) : void	84	2	4	16	2	2
IzracunajTrenutnuZaradu() : double	63	5	7	20	8	8
OslobodiParkingMjesto() (Transakcija, Clan) : void	76	2	8	10	3	3
Lokacija	80	20	1	10	104	22
Korisnik	81	16	1	6	88	16
ITransakcija	100	1	0	2	4	0
Clan	88	16	2	11	91	16

Metoda koja ima najveću ciklomatsku kompleksnost: `RadSaLokacijom(Lokacija, int, List<string>) : void`

Prikaz programskog koda metode sa najvećom ciklomatskom kompleksnošću:

Verifikacija i Validacija Softvera

```
0 references
public void RadSaLokacijom(Lokacija l, int opcija, List<string> podaci = null)
{
    if (opcija == 0)
    {
        Lokacija location = null;
        foreach (Lokacija lokacija in Lokacije)
        {
            if (lokacija.Naziv == l.Naziv)
                location = lokacija;
        }
        if (location != null)
            throw new InvalidOperationException("Lokacija već postoji!");

        Lokacije.Add(l);
    }
    else if (opcija == 1)
    {
        Lokacija location = null;
        foreach (Lokacija lokacija in Lokacije)
        {
            if (lokacija.Naziv == l.Naziv)
                location = lokacija;
        }
        if (location == null)
            throw new InvalidOperationException("Lokacija ne postoji!");

        Lokacije.Remove(location);
    }
    else if (opcija == 2)
    {
        Lokacija location = null;
        foreach (Lokacija lokacija in Lokacije)
        {
            if (lokacija.Naziv == l.Naziv)
                location = lokacija;
        }
        if (location == null)
            throw new InvalidOperationException("Lokacija ne postoji!");

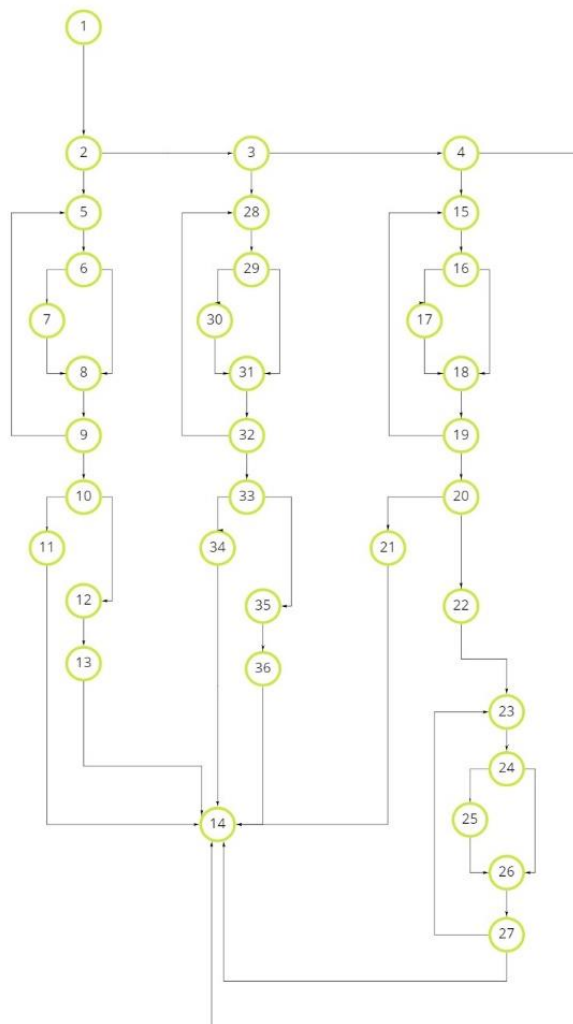
        foreach (string ulica in podaci)
            if (!location.Ulice.Contains(ulica))
                location.Ulice.Add(ulica);
    }
}
```

Šta je razlog za veliku ciklomatsku kompleksnost?

Veliki broj kontrolnih struktura.

Kako bi se izvršila provjera dobivene vrijednosti McCabe metrike, potrebno je konstruisati kontrolni graf za metodu i manuelno izračunati vrijednost ciklomatske kompleksnosti.

Izgled kontrolnog grafa metode:



Prikaz programskog koda metode koja je odabrana za izračun Halstead metrika:

Verifikacija i Validacija Softvera

```
0 references
public void RadSaLokacijom(Lokacija l, int opcija, List<string> podaci = null)
{
    if (opcija == 0)
    {
        Lokacija location = null;
        foreach (Lokacija lokacija in Lokacije)
        {
            if (lokacija.Naziv == l.Naziv)
                location = lokacija;
        }
        if (location != null)
            throw new InvalidOperationException("Lokacija već postoji!");

        Lokacije.Add(l);
    }
    else if (opcija == 1)
    {
        Lokacija location = null;
        foreach (Lokacija lokacija in Lokacije)
        {
            if (lokacija.Naziv == l.Naziv)
                location = lokacija;
        }
        if (location == null)
            throw new InvalidOperationException("Lokacija ne postoji!");

        Lokacije.Remove(location);
    }
    else if (opcija == 2)
    {
        Lokacija location = null;
        foreach (Lokacija lokacija in Lokacije)
        {
            if (lokacija.Naziv == l.Naziv)
                location = lokacija;
        }
        if (location == null)
            throw new InvalidOperationException("Lokacija ne postoji!");

        foreach (string ulica in podaci)
            if (!location.Ulice.Contains(ulica))
                location.Ulice.Add(ulica);
    }
}
```

Na osnovu čega je izvršen odabir metode? Šta se može postići izračunavanjem Halstead metrika za odabranu metodu?

Odabir metode je izvršen na osnovu broja operatora i operandada. Smatramo da je ovo metoda na kojoj najbolje možemo demonstrirati šta znači to što je Halstead metrika velika, jer ona ima najveći broj operatora i operandada. Izračunavanjem Halstead metrika za odabranu

U tabele ispod potrebno je unijeti vrijednosti operatora i operandada koje metoda sadrži. Potrebno je izračunati broj pojedinačnih i ukupan broj operatora i operandada n i N . Ove vrijednosti neophodne su za izračunavanje Halstead metrika. Zatim je potrebno izračunati Halstead metrike koje se nalaze u trećoj tabeli.

Operandi:

Ime operatora	Broj ponavljanja
==	8
=	5
()	7
!	1
throw	3
new	3

Verifikacija i Validacija Softvera

if	10
else	2
public	1
foreach	4
.	11
!=	1
;	11
{ }	7
Broj različitih operatora (n_1)	Ukupan broj operatora (N_1)
$n_1 = 14$	$N_1 = 74$

Operatori:

Ime operanda	Broj ponavljanja
void	1
l	5
opcija	4
podaci	2
null	7
int	1
string	2
0	1
1	1
2	1
"Lokacija već postoji!"	1
"Lokacija ne postoji!"	2
location	12
ulica	3
lokacija	9
Lokacije	5
List<>	1
Lokacija	7
Broj različitih operanada (n_2)	Ukupan broj operanada (N_2)
$n_2 = 18$	$N_2 = 65$

Halstead metrike	
Dužina programa	$N = N_1 + N_2 = 74 + 65 = 139$
Veličina vokabulara	$n = n_1 + n_2 = 14 + 18 = 32$
Volumen programa	$V = N * \log_2(n) = 139 * \log_2(32) = 695$
Nivo poteškoće	$D = \left(\frac{n_1}{2}\right) * \left(\frac{N_2}{n_2}\right) = \left(\frac{14}{2}\right) * \left(\frac{65}{18}\right) = 25.28$
Nivo programa	$L = \frac{1}{D} = \frac{1}{25.28} = 0.04$
Napor implementacije	$E = V * D = 695 * 25.28 = 17569.6$

Verifikacija i Validacija Softvera

Vrijeme implementacije	$T = \frac{E}{18} = \frac{17569.6}{18} = 976.08$
Broj isporučenih bugova	$B = \frac{E^2}{3000} = 0.23$

Izvršiti analizu dobivenih vrijednosti. Da li su vrijednosti nekih od metrika izvan referentnog opsega? Šta se može zaključiti na osnovu dobivenih vrijednosti?

Obzirom da je volumen programa 695, to je unutar referentnog opsega, pa metoda ne radi mnogo stvari. Obzirom da je nivo programa 0.04, to znači da je metoda više podložena greškama. Broj isporučenih bugova nije veliki, ali programski kod obično sadrži više grešaka nego što B sugerira. Vrijeme implementacije također nije veliko (traje oko pola sata). Što se tiče napora implementacije, ono je možda malo više nego što bi trebalo biti. Nivo poteškoće je oko 25, to samo znači da se neki operandi koriste više puta u metodi pa to može uzrokovati više grešaka.

Prikaz programskog koda metode koja je odabrana za izračun metrike informacijskog toka:

Verifikacija i Validacija Softvera

```
40  #region Metode
41  2 references | 1/1 passing
42  public void RadSaLokacijom(Lokacija l, int opcija, List<string> podaci = null)
43  {
44      if (opcija == 0)
45      {
46          Lokacija location = null;
47          foreach (Lokacija lokacija in Lokacije)
48          {
49              if (lokacija.Naziv == l.Naziv)
50                  location = lokacija;
51          }
52          if (location != null)
53              throw new InvalidOperationException("Lokacija već postoji!");
54          Lokacije.Add(l);
55      }
56      else if (opcija == 1)
57      {
58          Lokacija location = null;
59          foreach (Lokacija lokacija in Lokacije)
60          {
61              if (lokacija.Naziv == l.Naziv)
62                  location = lokacija;
63          }
64          if (location == null)
65              throw new InvalidOperationException("Lokacija ne postoji!");
66          Lokacije.Remove(location);
67      }
68      else if (opcija == 2)
69      {
70          Lokacija location = null;
71          foreach (Lokacija lokacija in Lokacije)
72          {
73              if (lokacija.Naziv == l.Naziv)
74                  location = lokacija;
75          }
76          if (location == null)
77              throw new InvalidOperationException("Lokacija ne postoji!");
78          foreach (string ulica in podaci)
79              if (!location.Ulice.Contains(ulica))
80                  location.Ulice.Add(ulica);
81      }
82  }
83  }
84  }
85  }
```

Na osnovu čega je izvršen odabir metode? Šta se može postići izračunavanjem metrike informacijskog toka za odabranu metodu?

Odabir metode smo izvršili na osnovu broja lokalnih tokova u modul, lokalnih tokova iz modula i broja linija koda metode. O

U tabelu ispod potrebno je unijeti vrijednosti koje su neophodne za izračunavanje metrike informacijskog toka i izračunati vrijednost te metrike.

Dužina programa	<i>length</i> = 26
-----------------	--------------------

Lokalni tok u modul	$t_1 = 10$
Strukture podataka koje se koriste	$s_1 = 2$
Lokalni tok iz modula	$t_2 = 6$
Strukture podataka koje se mijenjaju	$s_2 = 2$
Fanin	$F_1 = 12$
Fanout	$F_2 = 8$
Metrika informacijskog toka (IFC)	$IFC = 9216$
Težinska metrika informacijskog toka (WIFC)	$WIFC = 239616$

Izvršiti analizu dobivenih vrijednosti. Da li su vrijednosti nekih od metrika izvan referentnog opsega? Šta se može zaključiti na osnovu dobivenih vrijednosti?

Mjere informacijskog toka kompleksnosti su relativni indikatori. Visoke vrijednosti se pretežno vežu sa problemom pouzdanosti sistema i njegovim težim održavanjem. Visoka informacijska kompleksnost modula može indicirati: Kompleksnost informacijskog toka IFC je 9216, dok je težinska metrika informacijskog toka WIFC 239616. Na osnovu dobijenih vrijednosti možemo zaključiti da je metoda kandidat za testiranje kao i da ne postoje neke referentne vrijednosti na osnovu kojih bismo mogli donijeti neki zaključak o pouzdanosti sistema, obzirom da su mjere informacijskog toka kompleksnosti relativne vrijednosti.

*Potrebno je odabrati metode koje su najpovoljnije za vršenje refaktoringa i izvršiti refaktoring, a zatim objasniti kakve su promjene nastale nakon refaktoringa. **Potrebno je da svaki član tima izvrši barem po jedan refaktoring.***

Refaktoring – član 1 (Ime i prezime: Berina Suljić)

Prikaz programskog koda odabrane metode prije vršenja refaktoringa:

Verifikacija i Validacija Softvera

```
0 references
public void RadSaLokacijom(Lokacija l, int opcija, List<string> podaci = null)
{
    if (opcija == 0)
    {
        Lokacija location = null;
        foreach (Lokacija lokacija in Lokacije)
        {
            if (lokacija.Naziv == l.Naziv)
                location = lokacija;
        }
        if (location != null)
            throw new InvalidOperationException("Lokacija već postoji!");

        Lokacije.Add(l);
    }
    else if (opcija == 1)
    {
        Lokacija location = null;
        foreach (Lokacija lokacija in Lokacije)
        {
            if (lokacija.Naziv == l.Naziv)
                location = lokacija;
        }
        if (location == null)
            throw new InvalidOperationException("Lokacija ne postoji!");

        Lokacije.Remove(location);
    }
    else if (opcija == 2)
    {
        Lokacija location = null;
        foreach (Lokacija lokacija in Lokacije)
        {
            if (lokacija.Naziv == l.Naziv)
                location = lokacija;
        }
        if (location == null)
            throw new InvalidOperationException("Lokacija ne postoji!");

        foreach (string ulica in podaci)
            if (!location.Ulice.Contains(ulica))
                location.Ulice.Add(ulica);
    }
}
```

Koje refaktoringe je moguće primijeniti i na koji način (opisati šta je neophodno uraditi, kojoj grupi refaktoringa navedeni postupci pripadaju i šta se time postiže)? Odabrati najpovoljniji refaktoring i obrazložiti zbog čega je najpovoljniji.

Razlog za refaktoring ove metode je dupliciranje koda. Konkretno, tri puta se inicijalizuje location, tri puta je napisana ista foreach petlja... Moguće je primijeniti refaktoring da foreach petlju i inicijalizovanje radimo jednom, dok ćemo uslove stavljati nakon foreach petlje, i to pripada refaktoringu na nivou iskaza. Moguće je također odvojiti kod petlje u posebnu metodu i potom tu metodu pozivati kada je potrebno, i to pripada refaktoringu na nivou rutine. Također, jedan od mogućih refaktoringa jeste i zamijeniti izraz sa rutinom što smanjuje dupliciranje koda, a to pripada refaktoringu na nivou podataka. Najpovoljniji refaktoring u ovom slučaju je konsolidovati fragmente koji se dupliciraju unutar različitih dijelova uslovnih iskaza, jer ćemo time dobiti znatno manji broj operanada i operatora, te znatno manju dužinu koda.

Prikaz programskog koda metode nakon vršenja najpogodnijeg refaktoringa:

```
public void RadSaLokacijom(Lokacija l, int opcija, List<string> podaci = null)
{
    Lokacija location = null;
    foreach (Lokacija lokacija in Lokacije)
    {
        if (lokacija.Naziv == l.Naziv)
            location = lokacija;
    }

    if(opcija == 0)
    {
        if (location != null)
            throw new InvalidOperationException("Lokacija već postoji!");
        Lokacije.Add(l);
    }
    else if(opcija == 1)
    {
        if (location == null)
            throw new InvalidOperationException("Lokacija ne postoji!");
        Lokacije.Remove(location);
    }
    else if(opcija == 2)
    {
        if (location == null)
            throw new InvalidOperationException("Lokacija ne postoji!");

        foreach (string ulica in podaci)
            if (!location.Ulice.Contains(ulica))
                location.Ulice.Add(ulica);
    }
}
```

Da li je došlo do poboljšanja ciklomatske kompleksnosti? Kakav to značaj ima za korištenje aplikacije, a kakav za *white box* testiranje?

Došlo je do poboljšanja ciklomatske kompleksnosti, smanjila se sa 15 na 11. Broj neovisnih puteva koji su potrebni da bi se postigao puni obuhvat linija programa se smanjio, i to ima značaj za korištenje aplikacije takav da će održavanje aplikacije biti lakše. Također, smanjenje ciklomatske kompleksnosti znači da će biti potreban manji napor za *white box* testiranje.

Vrijednost indeksa održavanja metode prije vršenja refaktoringa: 51

Vrijednost indeksa održavanja metode nakon vršenja refaktoringa: 56

Da li je indeks održavanja promijenio kategoriju (crvena, žuta, zelena) nakon vršenja refaktoringa? Da li su ciklomatska kompleksnost i indeks održavanja u korelaciji i zašto?

Indeks održavanja je pripadao kategoriji zelena i prije vršenja refaktoringa, a pripada i nakon vršenja refaktoringa. Ciklomatska kompleksnost i indeks održavanja jesu u korelaciji, jer

ciklomatska kompleksnost se smanjila, dok se indeks održavanja povećao (smanjivanjem ciklomatske kompleksnosti, olakšava se održavanje programa).

Refactoring – član 2 (Ime i prezime: Amina Šiljak)

Prikaz programskog koda odabrane metode prije vršenja refaktoringa:

```
0 references
public void DodajKorisnika(Korisnik k, bool clan)
{
    Korisnik korisnik = Korisnici.Find(kor => kor.Username == k.Username);
    if (korisnik != null && !clan)
        throw new ArgumentException("Korisnik već postoji!");
    else if (korisnik != null)
    {
        Korisnici.Remove(korisnik);
        Korisnici.Add(k);
    }
    else
        Korisnici.Add(k);
}
```

Koje refaktoringe je moguće primijeniti i na koji način (opisati šta je neophodno uraditi, kojoj grupi refaktoringa navedeni postupci pripadaju i šta se time postiže)? Odabrati najpovoljniji refactoring i obrazložiti zbog čega je najpovoljniji.

Moguće je izvršiti refactoring na nivou izraza tako da izbjegnemo dupliciranje izraza za dodavanje novog korisnika, kao i jedno grananje, čime ćemo smanjiti kompleksnost koda. Ukoliko korisnik već postoji, ako je clan true baca se izuzetak, a ukoliko nije korisnik se briše iz liste. Korisnik se dodaje u svakom slučaju ako nije bačen izuzetak pa nema potrebe da on bude u bilo kakvom grananju.

Prikaz programskog koda metode nakon vršenja najpogodnijeg refaktoringa:

```
0 references
public void DodajKorisnikaRefactoring(Korisnik k, bool clan)
{
    Korisnik korisnik = Korisnici.Find(kor => kor.Username == k.Username);
    if (korisnik != null)
    {
        if(!clan) throw new ArgumentException("Korisnik već postoji!");
        Korisnici.Remove(korisnik);
    }
    Korisnici.Add(k);
}
```

Da li je došlo do poboljšanja ciklomatske kompleksnosti? Kakav to značaj ima za korištenje aplikacije, a kakav za *white box* testiranje?

Ciklomatska kompleksnost se smanjila za 1. Ovim će se olakšati održavanje aplikacije, te će se naravno i smanjiti napor za white box testiranje pošto je ciklomatska kompleksnost mjera za to.

Vrijednost indeksa održavanja metode prije vršenja refaktoringa: 67

Vrijednost indeksa održavanja metode nakon vršenja refaktoringa: 68

Da li je indeks održavanja promijenio kategoriju (crvena, žuta, zelena) nakon vršenja refaktoringa? Da li su ciklomatska kompleksnost i indeks održavanja u korelaciji i zašto?

Indeks održavanja je i prije i poslije refaktorina u zelenoj kategoriji. Ciklomatska kompleksnost i indeks održavanja jesu u korelaciji - što je manja ciklomatska kompleksnost indeks održavanja je veći, jer se ciklomatska kompleksnost uzima u obzir pri računanju indeksa održavanja.

Refaktoring – član 3 (Ime i prezime: Elma Šeremet)

Prikaz programskog koda odabrane metode prije vršenja refaktoringa:

```
8 references
public double IzračunajTrenutnuZaradu()
{
    double zarada = 0.0;

    foreach (Korisnik k in korisnici)
    {
        if (k is Clan)
        {
            var mjesto = ((Clan)k).RezervisanoParkingMjesto;
            if (mjesto != null)
            {
                if (mjesto.Item2.Ulice.Count == 0)
                    throw new ArgumentNullException("Lokacija nema ulica!");

                zarada += mjesto.Item2.Cijena / mjesto.Item2.Ulice.Count;
            }
        }
    }

    return zarada;
}
```

Koje refaktoringe je moguće primijeniti i na koji način (opisati šta je neophodno uraditi, kojoj grupi refaktoringa navedeni postupci pripadaju i šta se time postiže)? Odabrati najpovoljniji refaktoring i obrazložiti zbog čega je najpovoljniji.

Budući da su ostale metode uglavnom banalne, odabrala sam metodu IzračunajTrenutnuZaradu() zato što mi je zapalo za oko da bih ja uradila nešto na drugačiji i jednostavniji način. Naime, moguće je uraditi refaktoring na nivou rutine, te dio foreach petlje koja sadrži tri if ugniježđena uslova zamijeniti postojećim algoritmom za pronalazak instanci koji implementira metoda klase List FindAll. Na ovaj način grananja se zamjenjuju naredbom

lista = korisnici. korisnici.FindAll(k => k is Clan && ((Clan)k).RezervisanoParkingMjesto != null). Ovim smo svakako izbjegli prva dva if grananja budući da ćemo se sada kretati samo kroz dozvoljene korisnike što je velika prednost u odnosu na početno napisanu metodu. Međutim, budući da metoda sadrži još posla koji treba obaviti potrebna je foreach petlja sa samo jednim if uslovom, koja obavlja sve što je potrebno, ali na znatno jednostavniji način.

Prikaz programskog koda metode nakon vršenja najpogodnijeg refaktoringa:

```
0 references
public double IzračunajTrenutnuZaraduRefactor()
{
    double zarada = 0.0;
    List<Korisnik> lista = korisnici.FindAll(k => k is Clan && ((Clan)k).RezervisanoParkingMjesto != null);

    foreach (Korisnik k in lista)
    {
        var mjesto = ((Clan)k).RezervisanoParkingMjesto;
        if (mjesto.Item2.Ulice.Count == 0)
        {
            throw new ArgumentNullException("Lokacija nema ulica!");
        }
        zarada += mjesto.Item2.Cijena / mjesto.Item2.Ulice.Count;
    }
    return zarada;
}
```

Da li je došlo do poboljšanja ciklomatske kompleksnosti? Kakav to značaj ima za korištenje aplikacije, a kakav za *white box* testiranje?

Do poboljšanja ciklomatske kompleksnosti je došlo, budući da se ona smanjila za 1 što ima značaj za korištenje aplikacije budući da će olakšati održavanje aplikacije, te će se naravno i smanjiti napor za *white box* testiranje pošto je ciklomatska kompleksnost mjera za to.

Vrijednost indeksa održavanja metode prije vršenja refaktoringa: 63

Vrijednost indeksa održavanja metode nakon vršenja refaktoringa: 62

Da li je indeks održavanja promijenio kategoriju (crvena, žuta, zelena) nakon vršenja refaktoringa? Da li su ciklomatska kompleksnost i indeks održavanja u korelaciji i zašto?

Indeks održavanja je i prije i poslije refaktorina u zelenoj kategoriji. Ciklomatska kompleksnost i indeks održavanja jesu u korelaciji - što je manja ciklomatska kompleksnost indeks održavanja je veći, jer se ciklomatska kompleksnost uzima u obzir pri računanju indeksa održavanja.

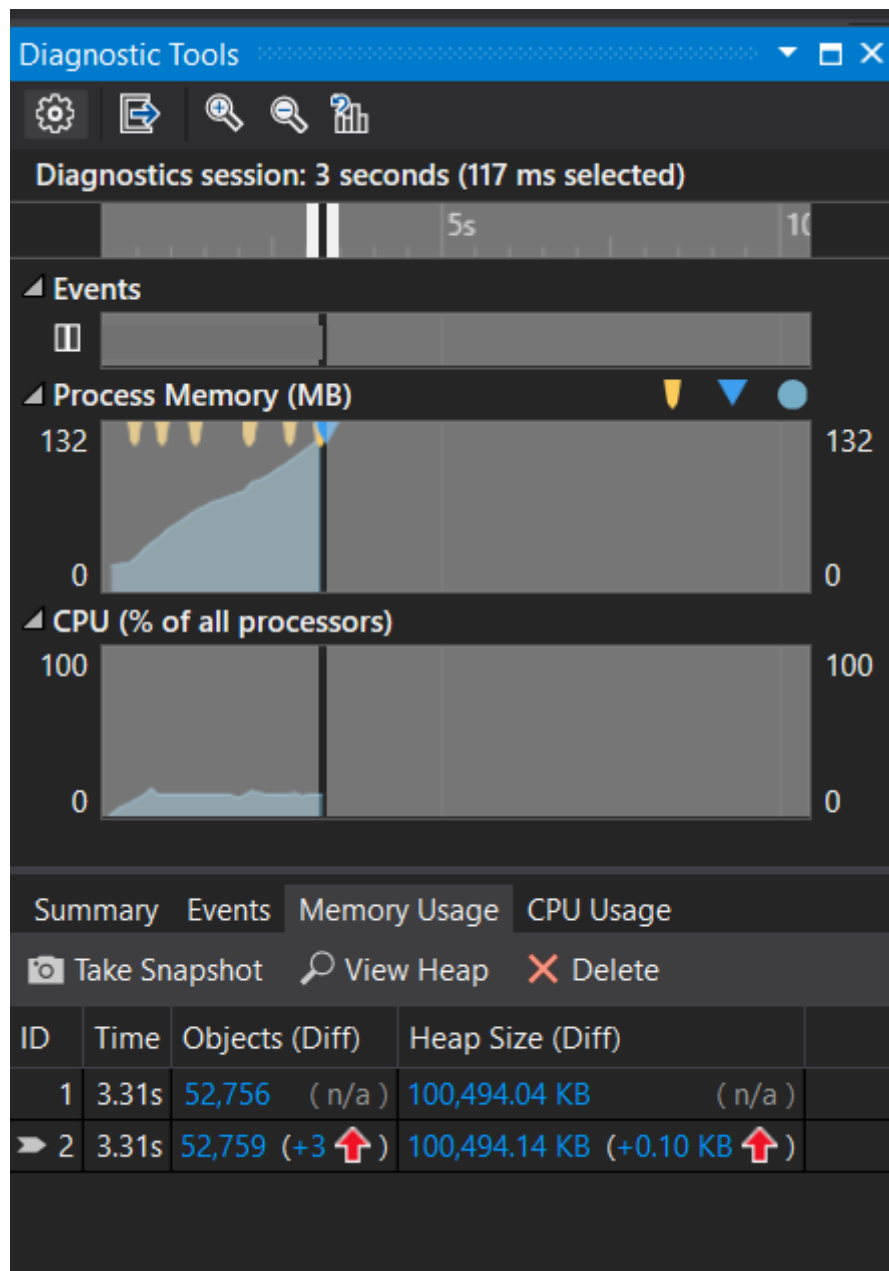
Zadatak 2. (Code tuning)

Za svaku nastavnu grupu definisana je različita metoda koja je pogodna za vršenje code tuninga. U nastavku je potrebno primijeniti tehnike code tuninga kako bi se smanjilo zauzeće memorijskih i procesorskih resursa pri pokretanju navedene metode.

Metoda koja je dodijeljena za *code tuning*: Parking.RadSaLokacijom

*Potrebno je definisati unit test projekat i dodati jedan unit test u kojem se poziva tražena metoda (sam rezultat testa nije važan – dovoljno je da se test može pokrenuti u svrhu pokretanja tražene metode). **Kolekcije elemenata koje se koriste u dodijeljenoj metodi trebaju imati 1,000,000 elemenata.** Pri pokretanju je potrebno izvršiti debugging kako bi se detektovale vrijednosti zauzeća memorije i procesorskih jedinica nakon završetka rada metode. Kako bi se dobili što relevantniji rezultati, **potrebno je izvršiti usrednjavanje dobivenih vrijednosti iz 10 izvršavanja.***

Prikaz *Visual Studio Diagnostic Tool* alata pri dostizanju *breakpointa* nakon završetka izvršavanja metode (dovoljan je prikaz jednog od 10 izvršavanja):



Srednja vrijednost zauzeća memorije: 116.97MB

Srednja vrijednost korištenja procesorskih jedinica: 13%

Srednje vrijeme izvršavanja metode: 3.17s

Da li su navedene vrijednosti prihvatljive i zašto?

Navedene vrijednosti su prihvatljive, jer metoda se izvršila relativno brzo, te nije zauzela previše memorije, koristeći samo 13% procesorskih jedinica.

*Potrebno je izvršiti code tuning nad datom metodom kako bi se poboljšale dobivene vrijednosti zauzeća resursa. **Potrebno je da svaki član tima uradi po jedan code tuning nad metodom.** Code tuning se vrši iterativno, tako da se na kraju dobivaju četiri metode – originalna metoda,*

Verifikacija i Validacija Softvera

metoda nakon vršenja prvog code tuninga, metoda nakon vršenja prvog i drugog tuninga, kao i metoda nakon vršenja svih code tuninga.

Prikaz programskog koda metode prije vršenja *code tuninga*:

```
40  #region Metode
41  2 references | 1/1 passing
42  public void RadSaLokacijom(Lokacija l, int opcija, List<string> podaci = null)
43  {
44      if (opcija == 0)
45      {
46          Lokacija location = null;
47          foreach (Lokacija lokacija in Lokacije)
48          {
49              if (lokacija.Naziv == l.Naziv)
50                  location = lokacija;
51          }
52          if (location != null)
53              throw new InvalidOperationException("Lokacija već postoji!");
54          Lokacije.Add(l);
55      }
56      else if (opcija == 1)
57      {
58          Lokacija location = null;
59          foreach (Lokacija lokacija in Lokacije)
60          {
61              if (lokacija.Naziv == l.Naziv)
62                  location = lokacija;
63          }
64          if (location == null)
65              throw new InvalidOperationException("Lokacija ne postoji!");
66          Lokacije.Remove(location);
67      }
68      else if (opcija == 2)
69      {
70          Lokacija location = null;
71          foreach (Lokacija lokacija in Lokacije)
72          {
73              if (lokacija.Naziv == l.Naziv)
74                  location = lokacija;
75          }
76          if (location == null)
77              throw new InvalidOperationException("Lokacija ne postoji!");
78          foreach (string ulica in podaci)
79              if (!location.Ulice.Contains(ulica))
80                  location.Ulice.Add(ulica);
81      }
82  }
83
84
85
```

Prikaz programskog koda metode nakon vršenja prvog *code tuninga*:

Verifikacija i Validacija Softvera

```
41 public void RadSaLokacijomTuning1(Lokacija l, int opcija, List<string> podaci = null)
42 {
43     if (opcija == 0)
44     {
45         Lokacija location = null;
46         foreach (Lokacija lokacija in Lokacije)
47         {
48             if (lokacija.Naziv == l.Naziv)
49             {
50                 location = lokacija;
51                 break;
52             }
53         }
54         if (location != null)
55             throw new InvalidOperationException("Lokacija već postoji!");
56
57         Lokacije.Add(l);
58     }
59     else if (opcija == 1)
60     {
61         Lokacija location = null;
62         foreach (Lokacija lokacija in Lokacije)
63         {
64             if (lokacija.Naziv == l.Naziv)
65             {
66                 location = lokacija;
67                 break;
68             }
69         }
70         if (location == null)
71             throw new InvalidOperationException("Lokacija ne postoji!");
72
73         Lokacije.Remove(location);
74     }
75     else if (opcija == 2)
76     {
77         Lokacija location = null;
78         foreach (Lokacija lokacija in Lokacije)
79         {
80             if (lokacija.Naziv == l.Naziv)
81             {
82                 location = lokacija;
83                 break;
84             }
85         }
86         if (location == null)
87             throw new InvalidOperationException("Lokacija ne postoji!");
88
89         foreach (string ulica in podaci)
90             if (!location.Ulice.Contains(ulica))
91                 location.Ulice.Add(ulica);
92     }
93 }
```

Kategorija izvršenog *code tuninga* i kratak opis onog što je urađeno:

Ovaj code tuning pripada u kategoriju Tuning logičkih izraza. Unutar foreach petlje, provjerava se da li postoji neka lokacija sa istim nazivom. Ukoliko bude ispunjen uslov u ifu, prekinut ćemo petlju, jer smo pronašli lokaciju i nema potrebe da idemo dalje kroz petlju i tražimo još.

Prikaz programskog koda metode nakon vršenja prvog i drugog *code tuninga*:

Verifikacija i Validacija Softvera

```
2 references | 0/1 passing
public void RadSaLokacijomTuning2(Lokacija l, int opcija, List<string> podaci = null)
{
    string imeTrazeneLokacije = l.Naziv;
    if (opcija == 0)
    {
        Lokacija location = null;
        foreach (Lokacija lokacija in Lokacije)
        {
            if (lokacija.Naziv == imeTrazeneLokacije)
            {
                location = lokacija;
                break;
            }
        }
        if (location != null)
            throw new InvalidOperationException("Lokacija već postoji!");

        Lokacije.Add(l);
    }
    else if (opcija == 1)
    {
        Lokacija location = null;
        foreach (Lokacija lokacija in Lokacije)
        {
            if (lokacija.Naziv == imeTrazeneLokacije)
            {
                location = lokacija;
                break;
            }
        }
        if (location == null)
            throw new InvalidOperationException("Lokacija ne postoji!");

        Lokacije.Remove(location);
    }
    else if (opcija == 2)
    {
        Lokacija location = null;
        foreach (Lokacija lokacija in Lokacije)
        {
            if (lokacija.Naziv == imeTrazeneLokacije)
            {
                location = lokacija;
                break;
            }
        }
        if (location == null)
            throw new InvalidOperationException("Lokacija ne postoji!");

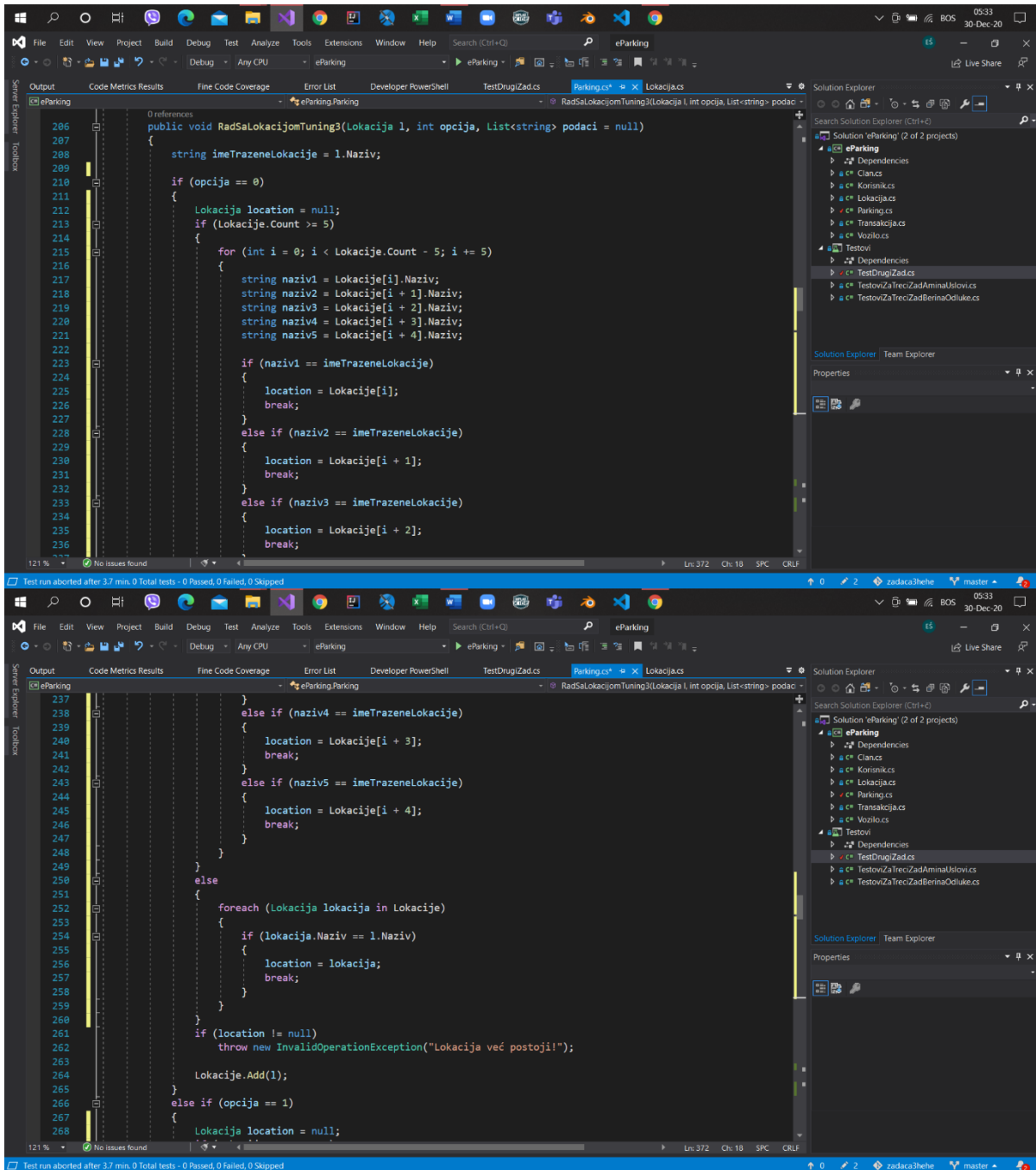
        foreach (string ulica in podaci)
            if (!location.Ulice.Contains(ulica))
                location.Ulice.Add(ulica);
    }
}
```

Kategorija izvršenog *code tuninga* i kratak opis onog što je urađeno:

Kategorija je Tuning logičkih izraza. Da se ne bi svaki put bespotrebno pozivao getter lokacije koja se prima kao parametar u foreach petlji, ime lokacije koje se dobije putem gettera je sačuvano u varijablu izvan svih petlji, i zatim iskorišteno unutar njih.

Prikaz programskog koda metode nakon vršenja svih *code tuninga*:

Verifikacija i Validacija Softvera



```
public void RadSaLokacijomTuning3(Lokacija l, int opcija, List<string> podaci = null)
{
    string imeTrazeneLokacije = l.Naziv;

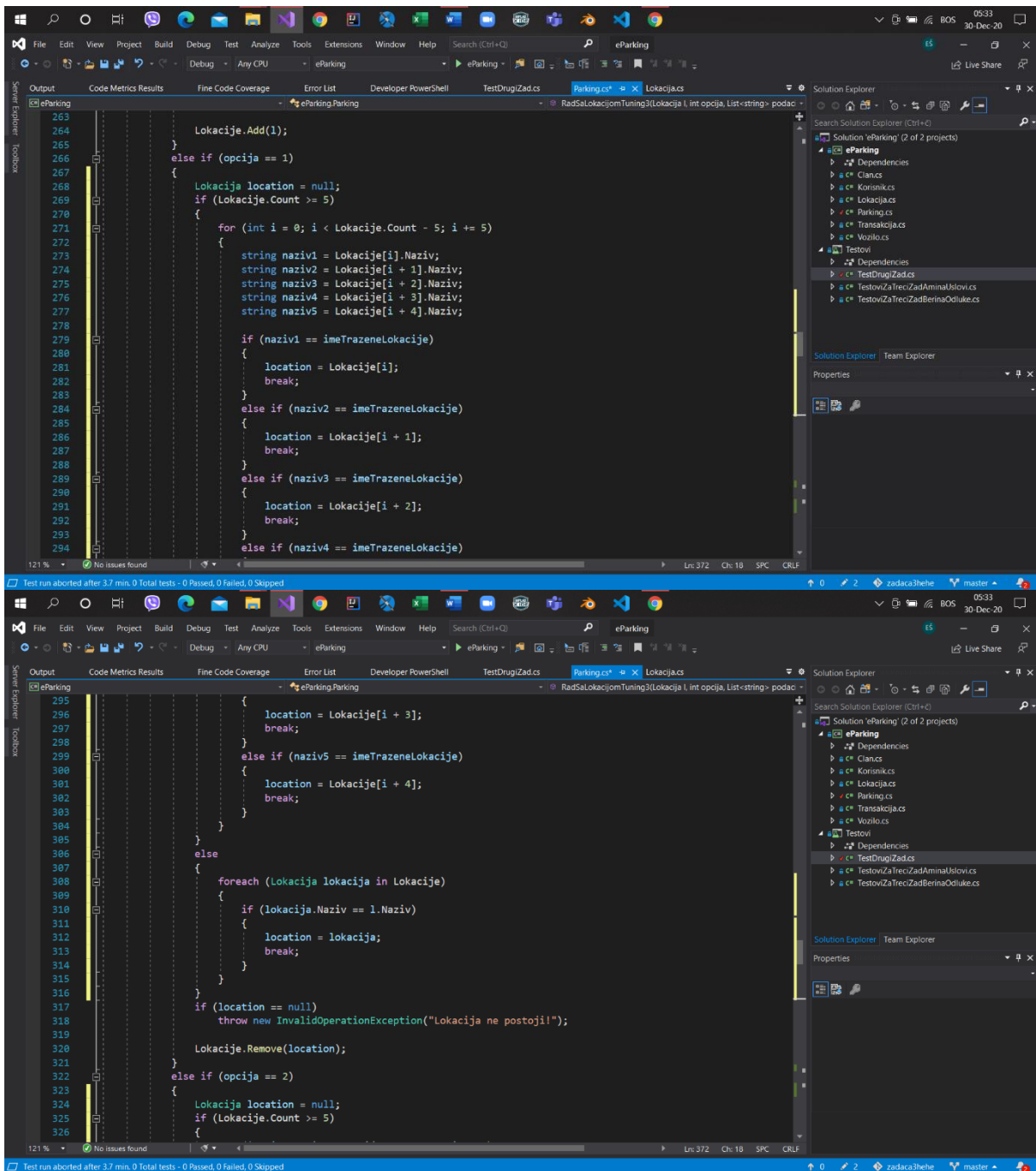
    if (opcija == 0)
    {
        Lokacija location = null;
        if (Lokacije.Count >= 5)
        {
            for (int i = 0; i < Lokacije.Count - 5; i += 5)
            {
                string naziv1 = Lokacije[i].Naziv;
                string naziv2 = Lokacije[i + 1].Naziv;
                string naziv3 = Lokacije[i + 2].Naziv;
                string naziv4 = Lokacije[i + 3].Naziv;
                string naziv5 = Lokacije[i + 4].Naziv;

                if (naziv1 == imeTrazeneLokacije)
                {
                    location = Lokacije[i];
                    break;
                }
                else if (naziv2 == imeTrazeneLokacije)
                {
                    location = Lokacije[i + 1];
                    break;
                }
                else if (naziv3 == imeTrazeneLokacije)
                {
                    location = Lokacije[i + 2];
                    break;
                }
                else if (naziv4 == imeTrazeneLokacije)
                {
                    location = Lokacije[i + 3];
                    break;
                }
                else if (naziv5 == imeTrazeneLokacije)
                {
                    location = Lokacije[i + 4];
                    break;
                }
            }
        }
        else
        {
            foreach (Lokacija lokacija in Lokacije)
            {
                if (lokacija.Naziv == l.Naziv)
                {
                    location = lokacija;
                    break;
                }
            }
        }

        if (location != null)
            throw new InvalidOperationException("Lokacija već postoji!");

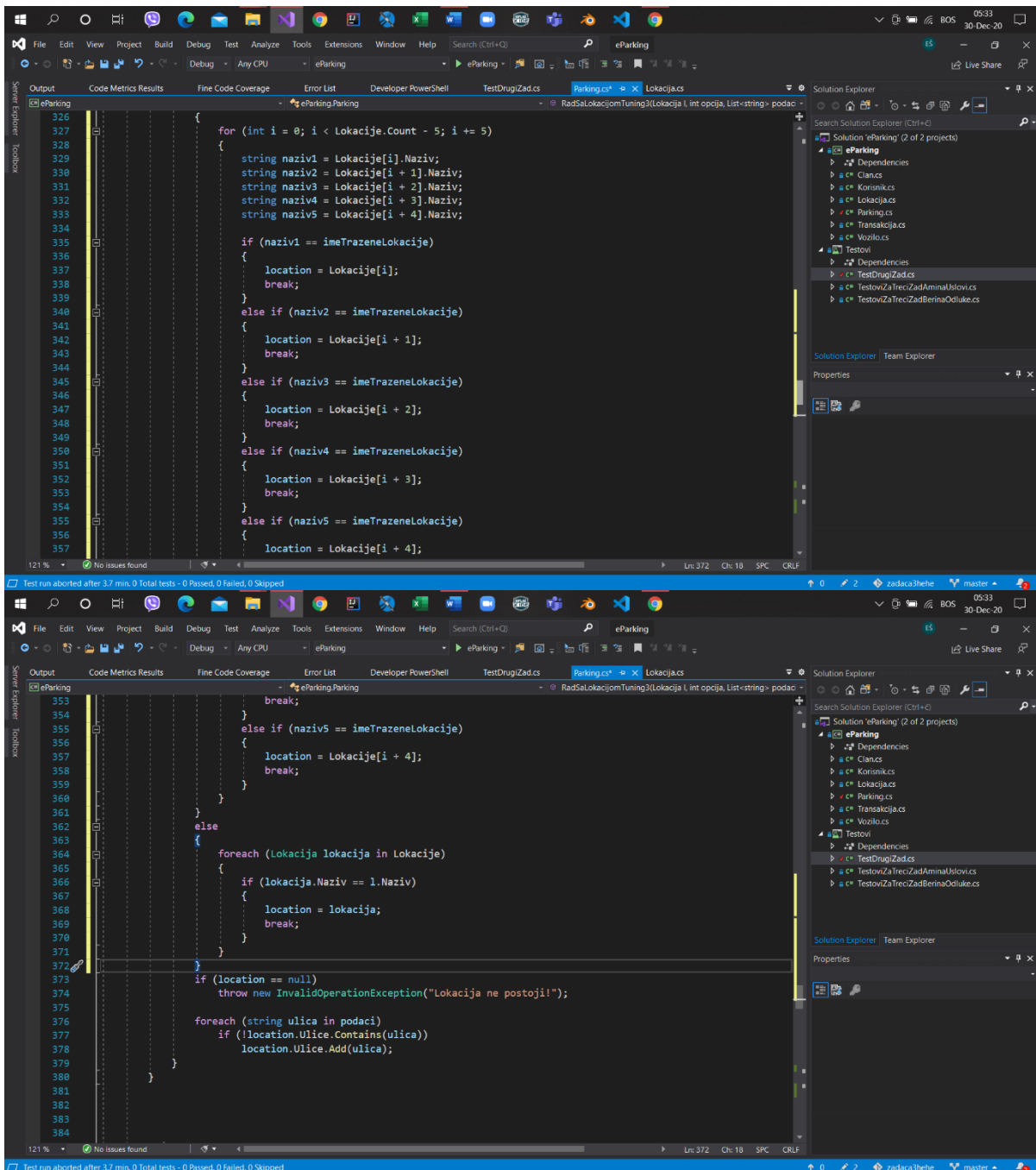
        Lokacije.Add(l);
    }
    else if (opcija == 1)
    {
        Lokacija location = null;
    }
}
```

Verifikacija i Validacija Softvera



```
263     Lokacije.Add(l);
264
265 }
266 else if (opcija == 1)
267 {
268     Lokacija location = null;
269     if (Lokacije.Count >= 5)
270     {
271         for (int i = 0; i < Lokacije.Count - 5; i += 5)
272         {
273             string naziv1 = Lokacije[i].Naziv;
274             string naziv2 = Lokacije[i + 1].Naziv;
275             string naziv3 = Lokacije[i + 2].Naziv;
276             string naziv4 = Lokacije[i + 3].Naziv;
277             string naziv5 = Lokacije[i + 4].Naziv;
278
279             if (naziv1 == imeTrazenLokacije)
280             {
281                 location = Lokacije[i];
282                 break;
283             }
284             else if (naziv2 == imeTrazenLokacije)
285             {
286                 location = Lokacije[i + 1];
287                 break;
288             }
289             else if (naziv3 == imeTrazenLokacije)
290             {
291                 location = Lokacije[i + 2];
292                 break;
293             }
294             else if (naziv4 == imeTrazenLokacije)
295             {
296                 location = Lokacije[i + 3];
297                 break;
298             }
299             else if (naziv5 == imeTrazenLokacije)
300             {
301                 location = Lokacije[i + 4];
302                 break;
303             }
304         }
305     }
306     else
307     {
308         foreach (Lokacija lokacija in Lokacije)
309         {
310             if (lokacija.Naziv == l.Naziv)
311             {
312                 location = lokacija;
313                 break;
314             }
315         }
316     }
317     if (location == null)
318         throw new InvalidOperationException("Lokacija ne postoji!");
319
320     Lokacije.Remove(location);
321 }
322 else if (opcija == 2)
323 {
324     Lokacija location = null;
325     if (Lokacije.Count >= 5)
326     {
```

Verifikacija i Validacija Softvera



```
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384
```

Test run aborted after 3.7 min. 0 Total tests - 0 Passed, 0 Failed, 0 Skipped

Kategorija izvršenog *code tuninga* i kratak opis onog što je urađeno:

Postojeću petlju moguće je djelimično odmotati, ručnim izračunavanjem nekoliko iteracija u okviru jednog prolaza kroz petlju. Pritom se vrijednost brojača petlje prestaje sekvencijalno uvećavati za 1 – umjesto toga, sekvencijalno uvećavanje se vrši 5 puta. Ovaj tuning spada u tuning petlje. Također, budući da bi došlo do izuzetka ukoliko bi se ostavilo da petlja ide do `Lokacije.Count` stavljeno je da ide do `Lokacije.Count-5` da ne bi došlo do nedozvoljenog pristupa indeksu. Također, iako sam ovo uradila zbog provjere smanjenja zauzeća memorijskih i procesorskih resursa pri slanju kolekcija sa velikim brojem elemenata koji svakako neće biti 5, dodala sam i jedan uslov budući da nisam željela da napravim metodu koja će smanjivati zauzeće resursa tako velikih kolekcija, a dolaziti će do problema kada radi sa naprimjer 3 elementa. Također, budući da je for petlja prilično duga i zauzela je dosta prostora

Verifikacija i Validacija Softvera

ponavljanjem tri puta u svakom if i else if uslovu, neophodan bi bio refaktoring naveden u prvom zadatku, međutim zbog samog zadatka to nisam učinila ovdje. Također, budući da je metoda sada jako duga, slike su dodane na ovaj način i nisu rezane budući da mi je ovako bilo lakše pratiti i nadam se da to ne predstavlja problem.

U tabelu ispod potrebno je unijeti vrijednosti vremena izvršavanja i zauzeća resursa od strane svih pojedinačnih metoda sa vršenjem code tuninga. Sve metode kao parametar trebaju primati listu sa 1,000,000 elemenata. Potrebno je vršiti usrednjavanje dobivenih vrijednosti iz 10 izvršavanja.

Implementacija metode	Vrijeme izvršavanja (ms)	Zauzeće memorije (MB)	Zauzeće procesora (%)
Bez code tuninga	3170	0.000263	13%
Tuning 1	2781	0.000179	13%
Tuning 2	2573	0,0001	13%
Tuning 3	2316	0,0001	12%

Koja implementacija pokazuje najbolje rezultate? Da li je to posljednja implementacija, nakon vršenja svih *code tuninga*? Ukoliko ne, zbog čega je to tako? Šta je razlog zbog čega ostale implementacije ne pokazuju jednako dobre rezultate?

Obzirom na rezultate u prethodnoj tabeli, možemo vidjeti da sve tri implementacije pokazuju skoro podjednako iste rezultate. Vrijeme izvršavanja se kao linearno smanjuje, kao i zauzeće memorije. Jedina razlika jeste što se u posljednjoj implementaciji smanjilo i zauzeće procesora, pa bi mogli reći da je posljednja implementacija pokazala najbolje rezultate. pokazuje implementacija Tuning 3, što je ujedno i posljednja implementacija, nakon vršenja svih code tuninga. Razlog zbog čega ostale implementacije nisu pokazale jednake rezultate što se tiče zauzeća procesora jeste što se baš u posljednjoj implementaciji pokušavalo pronaći rješenje misleći na poboljšanje zauzeća procesora, dok se prilikom osmišljavanja ostalih tuninga nije obraćala pažnja na to.

Zadatak 3. (White box testiranje)

Za svaku nastavnu grupu definisana je različita metoda za koju će biti izvršeno white box testiranje.

Metoda koja je dodijeljena za white box testiranje: Parking.IzračunajTrenutnuZaradu

Prikaz programskog koda metode:

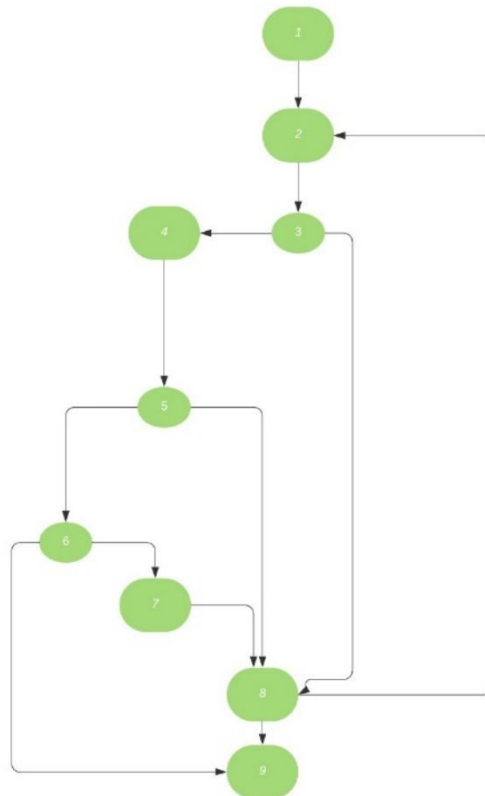
```
0 references
public double IzračunajTrenutnuZaradu()
{
    double zarada = 0.0;
    foreach(Korisnik k in korisnici)
    {
        if (k is Clan)
        {
            var mjesto = ((Clan)k).RezervisanoParkingMjesto;
            if (mjesto != null)
            {
                if (mjesto.Item2.Ulice.Count == 0)
                    throw new ArgumentNullException("Lokacija nema ulica!");

                zarada += mjesto.Item2.Cijena / mjesto.Item2.Ulice.Count;
            }
        }
    }

    return zarada;
}
```

Potrebno je napraviti graf programskog koda za dodijeljenu metodu. Svi čvorovi na grafu trebaju biti numerisani.

Prikaz grafa programskog toka metode:



U tabelu ispod potrebno je unijeti sve puteve kroz graf. Ukoliko se neki od puteva nikada ne može izvršiti zbog međusobne kontradiktornosti uslova ili drugih razloga, **takve puteve potrebno je označiti crvenom bojom.**

Redni broj puta	Čvorovi koje put obuhvata
01.	1-2-3-4-5-6-9
02.	1-2-3-4-5-6-7-8-9
03.	1-2-3-4-5-8-9
04.	1-2-3-8-9
05.	1-2-3-4-5-6-7-8-2
06.	1-2-3-4-5-8-2
07.	1-2-3-8-2

Šta je razlog zbog čega se putevi označeni crvenom bojom nikada ne mogu izvršiti (ukoliko takvi postoje):

Click or tap here to enter text.

Potrebno je formirati sve testne slučajeve za potpuni obuhvat puteva, linija koda i neovisnih puteva. Svaki član tima treba formirati testne slučajeve za jedan potpuni obuhvat. Testni slučajeve podrazumjevaju i specifikaciju vrijednosti parametara i ostalih varijabli koje će dovesti do izvršavanja određenih linija koda, kako je prethodno formiranim putevima

definisano. Nakon toga potrebno je izvršiti diskusiju o razlikama između formiranih testnih slučajeva.

Navesti sve testne slučajeve za postizanje potpunog obuhvata puteva:

Trebamo obuhvatiti sve puteve.

Za prvi put je potrebno pozvati metodu IzračunajTrenutnuZaradu nad instancom klase Parking, nazovimo je parking. U parking ćemo dodati clana new Clan(DateTime.Now.AddYears(2)) (drugi parametar metode DodajKorisnika – bool clan će biti postavljen na true). Clanu je rezervisano parking mjesto linijom clan.RezervišiMjesto(lokacija), gdje je lokacija new Lokacija("Lokacija", ulice, 1, 3), a ulice su niz stringova koji ne sadrži nijedan član, tj broj elemenata ove liste je nula.

Za drugi put je potrebno pozvati metodu IzračunajTrenutnuZaradu nad instancom klase Parking, nazovimo je parking. U parking ćemo dodati clana new Clan(DateTime.Now.AddYears(2)) (drugi parametar metode DodajKorisnika – bool clan će biti postavljen na true). Clanu je rezervisano parking mjesto linijom clan.RezervišiMjesto(lokacija), gdje je lokacija new Lokacija("Lokacija", ulice, 1, 3), a ulice su niz stringova koji sadrži samo jedan član, „Bosanska bb“.

Za treći put ćemo samo u parking dodati clana new Clan(DateTime.Now.AddYears(2)) (drugi parametar metode DodajKorisnika – bool clan će biti postavljen na true), i pozvati metodu IzračunajTrenutnu zaradu nad njim.

Za četvrti put ćemo u parking dodati korisnika new Korisnik() (drugi parametar metode DodajKorisnika – bool clan će biti postavljen na false), te pozvati metodu IzračunajTrenutnu zaradu nad njim.

Kako bismo obuhvatili peti put, uradit ćemo isto kao za drugi slučaj, s tim da ćemo u parking dodati još jednog člana, recimo new Clan(DateTime.Now.AddYears(2)).

Za šesti put ćemo u parking dodati dva clana new Clan(DateTime.Now.AddYears(2)), (drugi parametar metode DodajKorisnika – bool clan će biti postavljen na true), te pozvati metodu IzračunajTrenutnu zaradu nad njim.

I konačno, za sedmi put ćemo u parking dodati dva korisnika new Korisnik(), (drugi parametar metode DodajKorisnika – bool clan će biti postavljen na false), i pozvati nađu metodu nad njim.

Navesti sve testne slučajeve za postizanje potpunog obuhvata linija koda:

Slučaj koji obuhvata čvorove 1-2-3-4-5-6-7-8-9, slučaj 1-2-3-4-5-6-9, zatim slučaj koji obuhvata čvorove 1-2-3-4-5-8-9, te slučaj koji obuhvata čvorove 1-2-3-8-2.

Za prvi put je potrebno pozvati metodu IzračunajTrenutnuZaradu nad instancom klase Parking, nazovimo je parking. U parking ćemo dodati clana new Clan(DateTime.Now.AddYears(2)) (drugi parametar metode DodajKorisnika – bool clan će biti postavljen na true). Clanu je

Verifikacija i Validacija Softvera

rezervisano parking mjesto linijom `clan.RezervišiMjesto(lokalija)`, gdje je lokalija `new Lokacija("Lokacija", ulice, 1, 3)`, a ulice su niz stringova koji sadrži samo jedan član, „Bosanska bb“.

Za drugi put je potrebno pozvati metodu `IzračunajTrenutnuZaradu` nad instancom klase `Parking`, nazovimo je `parking`. U `parking` ćemo dodati člana `new Clan(DateTime.Now.AddYears(2))` (drugi parametar metode `DodajKorisnika` – bool član će biti postavljen na `true`). Članu je rezervisano parking mjesto linijom `clan.RezervišiMjesto(lokalija)`, gdje je lokalija `new Lokacija("Lokacija", ulice, 1, 3)`, a ulice su niz stringova koji ne sadrži nijedan član, tj broj elemenata ove liste je nula.

Za treći put ćemo samo u `parking` dodati člana `new Clan(DateTime.Now.AddYears(2))` (drugi parametar metode `DodajKorisnika` – bool član će biti postavljen na `true`), i pozvati metodu `IzračunajTrenutnu zaradu` nad njim.

I konačno, za četvrti put ćemo u `parking` dodati dva korisnika `new Korisnik()`, (drugi parametar metode `DodajKorisnika` – bool član će biti postavljen na `false`), i pozvati našu metodu nad njim.

Navesti sve testne slučajeve za postizanje potpunog obuhvata neovisnih puteva:

Vrijednost McCabe metrike kompleksnosti je $12 - 9 + 2 = 5$, pa imamo 5 neovisnih puteva. Naprimjer, uzet ćemo puteve 1-2-3-4-5-6-9, 1-2-3-4-5-6-7-8-9, 1-2-3-4-5-8-9, 1-2-3-8-9, i 1-2-3-8-2. Testni slučajevi za ove puteve su sljedeći:

Za prvi put je potrebno pozvati metodu `IzračunajTrenutnuZaradu` nad instancom klase `Parking`, nazovimo je `parking`. U `parking` ćemo dodati člana `new Clan(DateTime.Now.AddYears(2))` (drugi parametar metode `DodajKorisnika` – bool član će biti postavljen na `true`). Članu je rezervisano parking mjesto linijom `clan.RezervišiMjesto(lokalija)`, gdje je lokalija `new Lokacija("Lokacija", ulice, 1, 3)`, a ulice su niz stringova koji sadrži samo jedan član, „Bosanska bb“.

Za drugi put je potrebno pozvati metodu `IzračunajTrenutnuZaradu` nad instancom klase `Parking`, nazovimo je `parking`. U `parking` ćemo dodati člana `new Clan(DateTime.Now.AddYears(2))` (drugi parametar metode `DodajKorisnika` – bool član će biti postavljen na `true`). Članu je rezervisano parking mjesto linijom `clan.RezervišiMjesto(lokalija)`, gdje je lokalija `new Lokacija("Lokacija", ulice, 1, 3)`, a ulice su niz stringova koji ne sadrži nijedan član, tj broj elemenata ove liste je nula.

Za treći put ćemo samo u `parking` dodati člana `new Clan(DateTime.Now.AddYears(2))` (drugi parametar metode `DodajKorisnika` – bool član će biti postavljen na `true`), i pozvati metodu `IzračunajTrenutnu zaradu` nad njim.

Za četvrti put ćemo u `parking` dodati korisnika `new Korisnik()` (drugi parametar metode `DodajKorisnika` – bool član će biti postavljen na `false`), te pozvati metodu `IzračunajTrenutnu zaradu` nad njim.

Verifikacija i Validacija Softvera

Za peti put ćemo u parking dodati dva korisnika `new Korisnik()`, (drugi parametar metode `DodajKorisnika` – `bool` član će biti postavljen na `false`), i pozvati našu metodu nad njim.

Koja je razlika između tri prethodno formirana skupa testnih slučajeva? Koji od potpunih obuhvata zahtijeva najveći, a koji najmanji broj testova? Koje su prednosti, a koje mane odabira svakog od skupova testnih slučajeva?

Najveći broj testova zahtijeva potpun obuhvat puteva, a najmanje potpuni obuhvat linija. Potpuni obuhvat puteva će nam otkriti greške za svaki mogući slučaj, ali zahtijeva veliki broj testova i puno resursa kao što su vrijeme, novac i trud. Obuhvat linija pokriva samo sve linije i otkrit će minimalan broj grešaka, ali će također uštediti resurse. Obuhvat neovisnih puteva omogućava da se testira maksimalni broj puteva sa minimalnim brojem testova, tako što maksimizira koliko svaki test pokrije kod, tako da je negdje između obuhvata puteva i linija koda, zahtijeva manje resursa od obuhvata puteva, ali više od obuhvata puteva, a daje pokrivenost veću od obuhvata puteva a manju od obuhvata linija.

*Potrebno je definisati unit testove koristeći tri strategije testiranja: potpuni obuhvat odluka, uslova i petlji. **Svaki član tima treba izvršiti unit testiranje koristeći jednu strategiju.** Kako bi prikaz pokrivenosti koda metode bila relevantna, potrebno je pokrenuti samo testove iz određene strategije testiranja a zatim prikazati pokrivenost koda metode. Preporučuje se razdvajanje testnih strategija u više testnih klasa i pokretanje samo testova iz jedne testne klase za lakši prikaz pokrivenosti koda.*

Prikaz unit testova za potpuni obuhvat odluka:

Verifikacija i Validacija Softvera

```
10 0 references
11 public class TestoviZaTreciZadBerinaOdLuke
12 {
13     //svi ifovi pozitivni
14     [TestMethod]
15     public void TestIfPozitivno()
16     {
17         Parking parking = new Parking();
18         Clan clan1 = new Clan(DateTime.Now.AddYears(2));
19         Lokacija lokacija1 = new Lokacija("Gorazde", new List<string> { "Nurije Rasidkadica", 5, 30});
20         parking.RadSaLokacijom(lokacija1, 0);
21         clan1.RezervišiMjesto(lokacija1);
22         parking.DodajKorisnika(clan1, true);
23         double zarada = parking.IzračunajTrenutnuZaradu();
24         Assert.AreEqual(zarada, 5);
25     }
26
27     //negativan veliki if, korisnik nije clan
28     [TestMethod]
29     public void TestIfNegativno()
30     {
31         Parking parking = new Parking();
32         Korisnik k1 = new Korisnik();
33         parking.DodajKorisnika(k1, false);
34         double zarada = parking.IzračunajTrenutnuZaradu();
35         Assert.AreEqual(zarada, 0);
36     }
37
38     //pozitivan vanjski if, ali negativan if za mjesto
39     [TestMethod]
40     public void TestIfPozitivnoNegativno()
41     {
42         Parking parking = new Parking();
43         Clan clan1 = new Clan(DateTime.Now.AddYears(2));
44         Lokacija lokacija1 = new Lokacija("Gorazde", new List<string> { "Trg Branilaca", "Nurije Rasidkadica" }, 5, 30);
45         parking.RadSaLokacijom(lokacija1, 0);
46         parking.DodajKorisnika(clan1, true);
47         double zarada = parking.IzračunajTrenutnuZaradu();
48         Assert.AreEqual(zarada, 0);
49     }
50
51     //pozitivan vanjski if, pozitivan if za mjesto, ali negativan if za broj ulica
52     [TestMethod]
53     [ExpectedException(typeof(ArgumentNullException))]
54     public void TestIfPozitivnoPozitivnoNegativno()
55     {
56         Parking parking = new Parking();
57         Clan clan1 = new Clan(DateTime.Now.AddYears(2));
58         Lokacija lokacija1 = new Lokacija("Gorazde", new List<string> {}, 5, 30);
59         parking.RadSaLokacijom(lokacija1, 0);
60         clan1.RezervišiMjesto(lokacija1);
61         parking.DodajKorisnika(clan1, true);
62         double zarada = parking.IzračunajTrenutnuZaradu();
63     }
64
65
66
67 }
```

Prikaz pokrivenosti koda metode nakon testiranja:

Verifikacija i Validacija Softvera

```
334 7 references | 4/7 passing
335 public double IzračunajTrenutnuZaradu()
336 {
337     double zarada = 0.0;
338     foreach(Korisnik k in korisnici)
339     {
340         if (k is Clan)
341         {
342             var mjesto = ((Clan)k).RezervisanoParkingMjesto;
343             if (mjesto != null)
344             {
345                 if (mjesto.Item2.Ulice.Count == 0)
346                     throw new ArgumentNullException("Lokacija nema ulica!");
347                 zarada += mjesto.Item2.Cijena / mjesto.Item2.Ulice.Count;
348             }
349         }
350     }
351     return zarada;
352 }
353
354
355
```

Prikaz unit testova za potpuni obuhvat uslova:

Verifikacija i Validacija Softvera

```
[TestMethod]
✓ | 0 references
public void testSviIfoviZadovoljeni()
{
    Parking parking = new Parking();
    Clan clan = new Clan(DateTime.Now.AddYears(2));
    List<string> ulice = new List<string>();
    ulice.Add("Bosanska bb");
    Lokacija lokacija = new Lokacija("Lokacija", ulice, 1, 3);
    clan.RezervišiMjesto(lokacija);
    parking.DodajKorisnika(clan, true);
    double zarada = parking.IzračunajTrenutnuZaradu();
    Assert.AreEqual(zarada, 1);
}

[TestMethod]
✓ | 0 references
public void TestNemaRezervisanoMjesto()
{
    Parking parking = new Parking();
    Clan clan = new Clan(DateTime.Now.AddYears(2));
    parking.DodajKorisnika(clan, true);
    double zarada = parking.IzračunajTrenutnuZaradu();
    Assert.AreEqual(zarada, 0);
}

[TestMethod]
✓ | 0 references
public void TestNijeClan()
{
    Parking parking = new Parking();
    Korisnik korisnik = new Korisnik();
    parking.DodajKorisnika(korisnik, false);
    double zarada = parking.IzračunajTrenutnuZaradu();
    Assert.AreEqual(zarada, 0);
}

[TestMethod]
[ExpectedException(typeof(ArgumentNullException))]
✓ | 0 references
public void TestNemaUlica()
{
    Parking parking = new Parking();
    Clan clan = new Clan(DateTime.Now.AddYears(2));
    List<string> ulice = new List<string>();
    Lokacija lokacija = new Lokacija("Lokacija", ulice, 1, 3);
    clan.RezervišiMjesto(lokacija);
    parking.DodajKorisnika(clan, true);
    double zarada = parking.IzračunajTrenutnuZaradu();
}
}
```

Prikaz pokrivenosti koda metode nakon testiranja:

Verifikacija i Validacija Softvera

```
359 |
360 | public double IzračunajTrenutnuZaradu()
361 | {
362 |     double zarada = 0.0;
363 |     foreach(Korisnik k in korisnici)
364 |     {
365 |         if (k is Clan)
366 |         {
367 |             var mjesto = ((Clan)k).RezervisanoParkingMjesto;
368 |             if (mjesto != null)
369 |             {
370 |                 if (mjesto.Item2.Ulice.Count == 0)
371 |                     throw new ArgumentNullException("Lokacija nema ulica!");
372 |
373 |                 zarada += mjesto.Item2.Cijena / mjesto.Item2.Ulice.Count;
374 |             }
375 |         }
376 |     }
377 |
378 |     return zarada;
379 | }
380 |
```

Prikaz unit testova za potpuni obuhvat petlji:

```
7 | namespace Testovi
8 | {
9 |     [TestClass]
10 |     public class TestoviZaTreciZadElmaPetlje
11 |     {
12 |
13 |         //prolazak kojim se omogućuje ulazak u petlju
14 |         [TestMethod]
15 |         public void TestUlazak()
16 |         {
17 |             Parking parking = new Parking();
18 |             parking.DodajKorisnika(new Clan(DateTime.Now.AddYears(3)), true);
19 |             double zarada = parking.IzračunajTrenutnuZaradu();
20 |             Assert.AreEqual(zarada, 0.0);
21 |         }
22 |
23 |         //jedan prolazak kroz petlju
24 |         [TestMethod]
25 |         public void TestProlaz1()
26 |         {
27 |             Parking parking1 = new Parking();
28 |             Clan clan1 = new Clan(DateTime.Now.AddYears(3));
29 |             Lokacija lokacija1 = new Lokacija("Bugojno", new List<string> { "Voznik" }, 3, 200);
30 |             parking1.RadSaLokacijom(lokacija1, 0);
31 |             clan1.RezervisiMjesto(lokacija1);
32 |             parking1.DodajKorisnika(clan1, true);
33 |             double zarada = parking1.IzračunajTrenutnuZaradu();
```


Verifikacija i Validacija Softvera

```
34         Assert.AreEqual(zarada, 3);
35     }
36
37     //dva prolaska kroz petlju, takodjer obuhvata i n prolazaka kroz petlju
38     [TestMethod]
39     // 0 references
40     public void TestProlaz2()
41     {
42         Parking parking2 = new Parking();
43         Clan clan2 = new Clan(new DateTime(2023, 12, 25));
44         Lokacija lokacija2 = new Lokacija("VelikaKladuša", new List<string> { "Gromile", "Naselje", "Kalinska" }, 3, 200);
45         parking2.RadSaLokacijom(lokacija2, 0);
46         clan2.RezervisiMjesto(lokacija2);
47         parking2.DodajKorisnika(clan2, true);
48         clan2.Username = "clan2";
49         Clan clan3 = new Clan(new DateTime(2021, 12, 25));
50         Lokacija lokacija3 = new Lokacija("Sarajevo", new List<string> { "Dzemala Bijedica 129", "Ferhadija" }, 2, 200);
51         parking2.RadSaLokacijom(lokacija3, 0);
52         clan3.RezervisiMjesto(lokacija3);
53         clan3.Username = "clan3";
54         parking2.DodajKorisnika(clan3, true);
55         double zarada = parking2.IzracunajTrenutnuZaradu();
56         Assert.AreEqual(2, zarada);
57     }
58
59     //m prolazaka kroz petlju gdje je m<n, m = 3, n = 4, takodjer obuhvata i n-1
60     [TestMethod]
61     [ExpectedException(typeof(ArgumentNullException))]
62     // 0 references
63     public void TestProlazMN()
64     {
65         Parking parking3 = new Parking();
66         Clan clan4 = new Clan(DateTime.Now.AddYears(3));
67         Lokacija lokacija4 = new Lokacija("DonjiVakuf", new List<string> { "Donjici" }, 3, 200);
68         parking3.RadSaLokacijom(lokacija4, 0);
69         clan4.RezervisiMjesto(lokacija4);
70         parking3.DodajKorisnika(clan4, true);
71         Clan clan5 = new Clan(DateTime.Now.AddYears(3));
72         Lokacija lokacija5 = new Lokacija("GornjiVakuf", new List<string> { "Terzici" }, 2.5, 200);
73         parking3.RadSaLokacijom(lokacija5, 0);
74         clan5.RezervisiMjesto(lokacija5);
75         parking3.DodajKorisnika(clan5, true);
76         Clan clan6 = new Clan(DateTime.Now.AddYears(2));
77         Lokacija lokacija6 = new Lokacija("Travnik", new List<string> { }, 1, 30);
78         parking3.RadSaLokacijom(lokacija6, 0);
79         clan6.RezervisiMjesto(lokacija6);
80         parking3.DodajKorisnika(clan6, true);
81         double zarada = parking3.IzracunajTrenutnuZaradu();
82     }
83
84 }
```

Prikaz pokrivenosti koda metode nakon testiranja:

```
460 public double IzračunajTrenutnuZaradu()
461 {
462
463     double zarada = 0.0;
464
465     foreach (Korisnik k in korisnici)
466     {
467         if (k.is Clan)
468         {
469             var mjesto = ((Clan)k).RezervisanoParkingMjesto;
470             if (mjesto != null)
471             {
472                 if (mjesto.Item2.Ulice.Count == 0)
473                     throw new ArgumentNullException("Lokacija nema ulica!");
474
475                 zarada += mjesto.Item2.Cijena / mjesto.Item2.Ulice.Count;
476             }
477         }
478     }
479
480     return zarada;
481 }
482
```

Da li su prikazi pokrivenosti koda za sva tri scenarija isti i zašto?

Prikazi pokrivenosti koda za sva tri scenarija su isti, budući da smo sa sve tri vrste testova obuhvatili cijeli kod.

Na koji način se iz pokrivenosti koda može doći do zaključka koja strategija testiranja je iskorištena?

Obzirom da smo svakom strategijom dobili 100%-tnu pokrivenost koda, ne možemo na osnovu pokrivenosti saznati koja je strategija korištena.

Općenito, npr. ukoliko koristimo strategiju obuhvat petlji, ukoliko imamo neki kod van petlji, a kod petlje je pokriven, na osnovu toga možemo naslutiti da se radi o toj strategiji. Unutar petlji ne mora se svaki if uslov zadovoljiti kada se radi strategija petlji, pa ukoliko neki if nije pokriven, to također sluti na strategiju petlji, a ne na neku drugu.

Također, analogno i za obuhvat grananja i uslova.

Koje su prednosti, a koje mane korištenja svakog od pojedinačnih pristupa testiranja?

Testiranje obuhvatom uslova pomaže kod otkrivanja grešaka nastalih upotrebom logičkih i relacijskih operatora, nepravilno postavljenim zagradama. Nedostatak testiranja obuhvatom petlji jeste što je jako veliki broj testova. Najveću pristupačnost ima testiranje obuhvatom grananja, jer je najjednostavnije jer se preračunati izrazi tj. izrazi kao cijelina postavljaju na true i false.