UNIVERZITET U SARAJEVU
ELEKTROTEHNIČKI FAKULTET SARAJEVO

DOMAĆA ZADAĆA 1
VERIFIKACIJA I VALIDACIJA SOFTVERA
- grupni rad -

Studenti:

1. Mašović Haris – 17993
2. Nermin Krdžić – 17825
3. Amir Muminović – 17744
4. Belmin Divjan – 17827

Odsjek: Računarstvo i Informatika
Datum: 03.11.2018

Tema: Inspekcija

Zadatak c)

“Tim sa vježbi treba da napiše plan provođenja prolaza-kroz (walkthrough pregleda). Potrebno je izvršiti pregled koda prema tom planu i sačiniti odgovarajući izvještaj.”

Plan provođenja prolaza-kroz:

- Kod koji se review-a: **Inspekcija/AutoRentApp**
- Autor programa: Nermin Krdžić
- Osoba koja je tester: Amir Muminović
- Ostali učescnici review-a: Mašović Haris i Divjan Belmin

Tokom sastanka smo se “igrali kompjutera” i za svaki unaprijed definisani testni slučaj smo zapisivali stanje programa (ključne varijabli i njihove vrijednosti) što je evidentirano u nastavku. Plan je prikazan u sljedećem dijelu.

Inspekcija koda – Test cases

Test KreiranjeKorisnikaV1 ima cilj da pokuša inicijalizirati instancu klase klijent bez imena i prezimena. Takav slučaj ne bi trebao biti moguć pa očekujemo da program zaustavi takvu akciju (izuzetkom ili upozorenjem). Testovi KreiranjeKorisnikaV2, KreiranjeKorisnikaV3 i KreiranjeKorisnika V4 testiraju slične podslučajeve - kada nema imena, prezimena ili kad je unesen nevalidan datum.

Dodavanje poruka testiramo sa Dodaj1Poruku. Po konstrukciji klase očito je da može čuvati maksimalno pet poruka. Ako bude više poruka, briše se najstarija. To testiramo sa DodajViseOd5Poruka testom. Zajedno sa ovim testovima možemo izvesti i BrisiPoruku test - testira funkcionalnost brisanja. Ispis poruka testiramo sa IspisiPoruke testom za metodu SadrzajListeObavjestenja.

Testovi ObracunajPraznuListu i VratiVoziloBezVozila testiraju specijalne slučajeve za odgovarajuće metode unutar AutoRentShop klase. Ne očekujemo naplatu ako klijent nije rentao nijedno vozilo. Shodno tome, metoda ObracunajCijenuKoristenja treba da vrati -1.

U donjoj tabeli prikazani su testovi pripremljeni za prolaz kroz kod.

Naziv	Metoda	Ulaz	Očekivani izlaz	Stvarni izlaz
KreiranjeKorisnikaV1	Klijent konstuktor	"", "", DateTime.Now	Izuzetak	System.ArgumentOutOfRangeException
KreiranjeKorisnikaV2	Klijent konstuktor	"", "Prezimović", DateTime.Now	Izuzetak	System.ArgumentOutOfRangeException
KreiranjeKorisnikaV3	Klijent konstuktor	"Ime", "", DateTime.Now	Izuzetak	System.ArgumentOutOfRangeException
KreiranjeKorisnikaV4	Klijent konstuktor	"Ime", "Prezimović", new DateTime()	Izuzetak	Nista
Dodaj1Poruku	DodajPoruku	"Mi volimo VVS!"	"Mi volimo VVS!"	"Mi volimo VVS!"
DodajViseOd5Poruka	DodajPoruku	“Trebala bi biti obrisana!”, ”Mi volimo VVS!" x4, "Mi volimo C#!"	“Mi volimo VVS!” x4, "Mi volimo C#!"	“Mi volimo VVS!” x4, "Mi volimo C#!"
BrisiPoruku	BrisiPoruku	"Mi volimo VVS!"	""	"Mi volimo VVS!" x2
IspisiPoruke	SadrzajListeObavijesti	"R", "A", "D", "I"	RADI	RADI
ObracunajPraznuListu	ObracunajCijenuKoristenja	klijent, "5"	-1	-1
VratiVoziloBezVozila	VratiVozilo	klijent, “5”	NoCarRentedException	NoCarRentedException

- KreiranjeKorisnikaV1

Konstruktor prihvata tri argumenta - ime(i), prezime(p) i datum DateTime(dr).

```
public Klijent(String i, String p, DateTime dr)
```

Linija - Klijent.cs	Ime	Prezime	DatumRodjenja	id	listaObavijesti
20	“”				
21	“”	“”			
22	“”	“”	DateTime.Now		

U liniji 24 poziva se funkcija GenerišiID(). U toj funkciji pokušava se uzeti Ime.Substring(0,1) . Kako Ime ne sadrži nijedno slovo, dolazi do pozivanja izuzetka System.ArgumentOutOfRangeException.

Komentar:

Ovaj test je pokazao da konstruktor klase Klijent nije dovoljno dobro implementiran. Pokušaj kreiranja korisnika bez imena i prezimena će uzrokovati nastajanju izuzetka System.ArgumentOutOfRangeException. Taj izuzetak nam neće ništa reći o pravom razlogu nastajanja izuzetka. Neophodno je prije dodjele atributima provjeriti da li su vrijednosti dostavljene konstruktoru legalne za korištenje.

- ## KreiranjeKorisnikaV2

Ovaj put testiramo konstruktor sa vrijednostima i="", p="Prezimović", DateTime.Now

Linija - Klijent.cs	Ime	Prezime	DatumRodjenja	id	listaObavijesti
20	""				
21	""	Prezimović			
22	""	Prezimović	DateTime.now		

Ponovno, prilikom rada metode GenerišiID() dolazi do iste greške.

Komentar:

Koder je trebao prilikom kreiranja konstruktora da više pazi na ulazne podatke. Neophodno je dodati provjeru da li je Ime prazan string. Ako jeste, treba se obavijestiti korisnik da je zaboravio unijeti ime. Bez toga, program daje izuzetak koji nam ne daje jasnu sliku gdje je problem.

- ## KreiranjeKorisnikaV3

  U ovom slučaju testiramo slučaj kada je i=”Ime”, p=””, dr=DateTime.Now

Linija - Klijent.cs	Ime	Prezime	DatumRodjenja	id	listaObavijesti
20	Ime				
21	Ime	“”			
22	Ime	“”	DateTime.Now		

Dolazi do softverskog otkaza prilikom poziva funkcije GenerišiID().

Komentar:

Postaje očito da funkcija GenerišiID() zahtjeva validne vrijednosti parametara Ime i Prezime. Inače, imat ćemo softverski otkaz.

- ## KreiranjeKorisnikaV4

  U ovom slučaju testiramo slučaj kada je i=”Ime”, p=”Prezimović”, dr= new DateTime()

Linija - Klijent.cs	Ime	Prezime	DatumRodjenja	id	listaObavijesti
20	Ime				
21	Ime	Prezimović			
22	Ime	Prezimović	DateTime()		
24	Ime	Prezimović	DateTime()	ip0101000141	
26	Ime	Prezimović	DateTime()	ip0101000141	Prazna lista

U slučaju da ne damo pravi datum rođenja, program ne otkaže. Kreira se korisnik bez evidentiranog datuma rođenja.

Komentar:

Nedostatak evidentiranog datuma rođena nije fatalan za izvršenje programa nit će uzrokovati veće probleme. Kako bi program bio potpun, neophodno je da i za to vrši validacija. Ako bi se u budućnosti obavljali određeni popusti ili posebne akcije za određene starosne grupe, mogao bi nastati problem zbog nedostatka ovog podatka.

- ## Dodaj1Poruku

  Dodaj1Poruku je test koji će ispitati kako radi metoda DodajPoruku. Podsjetimo se da metoda Dodaj poruku prima parametar s, tipa string te koristi atribut listaObavijesti

```
public void DodajPoruku(String s)
```

Linija - Klijent.cs	s	listaObavijesti	listaObavijesti.Count
40	“Mi volimo VVS!”	{}	0
47	“Mi volimo VVS!”	{“Mi volimo VVS!}	1

Dodaj1Poruku je uspješan test - očekivani rezultat se poklapa sa stvarnim.

Komentar:

Metoda DodajPoruku pravilno dodaje jednu poruku. Prilikom diskusije, primjetili smo da ova metoda ima jednu manu. Naime, ako ponudimo metodi prazan string, ona će alocirati jedno mjesto u listi za prazan string - koji efektivno nema nikakvu informativnu vrijednost. Bio bi dobro kada bi se implementirao mehanizam koji ne dodaje poruku u slučaju da je ona prazna.

- ## DodajViseOd5Poruka

U ovom testu, prvenstveno dodajemo poruku “Trebala bi biti obrisana!”, “Mi volimo VVS” četiri puta te “Mi volimo C#” na kraju.

Dodavanje br:	Linija - Klijent.cs	s	listaObavijesti	listaObavijesti.Count
1.	40	“Trebala bih biti obrisana!”	{}	0
1.	47	“Trebala bih biti obrisana!”	{“Trebala bih biti obrisana!”}	1
2.	40	“Mi volimo VVS!”	{“Trebala bih biti obrisana!”}	1
2.	47	“Mi volimo VVS!”	{“Mi volimo VVS!”, “Trebala bih biti obrisana!”}	2
3.	40	“Mi volimo VVS!”	{“Mi volimo VVS!”, “Trebala bih biti obrisana!”}	2

3.	47	“Mi volimo VVS!”	{“Mi volimo VVS!”, “Mi volimo VVS!”, “Trebala bih biti obrisana!”}	3
4	40	“Mi volimo VVS!”	{“Mi volimo VVS!”, “Mi volimo VVS!”, “Trebala bih biti obrisana!”}	3
4	47	“Mi volimo VVS!”	{“Mi volimo VVS!”, “Mi volimo VVS!”, “Mi volimo VVS!”, “Trebala bih biti obrisana!”}	4
5	40	“Mi volimo VVS!”	{“Mi volimo VVS!”, “Mi volimo VVS!”, “Mi volimo VVS!”, “Trebala bih biti obrisana!”}	4
5	47	“Mi volimo VVS!”	{“Mi volimo VVS!,“ Mi volimo VVS!”, “Mi volimo VVS!”, “Mi volimo VVS!”, “Trebala bih biti obrisana!”}	5
6	40	“Mi volimo C#!”	{“Mi volimo VVS!,“ Mi volimo VVS!”, “Mi volimo VVS!”, “Mi volimo VVS!”, “Trebala bih biti obrisana!”}	5
6	44	“Mi volimo C#!”	{“Mi volimo VVS!,“ Mi volimo VVS!”, “Mi volimo VVS!”, “Mi volimo VVS!”, “Mi volimo C#!”}	5

Komentar:

Prilikom prilaza kroz kod, shvatili smo da ova metoda ima poprilično neefikasnu implementaciju. Uvijek će brisati poruku na zadnjem mjestu, čak i ako je ona *najmlađa*. Shodno tome, listaObavijesti može da sadrži 4 stare obavijesti i samo jednu novu. Predlažemo da se na drugi način riješi spomenuti problem - npr kao red sa First In First Out.

- ## BrisiPoruku

  Za ovaj test neophodno je da listaObavjestenja sadrži poruku “Mi volimo VVS!”. Parametar metode BrisiPoruku s imat će vrijednosti “Mi volimo VVS!”

Linija - Klijent.cs	s	listaObavijesti	listaObavijesti.Count
50	“Mi volimo VVS!”	{“Mi volimo VVS!”}	1
52	“Mi volimo VVS!”	{“Mi volimo VVS!”, “Mi volimo VVS!”}	2

Vidimo da naš test nije prošao. Očekivali smo praznu listu ali naša lista ima 2 elementa.

Komentar:

Inspekcija koda je prikazala da metoda BrisiPoruku zapravo ne briše poruke, već ih dodaje. To je ozbiljna greška i sigurno bi napravila brojne probleme u daljem razvoju produkta. Grešku treba ispraviti. Inače imat ćemo duplu kopiju svake poruke koju smo htjeli obrisati.

- ## IspisiPoruku

  SadžajListeObavijesti bi trebao da izlista sva obavještenja. Dodat ćemo četiri poruke: “R”, “A”, “D”, “I”. Ako je finalni sadržaj varijable rez “RADI”, onda je metoda prošla test. Vrijednosti iz foreach petlje će biti u varijabli s

Linija - Klijent.cs	Iteracija foreach-a	rez	s	listaObavijesti
57	/	“”	/	{“R”, “A”, “D”, “I”}
59	1	“”	“R”	{“R”, “A”, “D”, “I”}
60	1	“R”	“R”	{“R”, “A”, “D”, “I”}
59	2	“R”	“A”	{“R”, “A”, “D”, “I”}
60	2	“RA”	“A”	{“R”, “A”, “D”, “I”}
59	3	“RA”	“D”	{“R”, “A”, “D”, “I”}
60	3	“RAD”	“D”	{“R”, “A”, “D”, “I”}
59	4	“RAD”	“I”	{“R”, “A”, “D”, “I”}
60	4	“RADI”	“I”	{“R”, “A”, “D”, “I”}

Na kraju zaista dobijemo “RADI” što je u skladu sa očekivanim rezultatom.

Komentar:

Metoda je položila test i radi kao očekivano. Implementacija metode je koncizna i efikasna. Nema prijedloga za poboljšanje.

- ## ObracunajPraznuListu

  ObracunajCijenuKoristenja prima dva parametra, k tipa Klijent i brSas tipa String. Od atributa koristi listu tuple-ova, pri čemu x predstavlja jedan entry liste u foreach petlji.

Linija - AutoRentShop.cs	k	brSas	x	iznajmljenaVozila
112	klijent	5	{}	{}

Foreach petlja se neće nijednom izvršiti jer nema niti jedan element. Vratit će se -1, što je očekivani rezultat.

Komentar:

Trenutna implementacija metode ObracunajCijenuKoristneja ima jedan mogući propust. Ako korisnik nije iznajmio ijedno auto onda se vraća -1. Ako jeste, vratit će se neka vrijednost tipa Double. Ako direktno vežemo ovu metodu sa sistemom za plaćanje, može doći do situacije da AutoRentShop je dužan da plati klijentu $1 - (zbog vraćanja negativne vrijednosti). To je mogući exploit koji bi mogao biti rizik za AutoRentShop. Predlažemo da metoda vraća 0 umjesto -1.

- ## VratiVoziloBezVozila

Testirat ćemo metodu VratiVozilo na praznoj listi. Trebala bi da vrati NoCarRentedExceptiion.

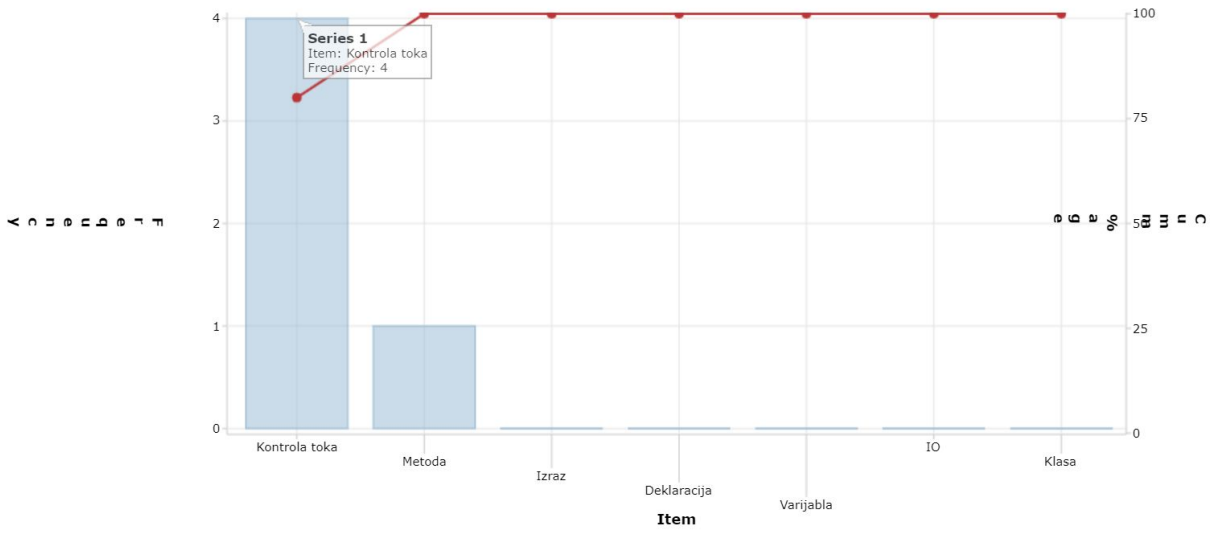
Linija - AutoRentShop.cs	k	brSas	x	iznajmljenaVozila	rez	x’	listaCekanja
83	klijent	5	{}	{}	/	{}	{}

Kako neće biti pronađeno nijedno iznajmljeno vozilo, program će baciti izutetak NoCarRentedExcpetion.

Komentar:

Program se je ponašao kao očekivano za posebni slučaj. Možemo primjetiti da su metode iz AutoRentShop-a detaljnije urađene te više otporne na nepredviđeno ponašanje programa.

Kategorija greške	Broj grešaka	Kumulativno	Kumilativni %
Kontrola toka	4	4	80%
Metoda	1	5	100%
Izraz	0	5	100%
Deklaracija	0	5	100%
Varijabla	0	5	100%
IO	0	5	100%
Klasa	0	5	100%



Tema: White box

“Za kod za koji vršena inspekcija odabrati dvije metode i planirati testne slučajeve”

- Izabrani kod: **Inspekcija/AutoRentApp**
- Izabrane metode:
    1. **Calculate iz klase Klijent.cs**
    2. **VratiVozilo iz klase AutoRentShop.cs**

```
public decimal Calculate(decimal amount, int type, int years)
{
    decimal result = 0;
    decimal disc = (years > 5) ? (decimal)5 / 100 : (decimal)years / 100;
    if (type == 1)
    {
        result = amount;
    }
    else if (type == 2)
    {
        result = (amount - (0.1m * amount)) - disc * (amount - (0.1m * amount));
    }
    else if (type == 3)
    {
        result = (0.7m * amount) - disc * (0.7m * amount);
    }
    else if (type == 4)
    {
        result = (amount - (0.5m * amount)) - disc * (amount - (0.5m * amount));
    }
    return result;
}
```

```
public Double VratiVozilo(Klijent k, String brSas)
{
    if (!iznajmljenaVozila.Any(x => x.Item1.Id.Equals(k.Id) && x.Item2.BrojSasije.Equals(brSas)))
        throw new NoCarRentedException("Klijent sa tim ID-em nema iznajmljeno vozilo.");

    Double rez = ObracunajCijenuKoristenja(k, brSas);

    foreach (Tuple<Klijent, MotornoVozilo, DateTime> x in iznajmljenaVozila)
    {
        if (x.Item1.Id.Equals(k.Id) && x.Item2.BrojSasije.Equals(brSas))
        {
            iznajmljenaVozila.Remove(x);
            break;
        }
    }

    foreach (Tuple<Klijent, MotornoVozilo, DateTime> x in listaCekanja.OrderBy(x=>x.Item3))
    {
        if (x.Item2.BrojSasije.Equals(brSas))
        {
            x.Item1.DodajPoruku("Auto " + x.Item2.ToString() + " koji ste nedavno tražili je sada dostupno.");
            listaCekanja.Remove(x);
            break;
        }
    }

    return rez;
}
```

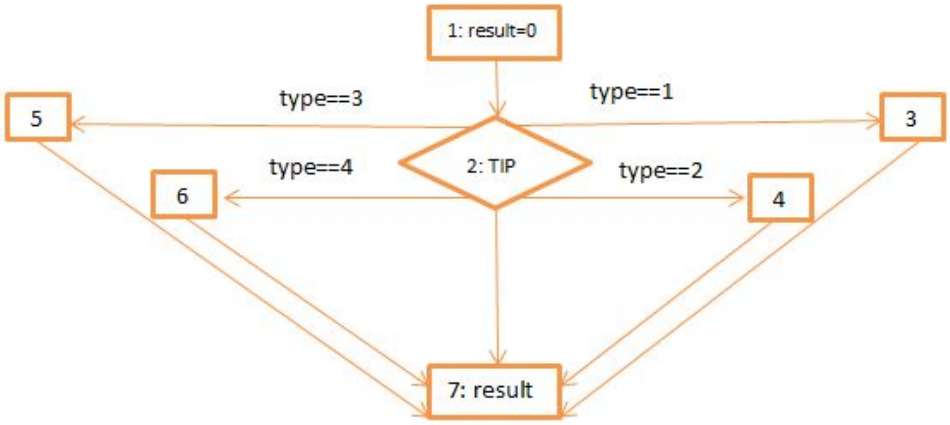
Obuhvat puteva:

Motivacija testiranja puteva je da se postigne kompletan obuhvat programskog koda testirajući sve njegove moguće puteve. Za određivanja broja puteva od pomoći je dijagram toka (flow chart).

1. Metoda **Calculate** : **Klijent.cs**

Metoda Calculate vraća vrijednost result koja se računa na sljedeći način:
(1) Ako je paramater type=1, onda je result=amount
(2) Ako je parameter type=2, onda je result=(amount-(0.1m*amount))-disc*(amount-(0.1m*amount))
(3) Ako je parameter type=3, onda je result=(0.7m*amount)-disc*(0.7m*amount)
(4) Ako je parametar type=4, onda je result=(amount-(0.5m*amount))-disc*(amount-(0.5m*amount))
(5) Za bilo koju drugu vrijednost paramtera type, result=0

Dijagram toka modula za ovu metodu je prikazan na sljedećoj slici:



Na dijagramu toka uočavamo 5 različitih mogućih puteva, što znači da je potrebno, da bi se postigao puni obuhvat puteva, pripremiti najmanje 5 testnih slučajeva. To je prikazano u narednoj tabeli:

Broj puta	Put
1	1-2-2-6
2	1-2-3-6
3	1-2-4-6
4	1-2-5-6
5	1-2-6

Testiranje obuhvata puteva metode Calculate:

```
class Program
{
    Klijent klijent;
    static void Main(string[] args)
    {
        Program program = new Program();
        program.klijent = new Klijent("Neko", "Nekic", new DateTime(1998, 1, 11));
        testPut1(program);
        testPut2(program);
        testPut3(program);
        testPut4(program);
        testPut5(program);
        Console.ReadLine();
    }
    static void testPut1(Program program) => Console.WriteLine(program.klijent.Calculate(15.0m, 1, 7));
    static void testPut2(Program program) => Console.WriteLine(program.klijent.Calculate(15.0m, 2, 7));
    static void testPut3(Program program) => Console.WriteLine(program.klijent.Calculate(15.0m, 3, 7));
    static void testPut4(Program program) => Console.WriteLine(program.klijent.Calculate(15.0m, 4, 7));
    static void testPut5(Program program) => Console.WriteLine(program.klijent.Calculate(15.0m, 10, 7));
}
```

2. Metoda **vratiVozilo** : **AutoRentShop.cs**

Metoda vratiVozilo vraća vrijednost rez, te prolazi kroz listu iznajmljenih vozila i vraćeno vozilo izbacuje iz te liste:

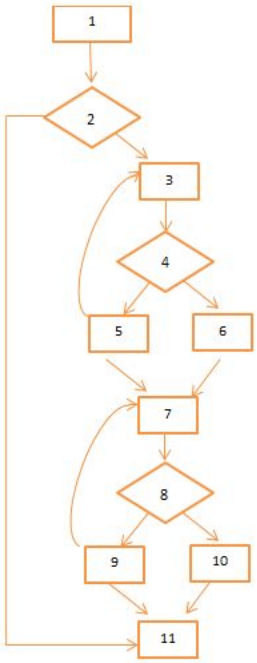
(1) Ako u listi nema vozilo sa tim id-em i brojem šasije, baca se izuzetak

(2) Ako je vozilo u listi, obračunava se cijena ulazi se u petlju koja prolazi listom

(3) U petlji se prolazi kroz listu, i kada se ispuni uslov(pronađe odgovarajuće vozilo), izbacuje se iz liste i petlja se prekida

(4) Ulazi se u sljedeću petlju, i kada se ispuni uslov, ispisuje se poruka te se vozilo izbacuje iz liste čekanja

Dijagram toka modula za ovu metodu je prikazan na sljedećoj slici: (uslov petlje nije provjeravan, jer lista ne može biti prazna zbog izuzetka)



Posmatrat ćemo slučaj kada je veličina liste n=1 odnosno kad znamo sigurno da je u listi naše vozilo. Na dijagramu toka tada uočavamo 2 različita moguća puta, što znači da je potrebno, da bi se postigao puni obuhvat puteva, pripremiti najmanje 2 testna slučaja. To je prikazano u narednoj tabeli:

Broj puta	Put
1	1-2-11
2	1-2-3-4-6-7-8-10-11

Testiranje obuhvata puteva metode Calculate:

```
class Program
{
    static void Main(string[] args)
    {
        //testiranje za put 1, tj. bacanja izuzetka jer je lista prazna
        AutoRentShop autoRentShop = new AutoRentShop();
        Klijent klijent = new Klijent("Neko", "Nekic", DateTime.Now);
        try
        {
            double rez = autoRentShop.VratiVozilo(klijent, "1A2B3C4567G111500");
            Console.WriteLine("Cijena je: " + rez.ToString());
        }
        catch(Exception e)
        {
            Console.WriteLine(e.ToString());
        }
        Console.ReadLine();
    }
}
```

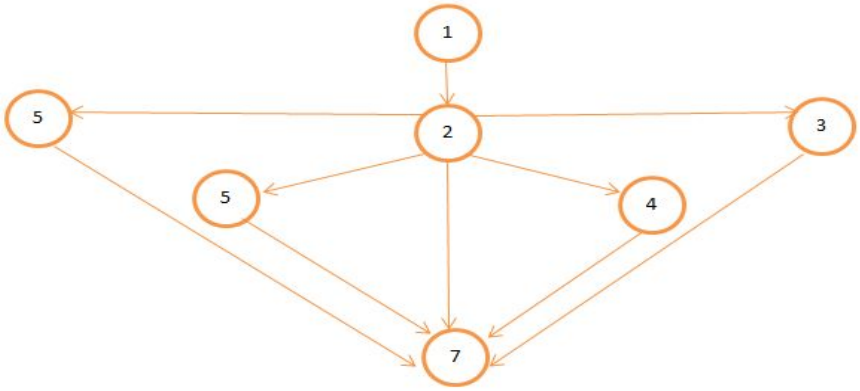
```
static void Main(string[] args)
{
    //testiranje za put 2, tj. slučaja kada je u listi jedan klijent
    AutoRentShop autoRentShop = new AutoRentShop();
    Klijent klijent = new Klijent("Neko", "Nekic", DateTime.Now);
    autoRentShop.klijenti.Add(klijent);
    autoRentShop.RegistrujNovoVozilo(new TeretnoVozilo("vozilo", "vozilo", "1A2B3C4567G111500", 5));
    autoRentShop.IznajmiAuto(klijent, "1A2B3C4567G111500", DateTime.Now);
    try
    {
        double rez = autoRentShop.VratiVozilo(klijent, "1A2B3C4567G111500");
        Console.WriteLine("Cijena je: " + rez.ToString());
    }
    catch(Exception e)
    {
        Console.WriteLine(e.ToString());
    }
    Console.ReadLine();
}
```

Obuhvat iskaza/linija:

Obuhvat iskaza/linija (statement/line coverage) strategija ima težnju da dizajnira testne slučajeve tako da se svaki iskaz tj. svaka linija koda u programu izvrši barem jednom. Za određivanja broja linija od pomoći je graf programskog toka (flow graph).

1. Metoda **Calculate** : **Klijent.cs**

Dijagram programskog toka ovu metodu je prikazan na sljedećoj slici:



Sa grafa programskog toka može se utvrditi da se puni linijski obuhvat može postići sa ispitivanjem minimalnog broja puteva – ukupno 4 kako je prikazano u narednoj tabeli:

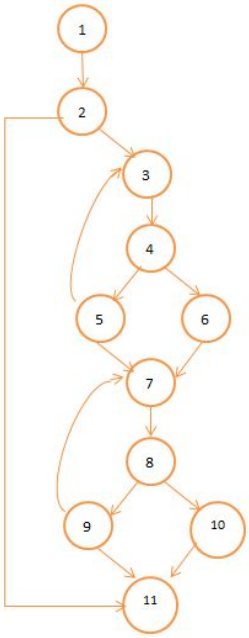
Broj puta	Put
1	1-2-3-7
2	1-2-4-7
3	1-2-5-7
4	1-2-6-7

Testiranje obuhvata iskaza/linija metode Calculate:

```
class Program
{
    Klijent klijent;
    static void Main(string[] args)
    {
        Program program = new Program();
        program.klijent = new Klijent("Neko", "Nekic", new DateTime(1998, 1, 11));
        testPut1(program);
        testPut2(program);
        testPut3(program);
        testPut4(program);
        Console.ReadLine();
    }
    static void testPut1(Program program) => Console.WriteLine(program.klijent.Calculate(15.0m, 1, 7));
    static void testPut2(Program program) => Console.WriteLine(program.klijent.Calculate(15.0m, 2, 7));
    static void testPut3(Program program) => Console.WriteLine(program.klijent.Calculate(15.0m, 3, 7));
    static void testPut4(Program program) => Console.WriteLine(program.klijent.Calculate(15.0m, 4, 7));
}
```

2. Metoda **vratiVozilo** : **AutoRentShop.cs**

Dijagram programskog toka ovu metodu je prikazan na sljedećoj slici:



Sa grafa programskog toka može se utvrditi da se puni linijski obuhvat može postići sa ispitivanjem najmanje 2 puta, kako je prikazano u narednoj tabeli (za veličinu liste n=1):

Broj puta	Put
1	1-2-11
2	1-2-3-4-6-7-8-10-11

Testiranje obuhvata iskaza/linija metode vratiVozilo:

```
class Program
{
    static void Main(string[] args)
    {
        //testiranje za put 1, tj. bacanja izuzetka jer je lista prazna
        AutoRentShop autoRentShop = new AutoRentShop();
        Klijent klijent = new Klijent("Neko", "Nekic", DateTime.Now);
        try
        {
            double rez = autoRentShop.VratiVozilo(klijent, "1A2B3C4567G111500");
            Console.WriteLine("Cijena je: " + rez.ToString());
        }
        catch(Exception e)
        {
            Console.WriteLine(e.ToString());
        }
        Console.ReadLine();
    }
}
```

```
static void Main(string[] args)
{
    //testiranje za put 2, tj. slučaja kada je u listi jedan klijent
    AutoRentShop autoRentShop = new AutoRentShop();
    Klijent klijent = new Klijent("Neko", "Nekic", DateTime.Now);
    autoRentShop.klijenti.Add(klijent);
    autoRentShop.RegistrujNovoVozilo(new TeretnoVozilo("vozilo", "vozilo", "1A2B3C4567G111500", 5));
    autoRentShop.IznajmiAuto(klijent, "1A2B3C4567G111500", DateTime.Now);
    try
    {
        double rez = autoRentShop.VratiVozilo(klijent, "1A2B3C4567G111500");
        Console.WriteLine("Cijena je: " + rez.ToString());
    }
    catch(Exception e)
    {
        Console.WriteLine(e.ToString());
    }
    Console.ReadLine();
}
```

Obuhvat grana:

1. Metoda **Calculate** : **Klijent.cs**

```
static void Main(string[] args)
{
    Program program = new Program();
    program.klijent = new Klijent("ime", "prezime", new DateTime(1997, 5, 21));
    test1(program);
    test2(program);
    test3(program);
    test4(program);
    test5(program);
}
//  svaki od testova obuhvata 20% grana sto je ukupno 100%
1 reference
static void test1(Program program) => Console.WriteLine(program.klijent.Calculate(45.0m, 1, 5));
1 reference
static void test2(Program program) => Console.WriteLine(program.klijent.Calculate(45.0m, 2, 5));
1 reference
static void test3(Program program) => Console.WriteLine(program.klijent.Calculate(45.0m, 3, 5));
1 reference
static void test4(Program program) => Console.WriteLine(program.klijent.Calculate(45.0m, 4, 5));
1 reference
static void test5(Program program) => Console.WriteLine(program.klijent.Calculate(45.0m, 1000, 5));
```

- Ovim testovima je omogucena 100% - tna obuhvatnost grana metode **Calculate** iz klase **Klijent.** Radi se o trivijalnim testovima promjene vrijednosti ulaznog parametra **type** metode **Calculate**.

2. Metoda **VratiVozilo** : **AutoRentShop.cs**
    - Da bi dobili 100% - tnu pokrivenost grana za ovu metodu trebat ce nam 3 testa.
    - Prvi test bi trebao da ispuni prvi uslov koji se javlja u metodi, drugi test bi trebao da ispuni dva uslova koja se javljaju u petljama za uklanjanje vozila iz liste iznajmljenih vozila i uklanjanje vozila iz liste cekanja. Treci test bi bio test koji uklanja vozilo iz liste iznajmljenih vozila ali ga ne uklanja iz liste cekanja jer niko nije cekao na to vozilo.

```
0 references
static void Main(string[] args)
{
    AutoRentShop autoRentShop = new AutoRentShop();
    autoRentShop.klijenti.Add(new Klijent("Nermin", "Krdzic", new DateTime(1997, 5, 21)));
    autoRentShop.klijenti.Add(new Klijent("ime", "prezime", new DateTime(1987, 7, 1)));
    autoRentShop.RegistrujNovoVozilo(new TeretnoVozilo("Leteci", "Tanjir", "L1234T43210000000", 100));
    autoRentShop.RegistrujNovoVozilo(new TeretnoVozilo("Undefined", "Pokemon", "U1234P43210000001", 20));
    autoRentShop.RegistrujNovoVozilo(new TeretnoVozilo("Zeleno", "Cudoviste", "Z1234C43210000002", 100));
    autoRentShop.RegistrujNovoVozilo(new TeretnoVozilo("Smart", "Insider", "L1234I43210000003", 100));
    autoRentShop.IznajmiAuto(autoRentShop.klijenti[0], "L1234T43210000000", DateTime.Today);
    autoRentShop.IznajmiAuto(autoRentShop.klijenti[1], "U1234P43210000001", DateTime.Today);
    autoRentShop.IznajmiAuto(autoRentShop.klijenti[1], "L1234T43210000000", DateTime.Today);

    try { test1(autoRentShop); }
    catch(NoCarRentedException e) { Console.WriteLine(e.Message); }

    test2(autoRentShop);
    test3(autoRentShop);

}
1 reference
static void test1(AutoRentShop autoRentShop) //obuhvatnost je 25% grana
    => Console.WriteLine(autoRentShop.VratiVozilo(autoRentShop.klijenti[0], "Z1234C43210000002"));
1 reference
static void test2(AutoRentShop autoRentShop) // obuhvatnost je 50% grana
    => Console.WriteLine(autoRentShop.VratiVozilo(autoRentShop.klijenti[0], "L1234T43210000000"));
1 reference
static void test3(AutoRentShop autoRentShop) // obuhvatnost je 25%
    => Console.WriteLine(autoRentShop.VratiVozilo(autoRentShop.klijenti[1], "U1234P43210000001"));
```

- Posmatrajuci testove vidimo da test1 pokusava vratiti vozilo koje nije iznajmljeno cime se baca izuzetak.
- Test 2 vraca vozilo koje je bilo iznajmljeno i obavijestava drugog klijenta koji je cekao na to vozilo, da je vozilo spremno za njega.
- Test 3 samo vraca iznajmljeno vozilo.
- Ovdje nismo mogli testirati slucaj u kojem nije niti jedan uslov ispunjen jer je to nemoguce.

Obuhvat uslova :

1. Metoda **Calculate** : **Klijent.cs**
    - Kod planiranja obuhvatnosti uslova metode **Calculate**. Ona je vec ispunjena testovima obuhvatnosti grana, uzimajuci u obzir

da nam zadnji test nije bio potreban za 100% - tnu obuhvatnost uslova.

- I za ovu obuhvatnost je kljucna bila promijena vrijednosti ulaznog parametra **type**, s time da nismo morali imati dio u kojem vrijednost **type** parametra ne zadovoljava niti jedan uslov iz metode **Calculate**.

2. Metoda **VratiVozilo** : **AutoRentShop.cs**
    - Za 100% - tnu obuhvatnost uslova ove metode odgovaraju testovi kreirani za obuhvatnost grana.

Obuhvat petlji:

1. **Metoda Calculate : Klijent.cs**

- Ukoliko posmatramo metodu Calculate iz željene klase, možemo primjetiti da se unutar nje ne nalazi niti jedna petlja, samim tim obuhvat petlji nema smisla ispitivati. Obuhvatnost petlji je 100% (odnosno 0% - zavisno od pogleda).

2. **Metoda VratiVozilo : AutoRentShop.cs**

- Možemo primjetiti da se unutar metode VratiVozilo nalaze 2 posebne petlje koje su disjunktne, samim tim za testiranje možemo samo koristiti situacije kad imamo jednostavne petlje sa n prolaza.

- Da bi dobili 100% obuhvatnost petlji testirati ćemo sljedeće slučajeve:
    - Testiranje petlje, da li uslov petlje ispunjen,
    - Prolazak kroz petlju sa ispunjenim uslovom, koji predstavlja i < velicina_liste,
    - Jedan prolazak kroz petlju,
    - Dva prolaska kroz petlju,
    - m prolazaka kroz petlju gdje m < n, a n je veličina liste. Ovdje ćemo testirati zapravo da li je zadovoljen uslov break-anja,

- (n-1) i (n) prolazaka kroz petlju sa zadovoljenim uslovom grananja na n-1 odnosno n poziciji petlje.

Na ovaj način smo prešli sve moguće slučajeve unutar petlje i zadovoljili obuhvatnost. Ukupan broj petlji je 7, odnosno 14 jer postoje 2 petlje. Slijedi prikaz testova:

Test1 testira ne-ispunjene uslove obje petlje, ovdje se petlje ne bi trebale izvršiti jer ne postoje iznajmljena vozila. Vidimo da će se baciti exception ukoliko pokušamo pristupiti praznoj listi shodno time je test1 ok. Ovaj test nam govori da bi uopšte pristupili petljama mora korisnik imati iznajmljeno vozilo.

```
static void Main(string[] args)
{
    AutoRentShop shop = new AutoRentShop();
    Klijent klijent = new Klijent("Haris", "Mašović", DateTime.Now);
    Klijent klijent2 = new Klijent("Haris1", "Mašović1", DateTime.Now);

    /*
    shop.RegistrujNovoVozilo(new TeretnoVozilo("nesto", "nesto2", "1A2B3C4567G111500", 16));
    shop.IznajmiAuto(klijent, "1A2B3C4567G111500", DateTime.Now);
    Console.WriteLine(klijent.SadrzajListeObavijesti());

    shop.IznajmiAuto(klijent2, "1A2B3C4567G111500", DateTime.Now);
    Console.WriteLine(klijent2.SadrzajListeObavijesti());
    */
    try
    {
        double rez = shop.VratiVozilo(klijent, "1A2B3C4567G111500");
        Console.WriteLine("Cijena korištenja: " + rez.ToString());
        Console.WriteLine(klijent2.SadrzajListeObavijesti());
    }
    catch(Exception e)
    {                                       // Test1: ne postoji klijent sa iznajmljenim vozilom, OK
        Console.WriteLine(e.ToString());
    }
    Console.ReadLine();


}
```

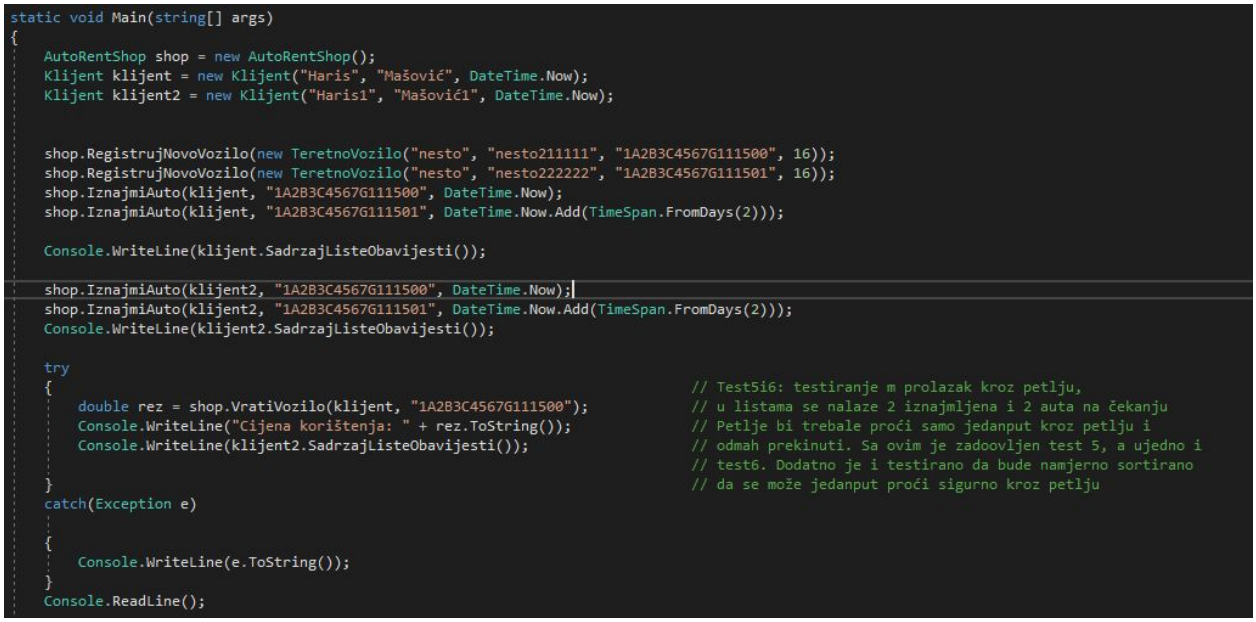
Test2i3 testira ispunjene obuhvatnosti petlji sa ispunjenim uslovom,odnosno testira prolazak kroz petlju sa ispunjenim uslovom i jedan prolazak kroz petlju.

```
static void Main(string[] args)
{
    AutoRentShop shop = new AutoRentShop();
    Klijent klijent = new Klijent("Haris", "Mašović", DateTime.Now);
    Klijent klijent2 = new Klijent("Haris1", "Mašović1", DateTime.Now);


    shop.RegistrujNovoVozilo(new TeretnoVozilo("nesto", "nesto2", "1A2B3C4567G111500", 16));
    shop.IznajmiAuto(klijent, "1A2B3C4567G111500", DateTime.Now);
    Console.WriteLine(klijent.SadrzajListeObavijesti());

    shop.IznajmiAuto(klijent2, "1A2B3C4567G111500", DateTime.Now);
    Console.WriteLine(klijent2.SadrzajListeObavijesti());

    try
    {                                                                   // Test2i3: Testiranje prolazka kroz petlje, uz zadovoljen uslov petlji
        double rez = shop.VratiVozilo(klijent, "1A2B3C4567G111500");    // klijet vraca vozilo, prolazak je omogucen kroz prvu petlju
        Console.WriteLine("Cijena korištenja: " + rez.ToString());      // Ispisuje se cijena koju mora platiti
        Console.WriteLine(klijent2.SadrzajListeObavijesti());           // Ispisuje se poruka korisniku, da je vozilo vraceno
    }                                                                   // Cime je test2 zadoovljen za 2 petlju
    catch(Exception e)                                                  // Test2 također testira i jedan prolazak kroz petlje, samim tim je i
                                                                        // taj dio ispunjen odnosno Test3
    {
        Console.WriteLine(e.ToString());
    }
    Console.ReadLine();
```

Test4i7 testira 2 prolazka kroz obje petlje. Ovdje i također zadovoljen test broj 7 za obje petlje, jer se prolazi kroz obje petlje n puta odnosno n = 2.

```
static void Main(string[] args)
{
    AutoRentShop shop = new AutoRentShop();
    Klijent klijent = new Klijent("Haris", "Mašović", DateTime.Now);
    Klijent klijent2 = new Klijent("Haris1", "Mašović1", DateTime.Now);


    shop.RegistrujNovoVozilo(new TeretnoVozilo("nesto", "nesto211111", "1A2B3C4567G111500", 16));
    shop.RegistrujNovoVozilo(new TeretnoVozilo("nesto", "nesto222222", "1A2B3C4567G111501", 16));
    shop.IznajmiAuto(klijent, "1A2B3C4567G111500", DateTime.Now);
    shop.IznajmiAuto(klijent, "1A2B3C4567G111501", DateTime.Now.Add(TimeSpan.FromDays(2)));

    Console.WriteLine(klijent.SadrzajListeObavijesti());

    shop.IznajmiAuto(klijent2, "1A2B3C4567G111500", DateTime.Now);
    shop.IznajmiAuto(klijent2, "1A2B3C4567G111501", DateTime.Now.Add(TimeSpan.FromDays(2)));
    Console.WriteLine(klijent2.SadrzajListeObavijesti());

    try
    {                                                                   // Test4i7: testiranje 2 prolaska kroz petlju,
        double rez = shop.VratiVozilo(klijent, "1A2B3C4567G111501");    // u listama se nalaze 2 iznajmljena i 2 auta na čekanju
        Console.WriteLine("Cijena korištenja: " + rez.ToString());      // Petlje bi trebale proći dva puta kroz petlju i
        Console.WriteLine(klijent2.SadrzajListeObavijesti());           // odmah prekinuti. Sa ovim je zadoovljen test 4, a ujedno i
                                                                        // test7. Dodatno je i testirano da bude namjerno sortirano
    }                                                                   // da se može jedanput proći sigurno kroz petlju
    catch(Exception e)


    {
        Console.WriteLine(e.ToString());
    }
    Console.ReadLine();
```

Test 5i6 testira m prolazaka kroz obje petlje gdje m < n, a n je veličina liste i (n-1) test prolazak, gdje n = 2 u ovom slučaju, a m = 1.



```
static void Main(string[] args)
{
    AutoRentShop shop = new AutoRentShop();
    Klijent klijent = new Klijent("Haris", "Mašović", DateTime.Now);
    Klijent klijent2 = new Klijent("Haris1", "Mašović1", DateTime.Now);


    shop.RegistrujNovoVozilo(new TeretnoVozilo("nesto", "nesto211111", "1A2B3C4567G111500", 16));
    shop.RegistrujNovoVozilo(new TeretnoVozilo("nesto", "nesto222222", "1A2B3C4567G111501", 16));
    shop.IznajmiAuto(klijent, "1A2B3C4567G111500", DateTime.Now);
    shop.IznajmiAuto(klijent, "1A2B3C4567G111501", DateTime.Now.Add(TimeSpan.FromDays(2)));

    Console.WriteLine(klijent.SadrzajListeObavijesti());

    shop.IznajmiAuto(klijent2, "1A2B3C4567G111500", DateTime.Now);
    shop.IznajmiAuto(klijent2, "1A2B3C4567G111501", DateTime.Now.Add(TimeSpan.FromDays(2)));
    Console.WriteLine(klijent2.SadrzajListeObavijesti());

    try
    {                                                                   // Test5i6: testiranje m prolazak kroz petlju,
        double rez = shop.VratiVozilo(klijent, "1A2B3C4567G111500");    // u listama se nalaze 2 iznajmljena i 2 auta na čekanju
        Console.WriteLine("Cijena korištenja: " + rez.ToString());      // Petlje bi trebale proći samo jedanput kroz petlju i
        Console.WriteLine(klijent2.SadrzajListeObavijesti());           // odmah prekinuti. Sa ovim je zadoovljen test 5, a ujedno i
                                                                        // test6. Dodatno je i testirano da bude namjerno sortirano
    }                                                                   // da se može jedanput proći sigurno kroz petlju
    catch(Exception e)

    {
        Console.WriteLine(e.ToString());
    }
    Console.ReadLine();
```

Ovim testovima smo napravili punu obuhvat petlji za obje petlje, odnosno 14 slučajeva uradili.