Univerzitet u Sarajevu
Elektrotehnički fakultet
Sarajevo

Projekat - drugi dio: White box testiranje

Pešković Dinela, 19359

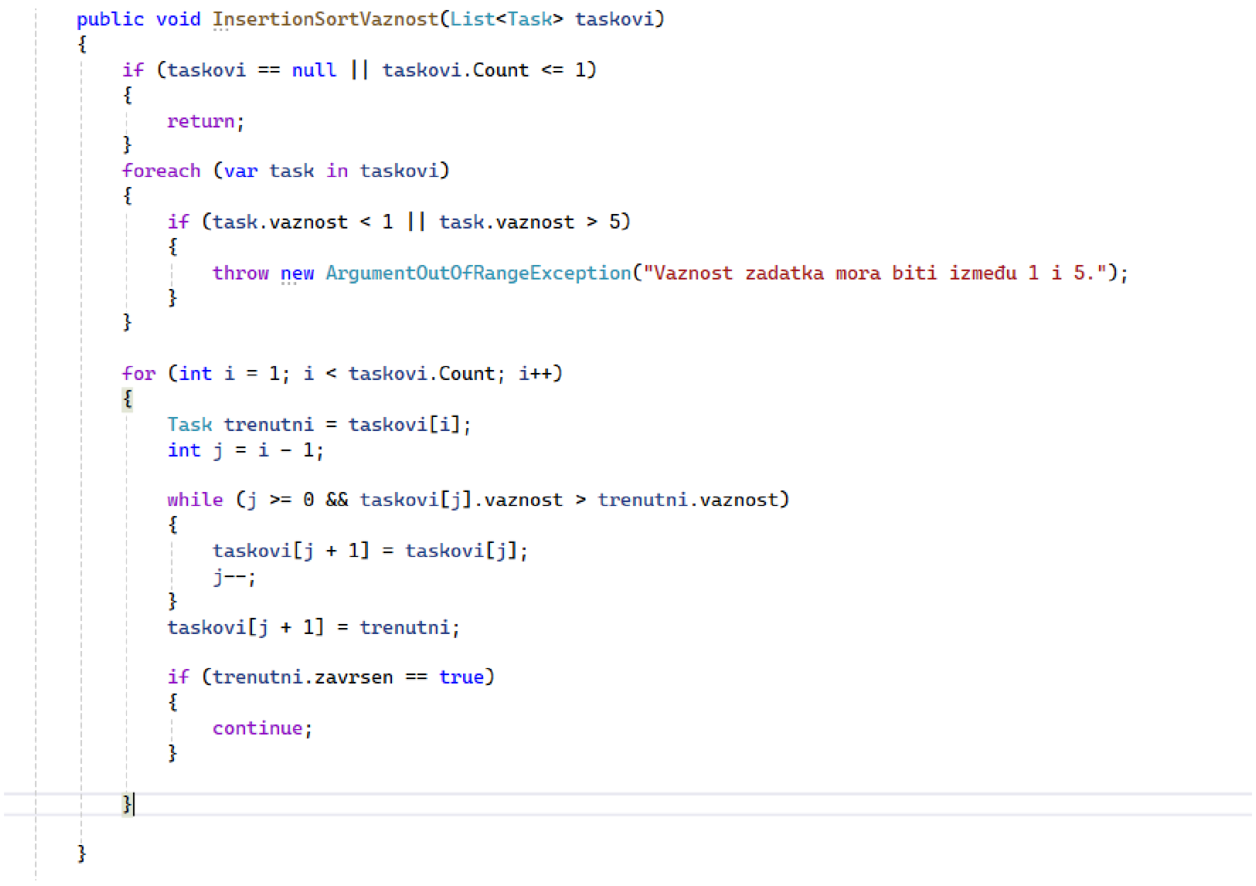
26.12.2024. god.

Postavka zadatka:

Potrebno je odabrati jednu metodu iz konzolne aplikacije napisane za 1. dio projekta koja ima što veći indeks održavanja (npr. sadrži if iskaze, petlje, implementirane algoritme). Za tu metodu je potrebno izvršiti sljedeće:

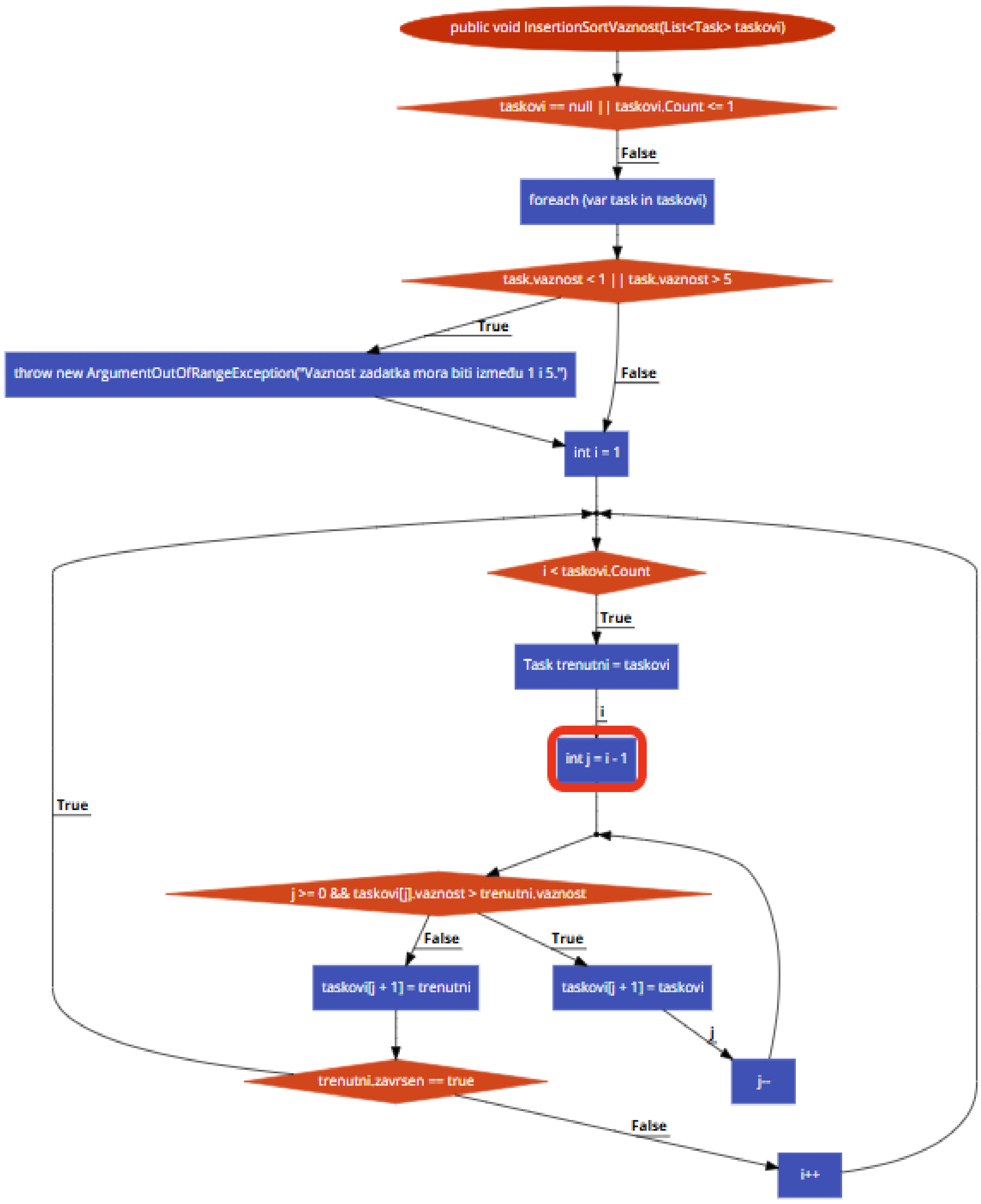
1. Izračunati McCabe metriku i nacrtati kontrolni graf. Uporediti rezultate sa rezultatima iz Code Metrics alata i prodiskutovati kvalitet koda i dobijene rezultate.
2. Uraditi white box testiranje nad metodom koristeći primjenjiva 3 različita primjenjiva pristupa white box[1]. Prikazati odabrane metode, dizajnirane testne slučajeve i provedeni postupak testiranja.
3. Izvršiti code tuning[2] za odabranu metodu primjenom minimalno 3 različite tehnike. Navesti koje su tehnike korištene i prikazati novi kod. Nakon svake urađene tehnike zabilježiti vrijeme izvršavanja i alociranu memoriju sa velikim brojem pokretanja. Zabilježiti i prodiskutovati i McCabe metriku i indeks održavanja te metode prije i poslije code tuning-a.
4. Nakon code tuning-a, izvršiti refaktoring[3] tako da se postignu što bolje vrijednosti metrika. Prikazati kojoj stavki unutar checklist-e za refaktoring svaka promjena pripada, i kako ona utječe na metrike u sklopu Code Metrics.

Metoda: InsertionSortVaznost(List<Task> taskovi)

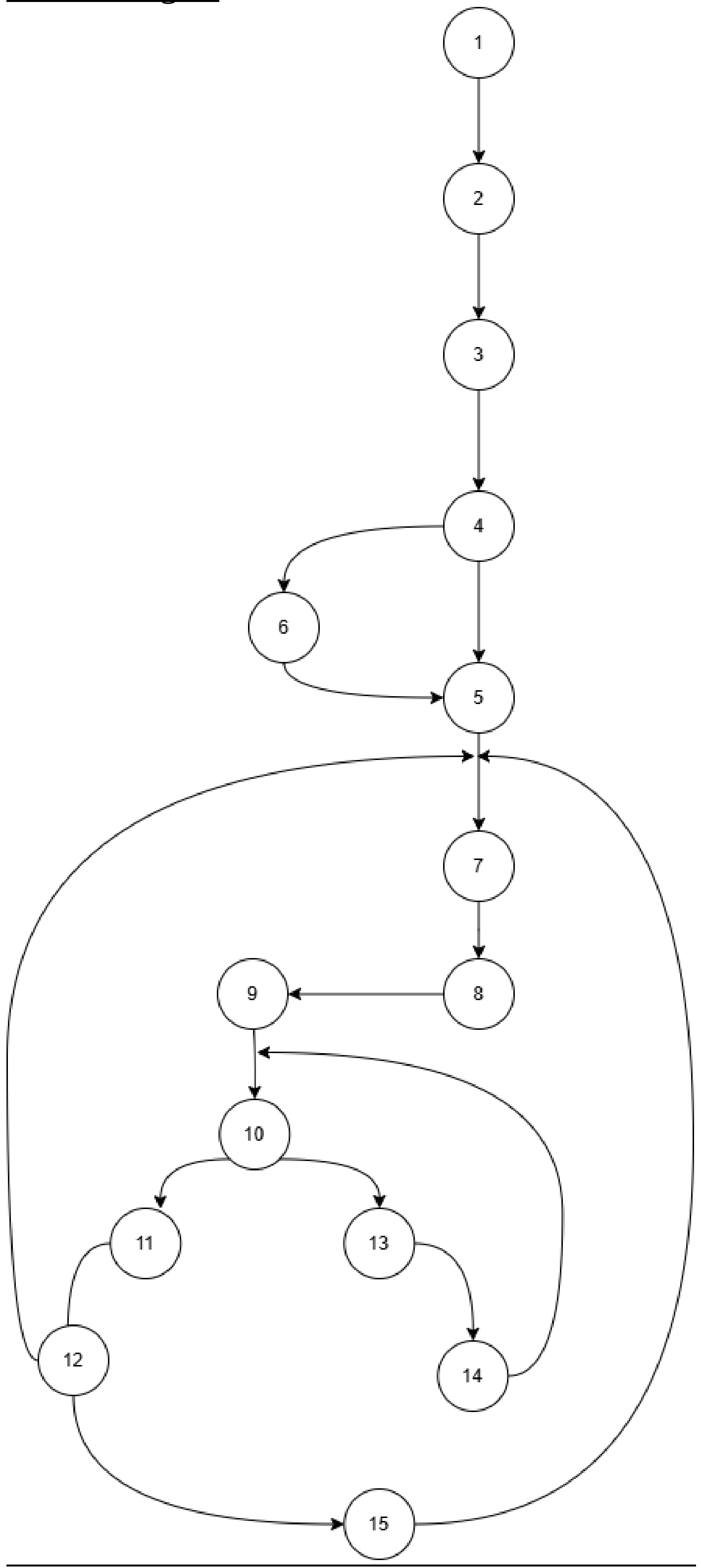


```
public void InsertionSortVaznost(List<Task> taskovi)
{
    if (taskovi == null || taskovi.Count <= 1)
    {
        return;
    }
    foreach (var task in taskovi)
    {
        if (task.vaznost < 1 || task.vaznost > 5)
        {
            throw new ArgumentOutOfRangeException("Vaznost zadatka mora biti između 1 i 5.");
        }
    }

    for (int i = 1; i < taskovi.Count; i++)
    {
        Task trenutni = taskovi[i];
        int j = i - 1;

        while (j >= 0 && taskovi[j].vaznost > trenutni.vaznost)
        {
            taskovi[j + 1] = taskovi[j];
            j--;
        }
        taskovi[j + 1] = trenutni;

        if (trenutni.zavrsen == true)
        {
            continue;
        }

    }

}
```

1. <u>Dijagram toka:</u>


public void InsertionSortVaznost(List<Task> taskovi)
taskovi == null || taskovi.Count <= 1
False
foreach (var task in taskovi)
task.vaznost < 1 || task.vaznost > 5
True
throw new ArgumentOutOfRangeException("Vaznost zadatka mora biti između 1 i 5.")
False
int i = 1
i < taskovi.Count
True
Task trenutni = taskovi
i
int j = i - 1
True
j >= 0 && taskovi[j].vaznost > trenutni.vaznost
False
True
taskovi[j + 1] = trenutni
taskovi[j + 1] = taskovi
j
j--
trenutni.zavrsen == true
False
i++

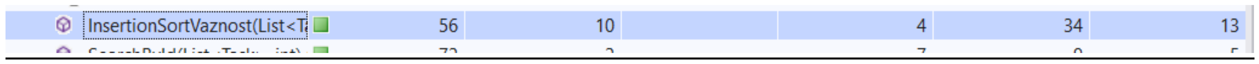
2. Kontrolni graf



1. Računanje McCabe metrike:

e=18
n=15
p=1
M = e - n + 2*p=18-15+2=5

Rezultati iz Code Metrics alata:



InsertionSortVaznost(List<T	🟩	56	10		4	34	13

Vidimo da je ciklomatska kompleksnost jednaka 10, a proračunom za McCabe metriku dobijamo vrijednost 5. Iako bi se trebale dobiti iste vrijednosti, razlog zbog kojeg se razlikuju je način na koji Visual Studio analizira izraze u kodu. U if iskazu koji ima operatore u sebi poput || i && Visual Studio dodatno računa složenost tih operatora, pa samim tim se povećava broj grana. Ukoliko bi na kontrolnom grafu dodala izlaz za svaki uslov vezan operatorima, te još jedan dodatni čvor koji bi označavao izlaz za drugi čvor na osnovu njega McCabe metrika bi se računala kao: 24-16+2=8+2=10, čime bih dobila isti rezultat kao u Visual Studio. Vrijednost izračunate ciklomatske kompleksnosti je manja od 10, što ovaj kod čini veoma jednostavnim za testiranje i razumijevanje. Također, u rezultatima uočavamo da je indeks održavanja 56, što se smatra održivim, mada bi se mogao uraditi potencijalni refaktoring da bi se vrijednost poboljšala. Na dobru održivost koda nam ukazuje i to što je Class Coupling nizak.

2. White-box testiranje:

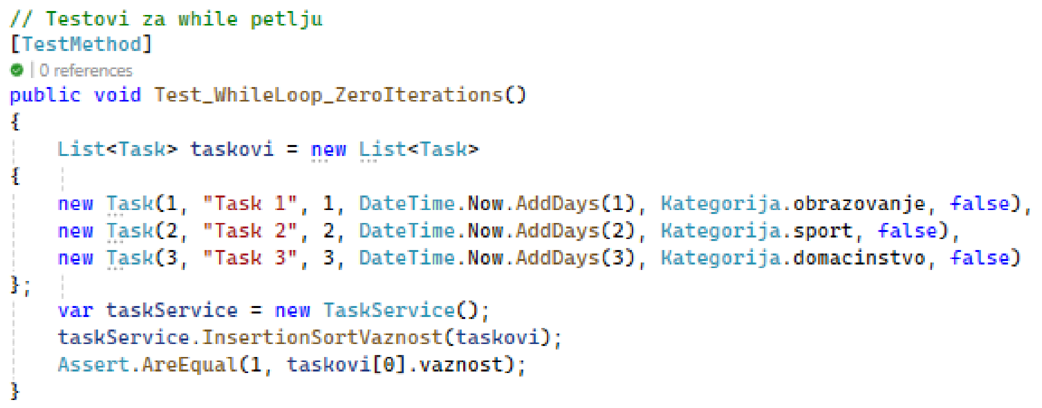
Potpuni obuhvat puteva:

1.	1-2-3-4-5-7-8-9-10-11-12-15
2.	1-2-3-4-6-5-7-8-9-10-11-12-15
3.	1-2-3-4-5-7-8-9-10-13-14-10-11-12-15
4.	1-2-3-4-6-5-7-8-9-10-13-14-10-11-12-15
5.	1-2-3-4-5-7-8-9-10-13-14-10-11-12-7-8-9-10-11-12-15
6.	1-2-3-4-6-5-7-8-9-10-13-14-10-11-12-7-8-9-10-11-12-15
7.	1-2-3-4-5-7-8-9-10-11-12-7-8-9-10-11-12-15
8.	1-2-3-4-6-5-7-8-9-10-11-12-7-8-9-10-11-12-15
9.	1-2-3-4-5-7-8-9-10-11-12-15-7-8-9-10-11-12-15
10.	1-2-3-4-5-7-8-9-10-11-12-15-7-8-9-10-13-14-10-11-12-15
11.	1-2-3-4-6-5-7-8-9-10-11-12-15-7-8-9-10-11-12-15
12.	1-2-3-4-6-5-7-8-9-10-11-12-15-7-8-9-10-13-14-10-11-12-15

13.	1-2-3-4-5-7-8-9-10-13-14-10-11-12-7-8-9-10-11-12-15-7-8-9-10-11-12-15
14.	1-2-3-4-6-5-7-8-9-10-13-14-10-11-12-7-8-9-10-11-12-15-7-8-9-10-11-12-15
15.	1-2-3-4-5-7-8-9-10-11-12-7-8-9-10-11-12-15-7-8-9-10-11-12-15
16.	1-2-3-4-6-5-7-8-9-10-11-12-7-8-9-10-11-12-15-7-8-9-10-11-12-15

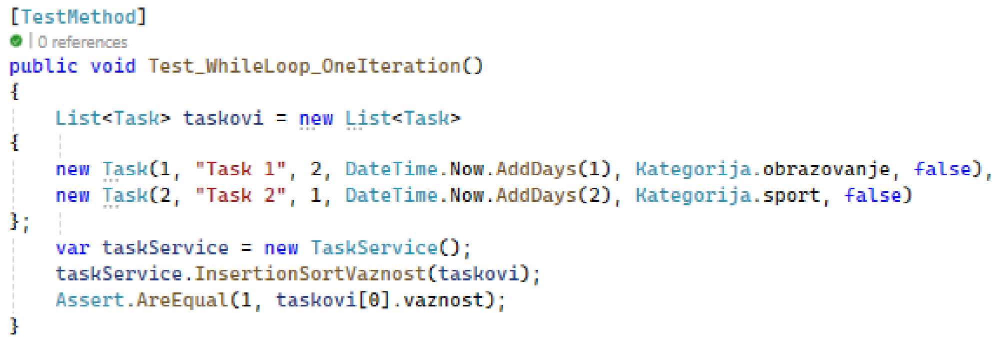
Obuhvat petlji:
Unutrašnja petlja:

1.Preskočiti unutrašnjost petlje



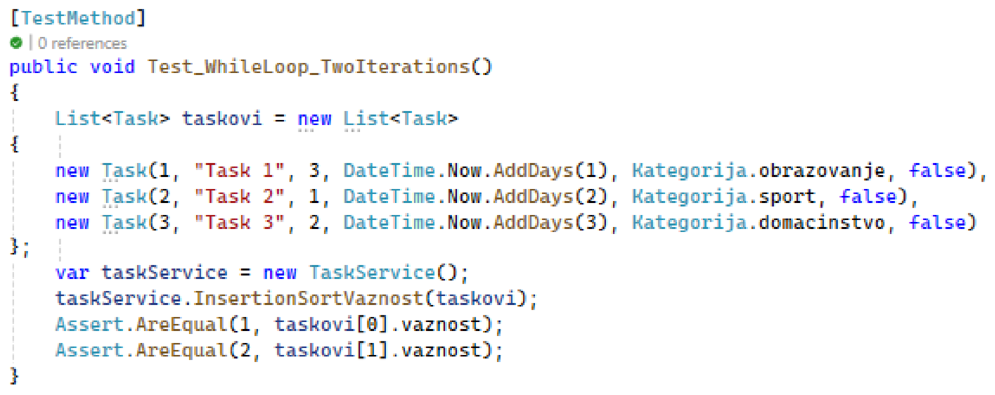
```
// Testovi za while petlju
[TestMethod]
● | 0 references
public void Test_WhileLoop_ZeroIterations()
{
    List<Task> taskovi = new List<Task>
{
    new Task(1, "Task 1", 1, DateTime.Now.AddDays(1), Kategorija.obrazovanje, false),
    new Task(2, "Task 2", 2, DateTime.Now.AddDays(2), Kategorija.sport, false),
    new Task(3, "Task 3", 3, DateTime.Now.AddDays(3), Kategorija.domacinstvo, false)
};
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.AreEqual(1, taskovi[0].vaznost);
}
```

2. Jedan prolazak kroz petlju



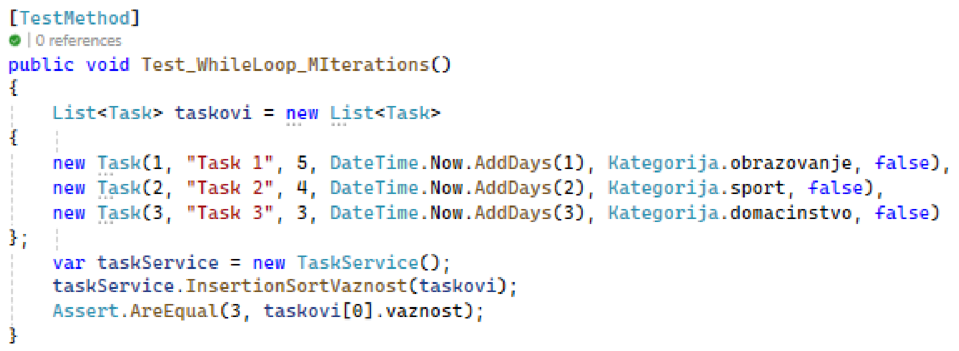
```
[TestMethod]
● | 0 references
public void Test_WhileLoop_OneIteration()
{
    List<Task> taskovi = new List<Task>
{
    new Task(1, "Task 1", 2, DateTime.Now.AddDays(1), Kategorija.obrazovanje, false),
    new Task(2, "Task 2", 1, DateTime.Now.AddDays(2), Kategorija.sport, false)
};
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.AreEqual(1, taskovi[0].vaznost);
}
```

3. Dva prolaska kroz petlju



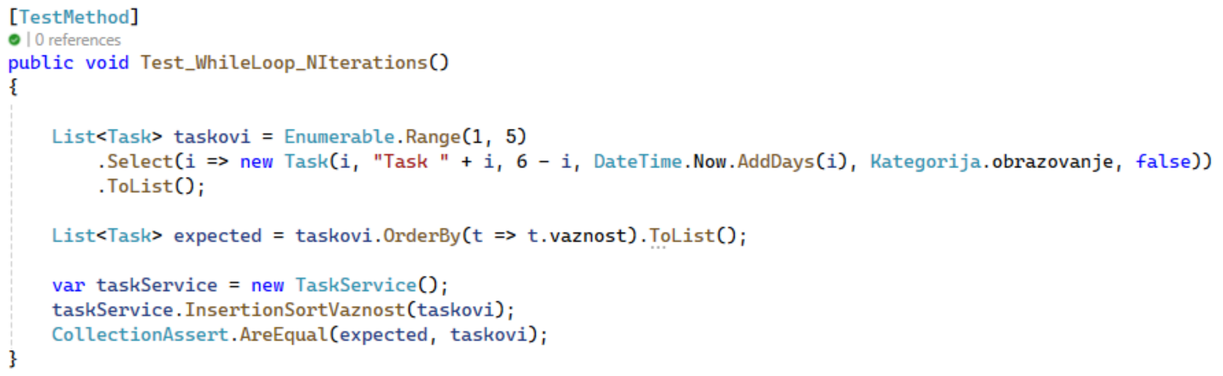
```
[TestMethod]
● | 0 references
public void Test_WhileLoop_TwoIterations()
{
    List<Task> taskovi = new List<Task>
{
    new Task(1, "Task 1", 3, DateTime.Now.AddDays(1), Kategorija.obrazovanje, false),
    new Task(2, "Task 2", 1, DateTime.Now.AddDays(2), Kategorija.sport, false),
    new Task(3, "Task 3", 2, DateTime.Now.AddDays(3), Kategorija.domacinstvo, false)
};
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.AreEqual(1, taskovi[0].vaznost);
    Assert.AreEqual(2, taskovi[1].vaznost);
}
```

4. M prolazaka kroz petlju



```
[TestMethod]
● | 0 references
public void Test_WhileLoop_MIterations()
{
    List<Task> taskovi = new List<Task>
{
    new Task(1, "Task 1", 5, DateTime.Now.AddDays(1), Kategorija.obrazovanje, false),
    new Task(2, "Task 2", 4, DateTime.Now.AddDays(2), Kategorija.sport, false),
    new Task(3, "Task 3", 3, DateTime.Now.AddDays(3), Kategorija.domacinstvo, false)
};
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.AreEqual(3, taskovi[0].vaznost);
}
```

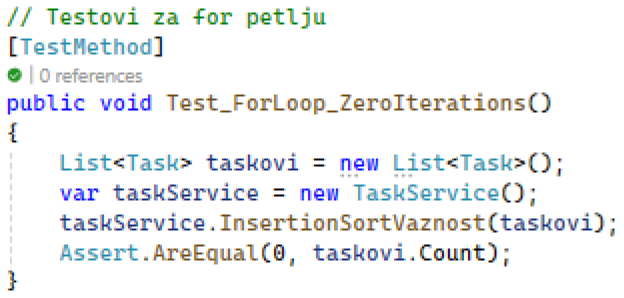
5. N prolazaka kroz petlju



```
[TestMethod]
● | 0 references
public void Test_WhileLoop_NIterations()
{

    List<Task> taskovi = Enumerable.Range(1, 5)
        .Select(i => new Task(i, "Task " + i, 6 - i, DateTime.Now.AddDays(i), Kategorija.obrazovanje, false))
        .ToList();

    List<Task> expected = taskovi.OrderBy(t => t.vaznost).ToList();

    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    CollectionAssert.AreEqual(expected, taskovi);
}
```

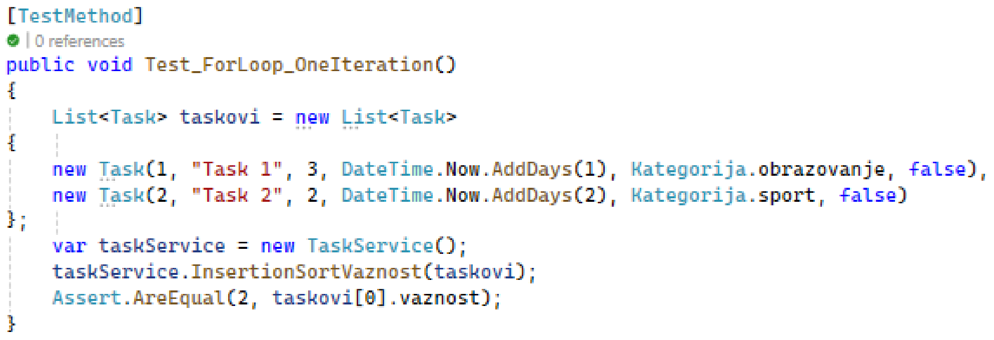
Vanjska petlja

1. Preskočiti unutrašnjost petlje



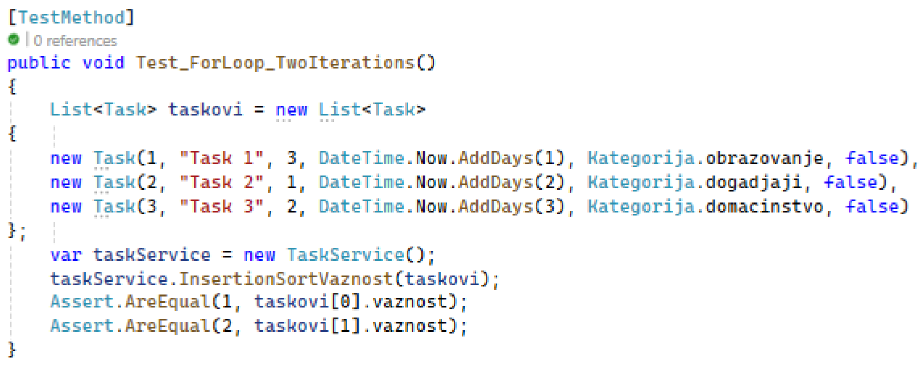
```
// Testovi za for petlju
[TestMethod]
public void Test_ForLoop_ZeroIterations()
{
    List<Task> taskovi = new List<Task>();
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.AreEqual(0, taskovi.Count);
}
```

2. Jedan prolazak kroz petlju



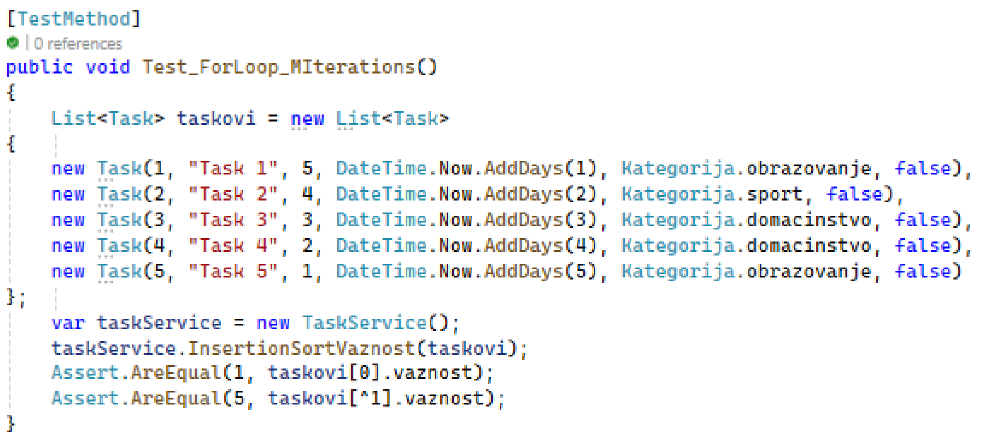
```
[TestMethod]
public void Test_ForLoop_OneIteration()
{
    List<Task> taskovi = new List<Task>
{
    new Task(1, "Task 1", 3, DateTime.Now.AddDays(1), Kategorija.obrazovanje, false),
    new Task(2, "Task 2", 2, DateTime.Now.AddDays(2), Kategorija.sport, false)
};
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.AreEqual(2, taskovi[0].vaznost);
}
```

3. Dva prolaska kroz petlju



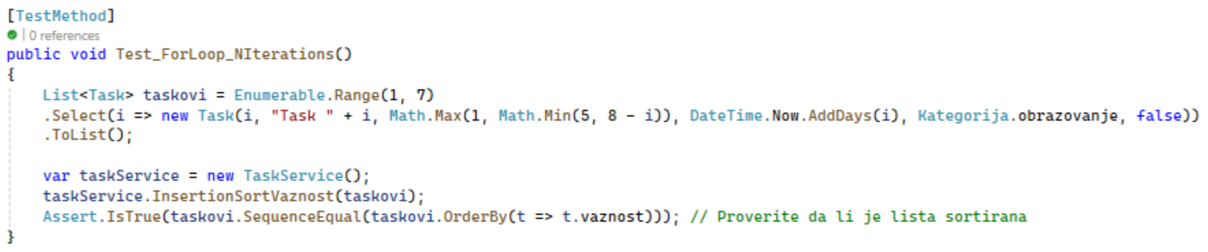
```
[TestMethod]
public void Test_ForLoop_TwoIterations()
{
    List<Task> taskovi = new List<Task>
{
    new Task(1, "Task 1", 3, DateTime.Now.AddDays(1), Kategorija.obrazovanje, false),
    new Task(2, "Task 2", 1, DateTime.Now.AddDays(2), Kategorija.dogadjaji, false),
    new Task(3, "Task 3", 2, DateTime.Now.AddDays(3), Kategorija.domacinstvo, false)
};
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.AreEqual(1, taskovi[0].vaznost);
    Assert.AreEqual(2, taskovi[1].vaznost);
}
```

4. M prolazaka kroz petlju



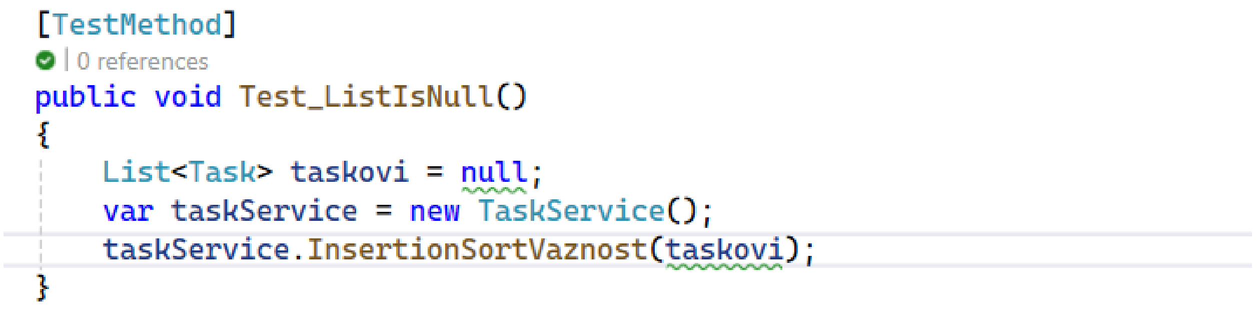
```
[TestMethod]
● | 0 references
public void Test_ForLoop_MIterations()
{
    List<Task> taskovi = new List<Task>
{
    new Task(1, "Task 1", 5, DateTime.Now.AddDays(1), Kategorija.obrazovanje, false),
    new Task(2, "Task 2", 4, DateTime.Now.AddDays(2), Kategorija.sport, false),
    new Task(3, "Task 3", 3, DateTime.Now.AddDays(3), Kategorija.domacinstvo, false),
    new Task(4, "Task 4", 2, DateTime.Now.AddDays(4), Kategorija.domacinstvo, false),
    new Task(5, "Task 5", 1, DateTime.Now.AddDays(5), Kategorija.obrazovanje, false)
};
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.AreEqual(1, taskovi[0].vaznost);
    Assert.AreEqual(5, taskovi[^1].vaznost);
}
```

5. N prolazaka kroz petlju

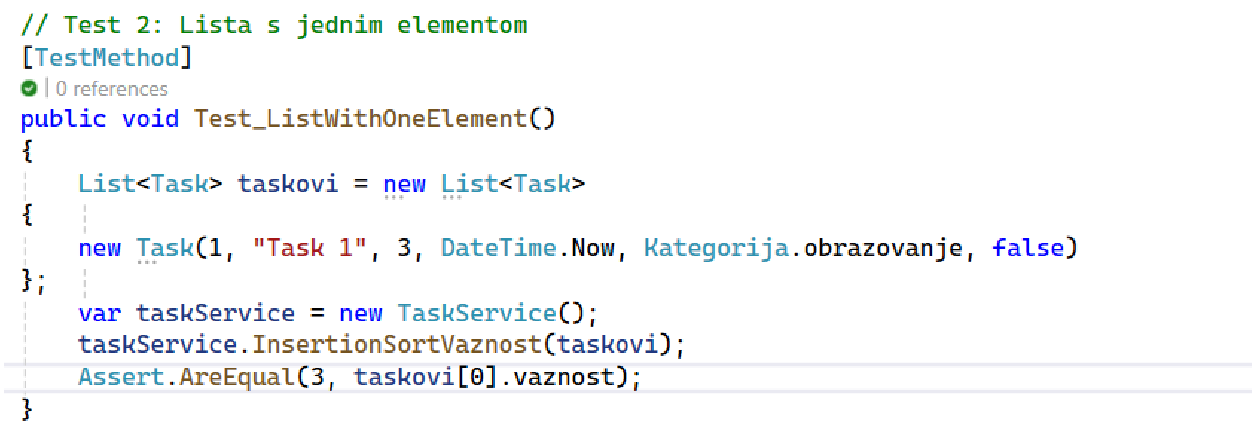


```
[TestMethod]
● | 0 references
public void Test_ForLoop_NIterations()
{
    List<Task> taskovi = Enumerable.Range(1, 7)
    .Select(i => new Task(i, "Task " + i, Math.Max(1, Math.Min(5, 8 - i)), DateTime.Now.AddDays(i), Kategorija.obrazovanje, false))
    .ToList();

    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.IsTrue(taskovi.SequenceEqual(taskovi.OrderBy(t => t.vaznost))); // Proverite da li je lista sortirana
}
```

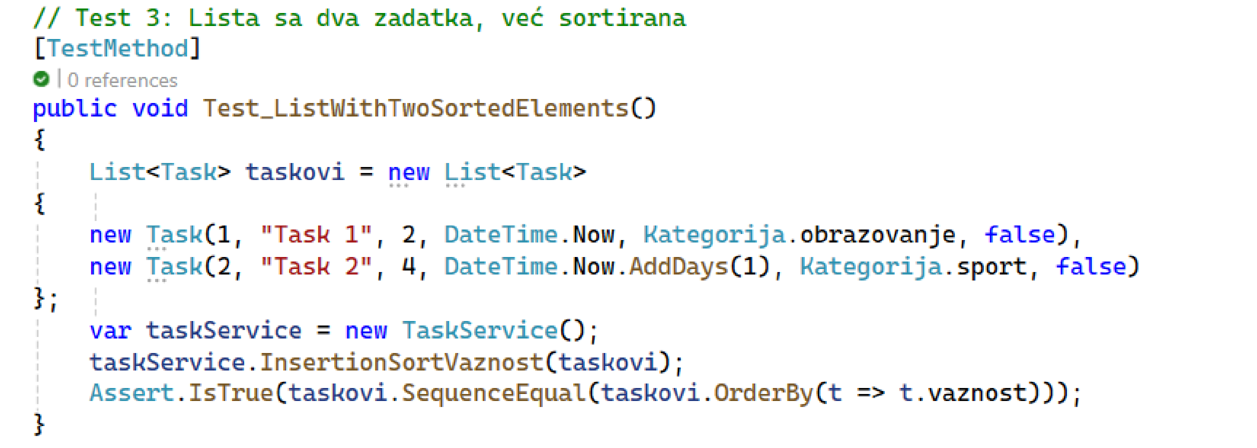
Obuhvat linija/iskaza:



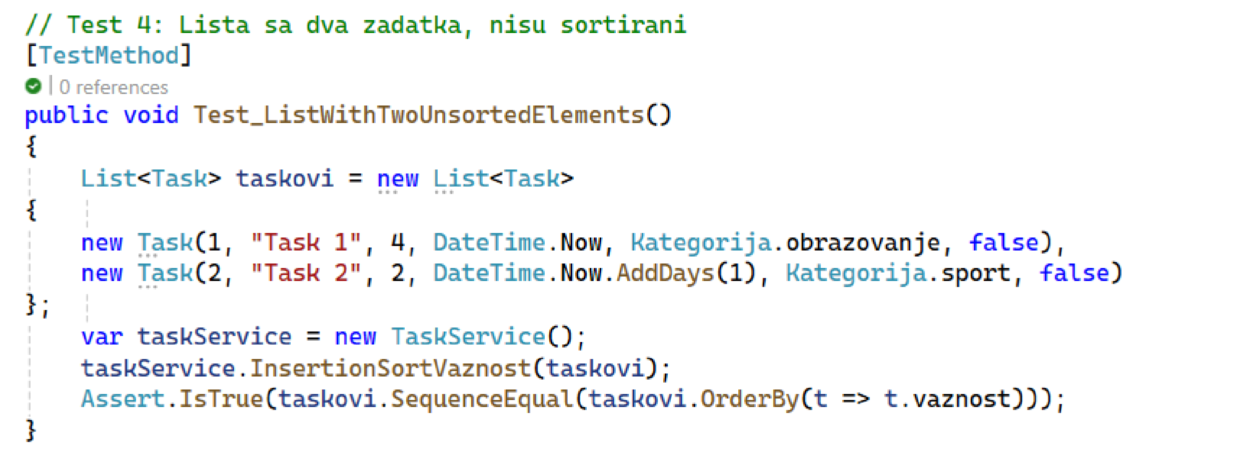
```
[TestMethod]
| 0 references
public void Test_ListIsNull()
{
    List<Task> taskovi = null;
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
}
```



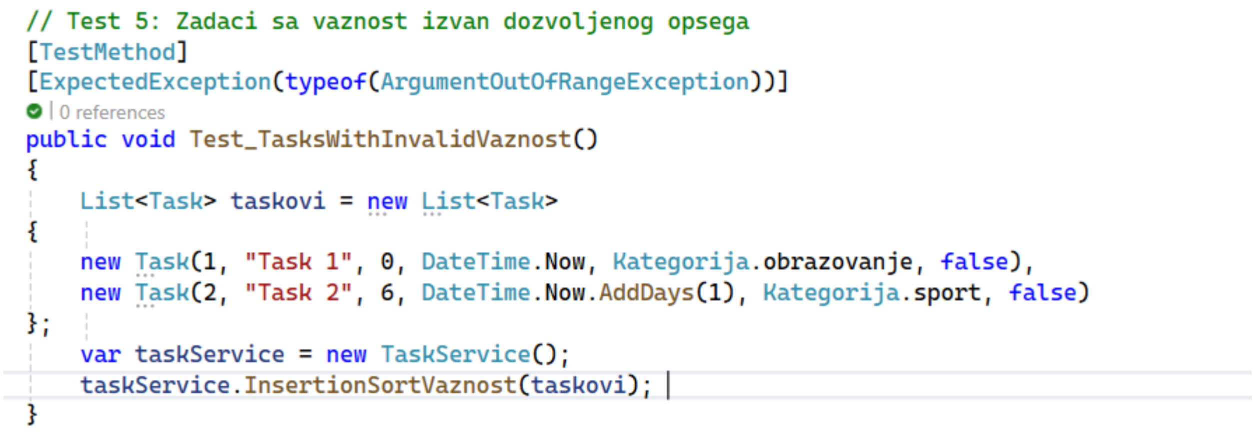
```
// Test 2: Lista s jednim elementom
[TestMethod]
| 0 references
public void Test_ListWithOneElement()
{
    List<Task> taskovi = new List<Task>
{
    new Task(1, "Task 1", 3, DateTime.Now, Kategorija.obrazovanje, false)
};
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.AreEqual(3, taskovi[0].vaznost);
}
```



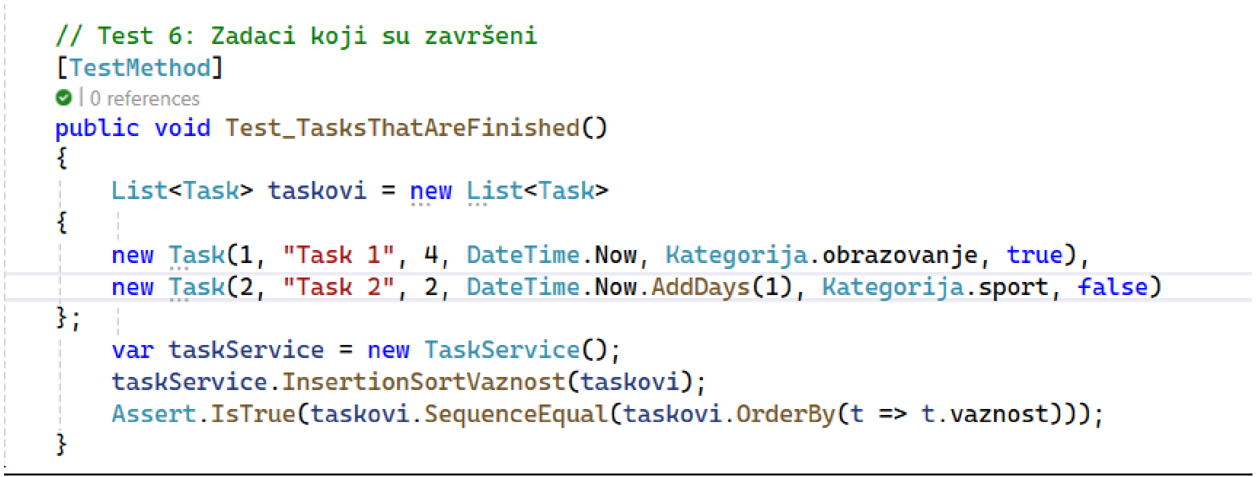
```
// Test 3: Lista sa dva zadatka, već sortirana
[TestMethod]
| 0 references
public void Test_ListWithTwoSortedElements()
{
    List<Task> taskovi = new List<Task>
{
    new Task(1, "Task 1", 2, DateTime.Now, Kategorija.obrazovanje, false),
    new Task(2, "Task 2", 4, DateTime.Now.AddDays(1), Kategorija.sport, false)
};
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.IsTrue(taskovi.SequenceEqual(taskovi.OrderBy(t => t.vaznost)));
}
```



```
// Test 4: Lista sa dva zadatka, nisu sortirani
[TestMethod]
| 0 references
public void Test_ListWithTwoUnsortedElements()
{
    List<Task> taskovi = new List<Task>
{
    new Task(1, "Task 1", 4, DateTime.Now, Kategorija.obrazovanje, false),
    new Task(2, "Task 2", 2, DateTime.Now.AddDays(1), Kategorija.sport, false)
};
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.IsTrue(taskovi.SequenceEqual(taskovi.OrderBy(t => t.vaznost)));
}
```

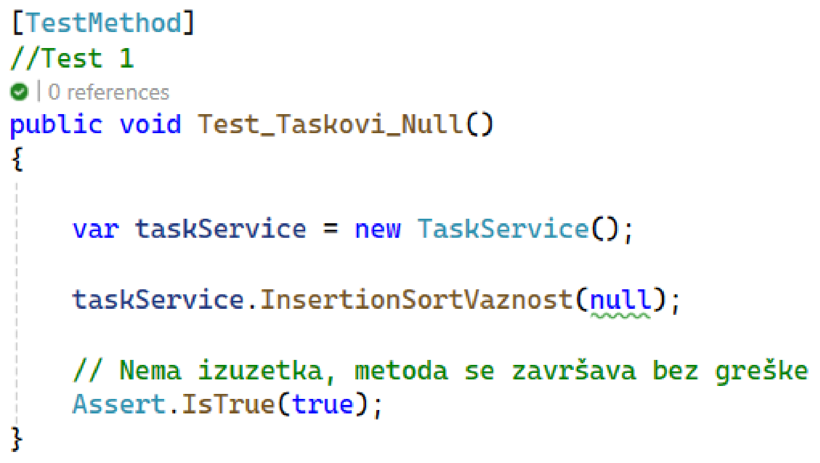


```
// Test 5: Zadaci sa vaznost izvan dozvoljenog opsega
[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))]
| 0 references
public void Test_TasksWithInvalidVaznost()
{
    List<Task> taskovi = new List<Task>
{
    new Task(1, "Task 1", 0, DateTime.Now, Kategorija.obrazovanje, false),
    new Task(2, "Task 2", 6, DateTime.Now.AddDays(1), Kategorija.sport, false)
};
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
}
```

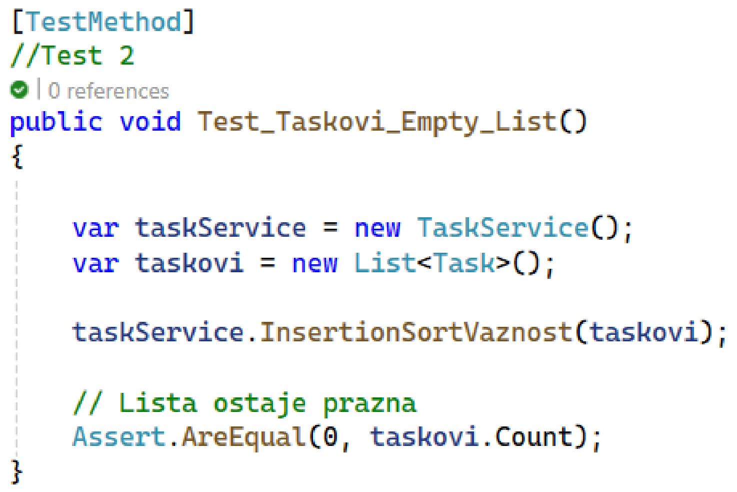


```
// Test 6: Zadaci koji su završeni
[TestMethod]
| 0 references
public void Test_TasksThatAreFinished()
{
    List<Task> taskovi = new List<Task>
{
    new Task(1, "Task 1", 4, DateTime.Now, Kategorija.obrazovanje, true),
    new Task(2, "Task 2", 2, DateTime.Now.AddDays(1), Kategorija.sport, false)
};
    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
    Assert.IsTrue(taskovi.SequenceEqual(taskovi.OrderBy(t => t.vaznost)));
}
```

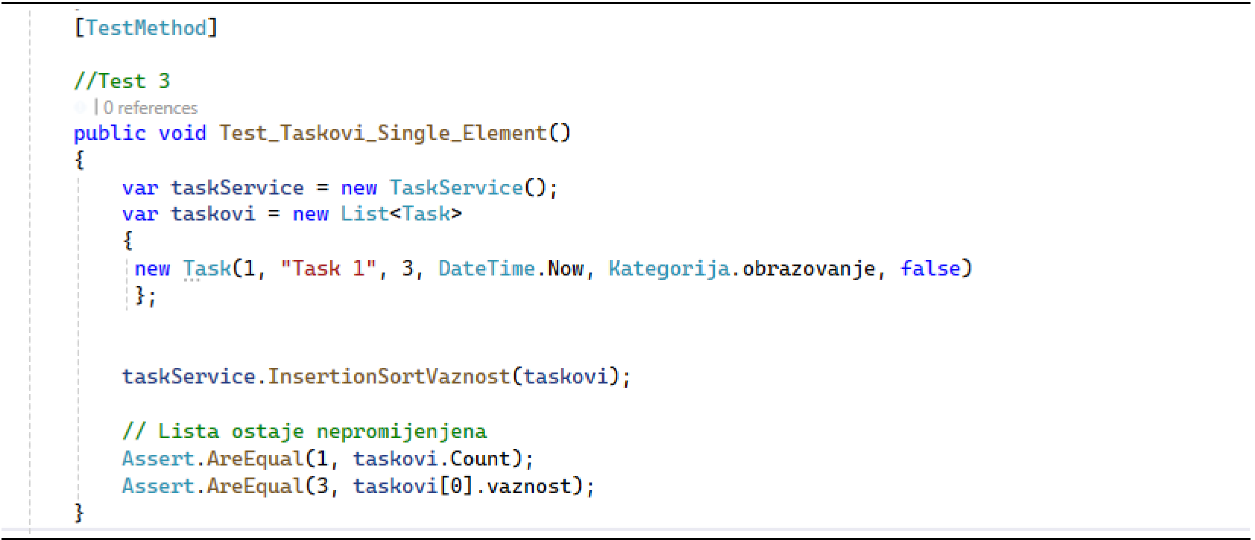
Obuhvat grana/odluka:



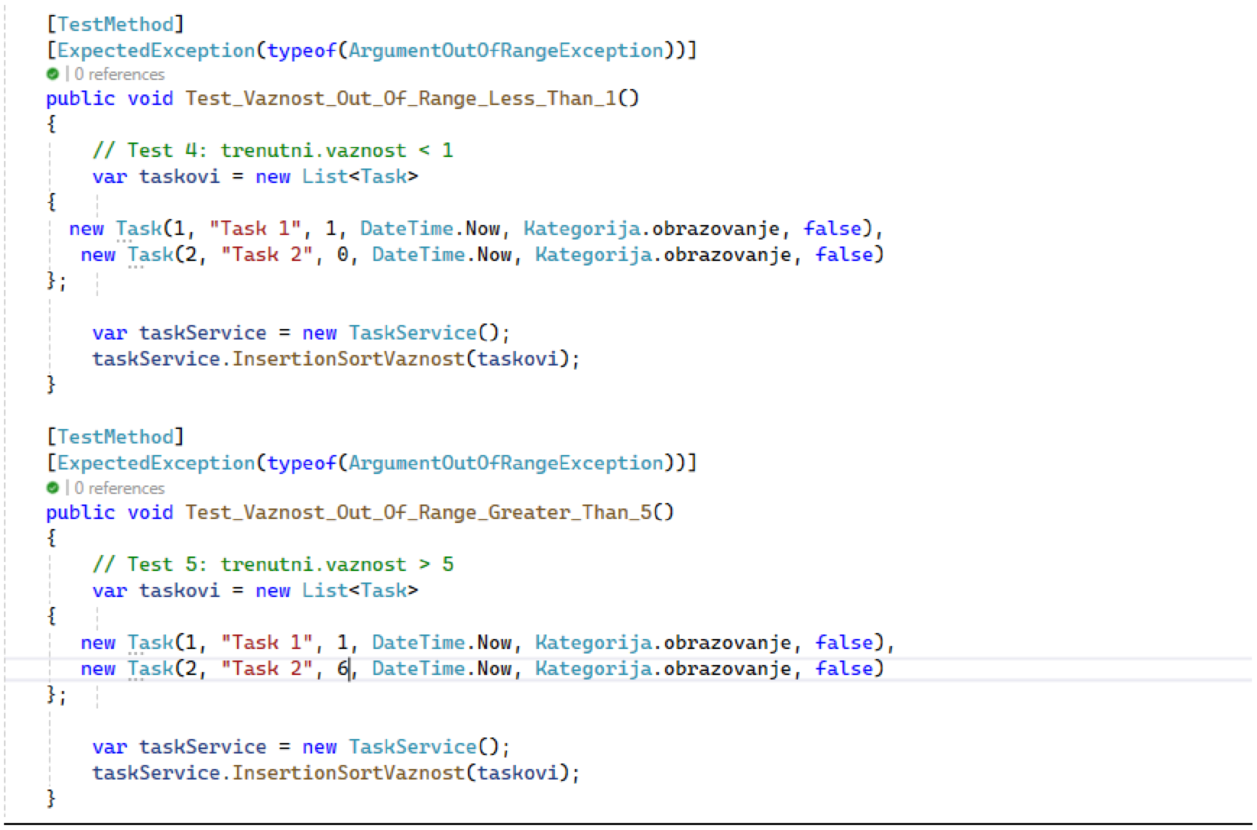
```
[TestMethod]
//Test 1
| 0 references
public void Test_Taskovi_Null()
{

    var taskService = new TaskService();

    taskService.InsertionSortVaznost(null);

    // Nema izuzetka, metoda se završava bez greške
    Assert.IsTrue(true);
}
```



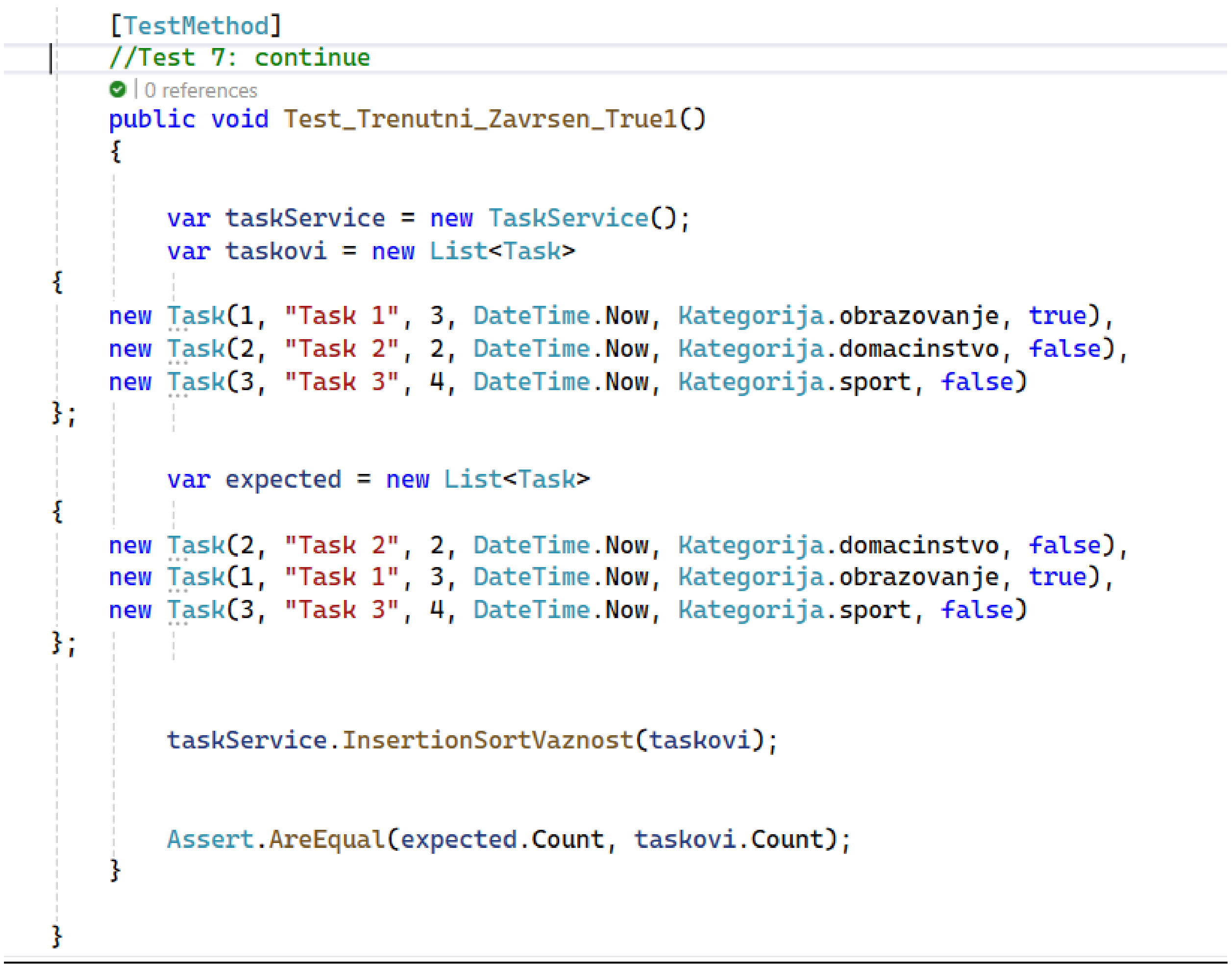
```
[TestMethod]
//Test 2
| 0 references
public void Test_Taskovi_Empty_List()
{

    var taskService = new TaskService();
    var taskovi = new List<Task>();

    taskService.InsertionSortVaznost(taskovi);

    // Lista ostaje prazna
    Assert.AreEqual(0, taskovi.Count);
}
```



```
[TestMethod]

//Test 3
| 0 references
public void Test_Taskovi_Single_Element()
{
    var taskService = new TaskService();
    var taskovi = new List<Task>
    {
     new Task(1, "Task 1", 3, DateTime.Now, Kategorija.obrazovanje, false)
     };


    taskService.InsertionSortVaznost(taskovi);

    // Lista ostaje nepromijenjena
    Assert.AreEqual(1, taskovi.Count);
    Assert.AreEqual(3, taskovi[0].vaznost);
}
```



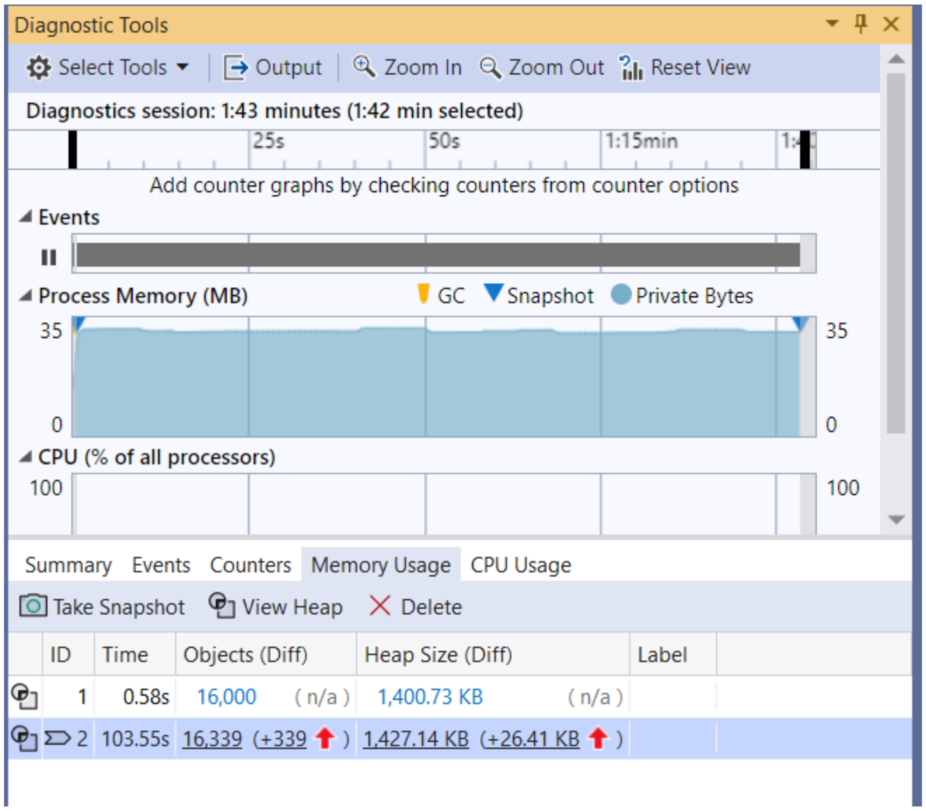
```
[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))]
| 0 references
public void Test_Vaznost_Out_Of_Range_Less_Than_1()
{
    // Test 4: trenutni.vaznost < 1
    var taskovi = new List<Task>
{
  new Task(1, "Task 1", 1, DateTime.Now, Kategorija.obrazovanje, false),
   new Task(2, "Task 2", 0, DateTime.Now, Kategorija.obrazovanje, false)
};

    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
}

[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))]
| 0 references
public void Test_Vaznost_Out_Of_Range_Greater_Than_5()
{
    // Test 5: trenutni.vaznost > 5
    var taskovi = new List<Task>
{
   new Task(1, "Task 1", 1, DateTime.Now, Kategorija.obrazovanje, false),
   new Task(2, "Task 2", 6, DateTime.Now, Kategorija.obrazovanje, false)
};

    var taskService = new TaskService();
    taskService.InsertionSortVaznost(taskovi);
}
```



```
        [TestMethod]
        //Test 7: continue
        | 0 references
        public void Test_Trenutni_Zavrsen_True1()
        {

            var taskService = new TaskService();
            var taskovi = new List<Task>
    {
        new Task(1, "Task 1", 3, DateTime.Now, Kategorija.obrazovanje, true),
        new Task(2, "Task 2", 2, DateTime.Now, Kategorija.domacinstvo, false),
        new Task(3, "Task 3", 4, DateTime.Now, Kategorija.sport, false)
    };

            var expected = new List<Task>
    {
        new Task(2, "Task 2", 2, DateTime.Now, Kategorija.domacinstvo, false),
        new Task(1, "Task 1", 3, DateTime.Now, Kategorija.obrazovanje, true),
        new Task(3, "Task 3", 4, DateTime.Now, Kategorija.sport, false)
    };


            taskService.InsertionSortVaznost(taskovi);


            Assert.AreEqual(expected.Count, taskovi.Count);
        }

    }
```

3. Code Tuning

Prije primjene tehnika code tuninga, vrijeme izvršavanja i potrošnja memorije je sljedeća:



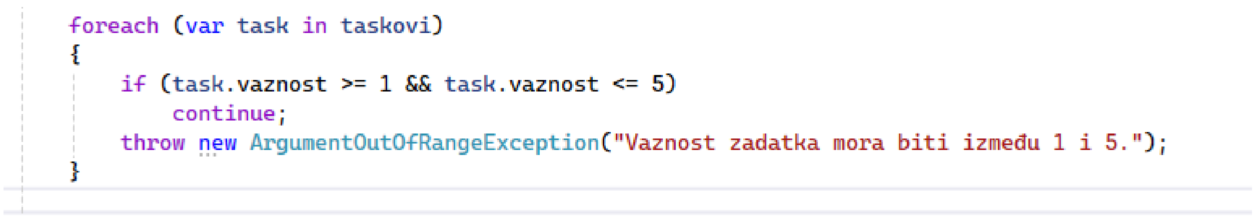
Diagnostics session: 1:43 minutes (1:42 min selected)

ID	Time	Objects (Diff)		Heap Size (Diff)		Label
1	0.58s	16,000	(n/a)	1,400.73 KB	(n/a)	
2	103.55s	16,339	(+339 ↑)	1,427.14 KB	(+26.41 KB ↑)	

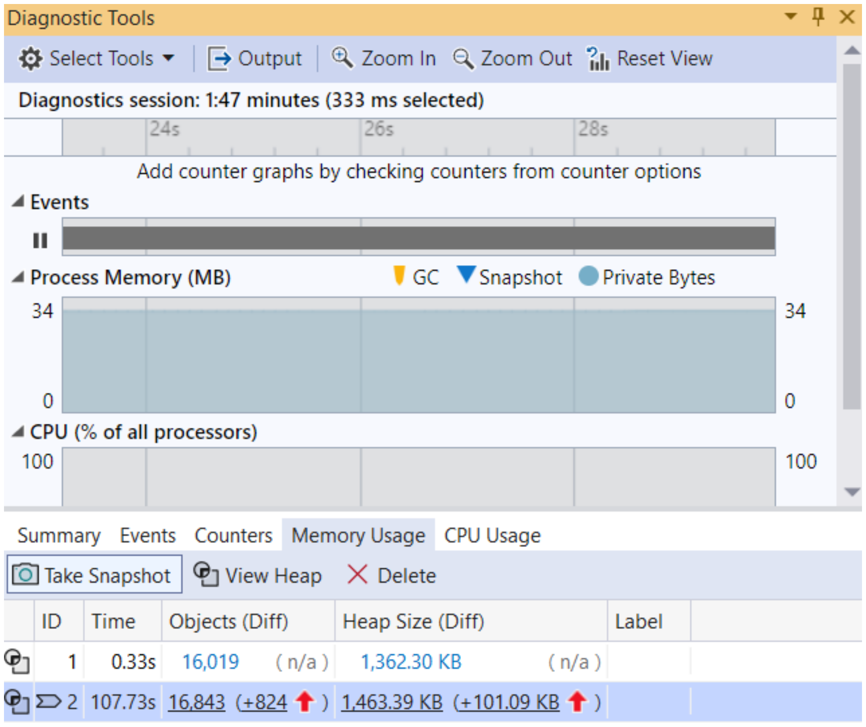
Vrijeme potrošnje: 103.55-0.58=102.97s
Potrošnja memorije:1427.14-1400.73=26.41KB

1. Tuning logičkih izraza: Uređivanje testova po frekvenciji

Kada logički izrazi imaju više provjera, najčešće stanje treba provejriti prvo da bi se smanjio broj izvršenih provjera. Ovo smanjuje ukupno vrijeme izvršavanja koda.
U dijelu gde se provjerava opseg vrijednosti za važnost, ove provjere su optimizovane tako da se prvo provjeri češći slučaj (validne vrijednosti).



```
foreach (var task in taskovi)
{
    if (task.vaznost >= 1 && task.vaznost <= 5)
        continue;
    throw new ArgumentOutOfRangeException("Vaznost zadatka mora biti između 1 i 5.");
}
```

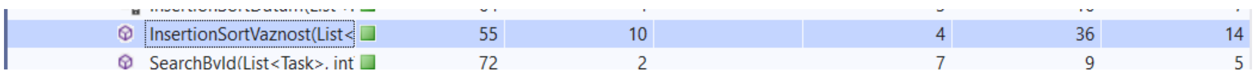


Diagnostics session: 1:47 minutes (333 ms selected)

Add counter graphs by checking counters from counter options

ID	Time	Objects (Diff)	Heap Size (Diff)	Label
1	0.33s	16,019 (n/a)	1,362.30 KB (n/a)	
2	107.73s	16,843 (+824)	1,463.39 KB (+101.09 KB)	

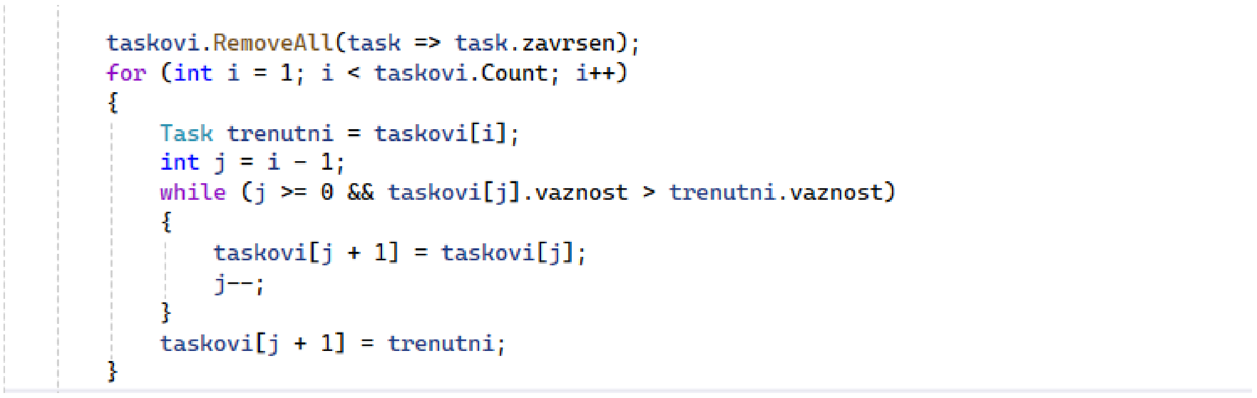
Vrijeme potrošnje: 107.73-0.33=107.4s
Potrošnja memorije:1463.39-1362.30=101.09KB



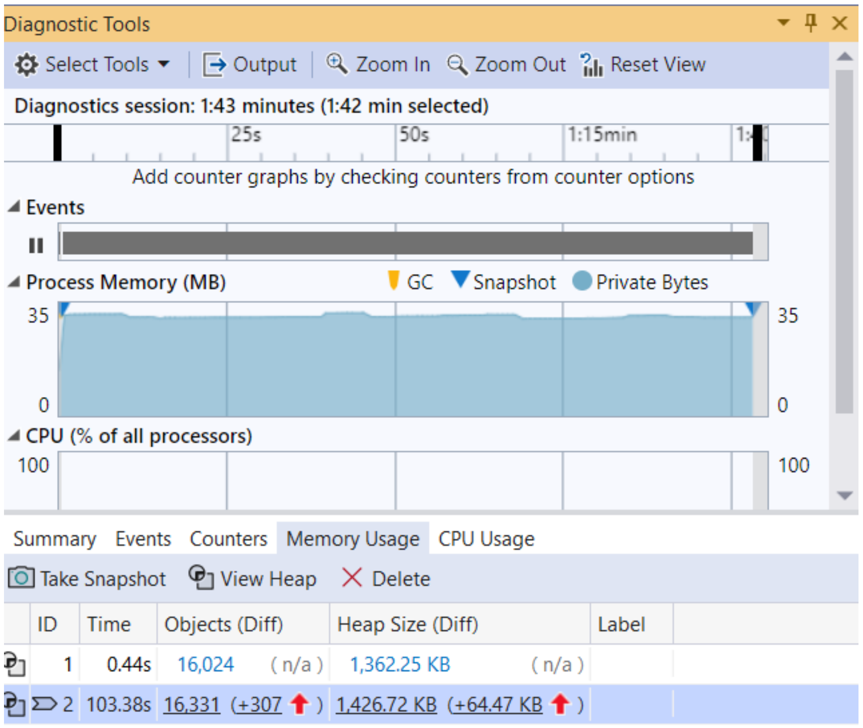
InsertionSortVaznost(List< 🟩	55	10		4	36	14
SearchById(List<Task>, int 🟩	72	2		7	9	5

2. Tuning petlji: Minimiziranje rada unutar petlje

Smanjenje operacija unutar petlje povećava efikasnost. Operacije koje se ne moraju ponavljati u svakoj iteraciji izdvajaju se iz petlje.
Uklanjanje provjeravanja stanja trenutni.zavrsen nakon umetanja u odgovarajuću poziciju. Provjeravanje se može obaviti prije početka petlje jer ne utiče na osnovnu logiku umetanja.



```
taskovi.RemoveAll(task => task.zavrsen);
for (int i = 1; i < taskovi.Count; i++)
{
    Task trenutni = taskovi[i];
    int j = i - 1;
    while (j >= 0 && taskovi[j].vaznost > trenutni.vaznost)
    {
        taskovi[j + 1] = taskovi[j];
        j--;
    }
    taskovi[j + 1] = trenutni;
}
```

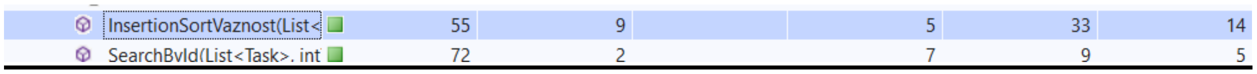


Diagnostics session: 1:43 minutes (1:42 min selected)

Add counter graphs by checking counters from counter options

ID	Time	Objects (Diff)	Heap Size (Diff)	Label
1	0.44s	16,024 (n/a)	1,362.25 KB (n/a)	
2	103.38s	16,331 (+307)	1,426.72 KB (+64.47 KB)	

Vrijeme potrošnje: 103.38-0.44=102.94s
Potrošnja memorije:1426.72-1362.25=64,47KB

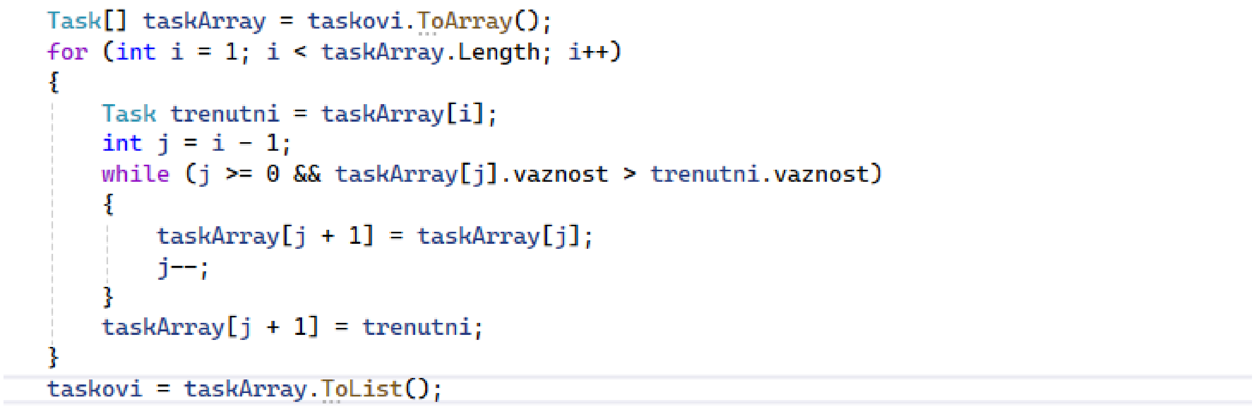


InsertionSortVaznost(List< 🟩	55	9		5	33	14
SearchById(List<Task>, int 🟩	72	2		7	9	5

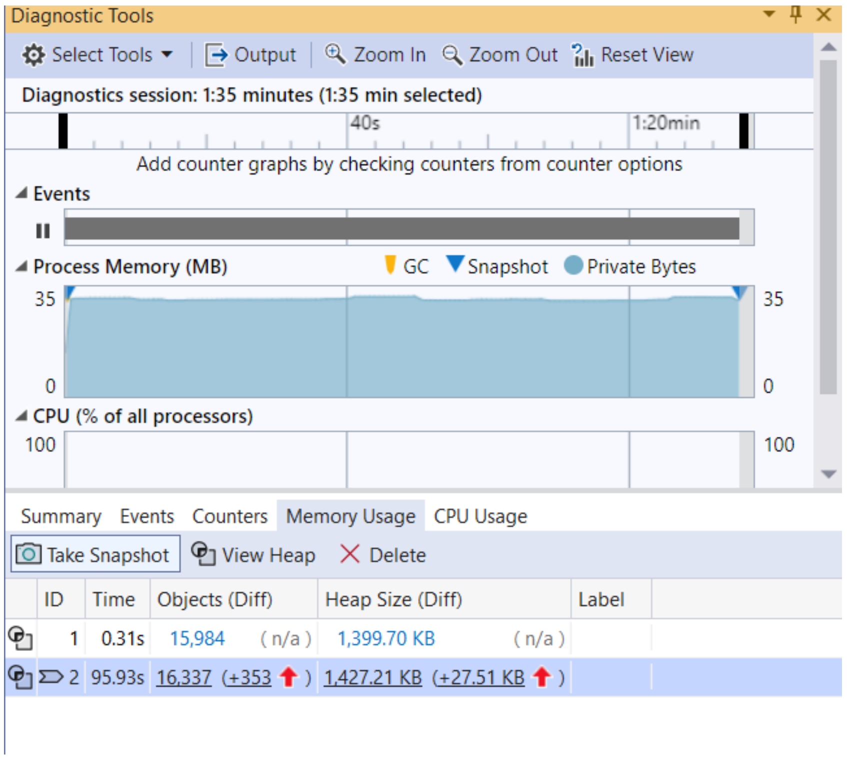
3. Tuning podataka: Korištenje integer tipa i minimiziranje referenciranja nizova

Korištenje manje zahtjevnih tipova podataka, kao što su int, umjesto referenci i složenih objekata, može smanjiti vrijeme izvršavanja. Također,

smanjenje nepotrebnog pristupa listama ili nizovima dodatno povećava performanse.
U kodu kreiram lokalni niz za zamjenu referenciranja taskovi unutar petlji.



```
Task[] taskArray = taskovi.ToArray();
for (int i = 1; i < taskArray.Length; i++)
{
    Task trenutni = taskArray[i];
    int j = i - 1;
    while (j >= 0 && taskArray[j].vaznost > trenutni.vaznost)
    {
        taskArray[j + 1] = taskArray[j];
        j--;
    }
    taskArray[j + 1] = trenutni;
}
taskovi = taskArray.ToList();
```

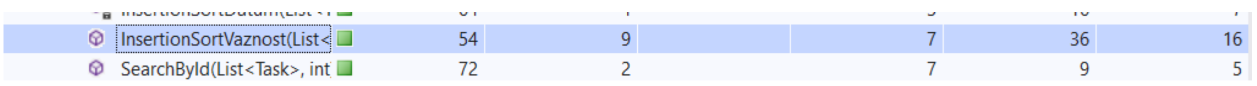


Diagnostics session: 1:35 minutes (1:35 min selected)

Add counter graphs by checking counters from counter options

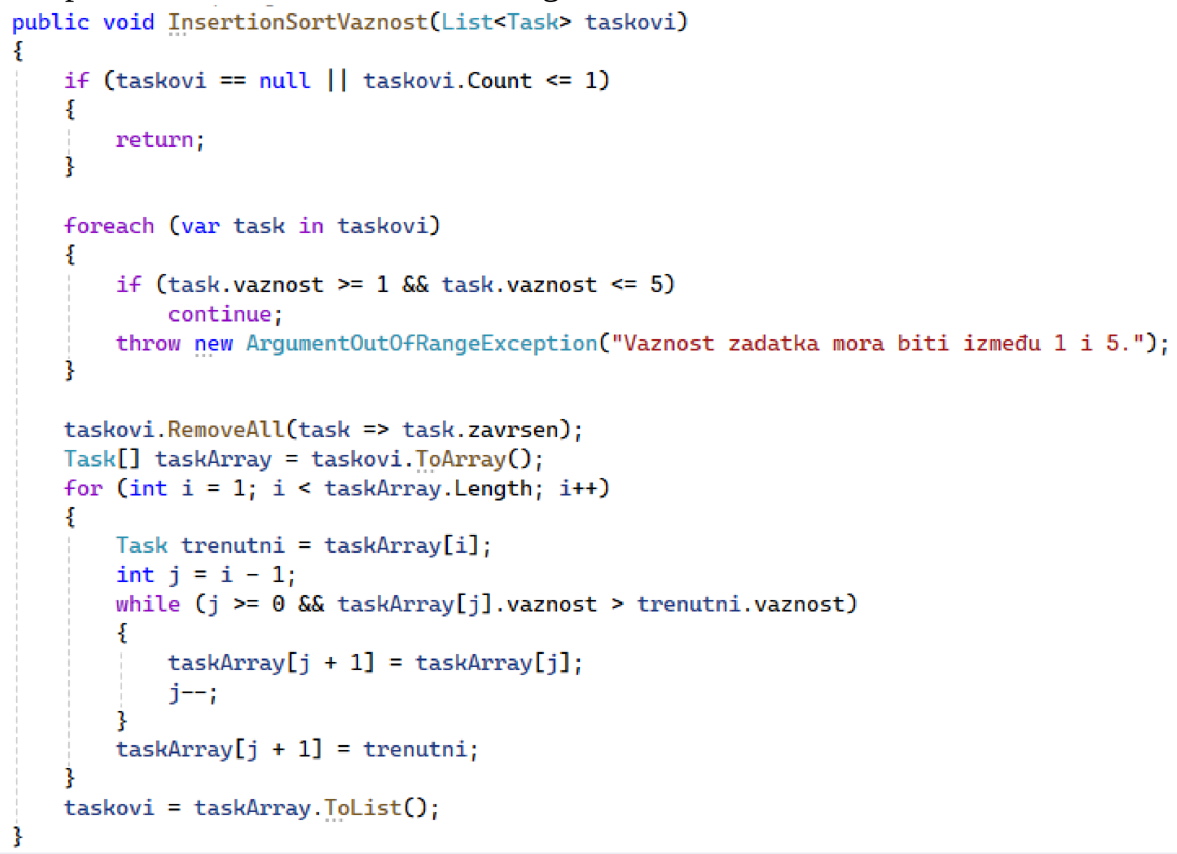
ID	Time	Objects (Diff)	Heap Size (Diff)	Label
1	0.31s	15,984 (n/a)	1,399.70 KB (n/a)	
2	95.93s	16,337 (+353 ↑)	1,427.21 KB (+27.51 KB ↑)	

Vrijeme potrošnje: 95.93-0.31=95.62s
Potrošnja memorije:1427.21-1399.70=28.04KB



InsertionSortVaznost(List< 🟩	54	9		7	36	16
SearchById(List<Task>, int 🟩	72	2		7	9	5

Kompletna metoda nakon code tuninga:



```
public void InsertionSortVaznost(List<Task> taskovi)
{
    if (taskovi == null || taskovi.Count <= 1)
    {
        return;
    }

    foreach (var task in taskovi)
    {
        if (task.vaznost >= 1 && task.vaznost <= 5)
            continue;
        throw new ArgumentOutOfRangeException("Vaznost zadatka mora biti između 1 i 5.");
    }

    taskovi.RemoveAll(task => task.zavrsen);
    Task[] taskArray = taskovi.ToArray();
    for (int i = 1; i < taskArray.Length; i++)
    {
        Task trenutni = taskArray[i];
        int j = i - 1;
        while (j >= 0 && taskArray[j].vaznost > trenutni.vaznost)
        {
            taskArray[j + 1] = taskArray[j];
            j--;
        }
        taskArray[j + 1] = trenutni;
    }
    taskovi = taskArray.ToList();
}
```

Test korišten za code tuning:

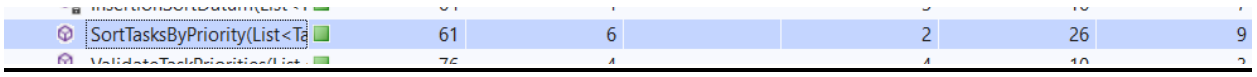
```
public void InsertionSort_ShouldSortLargeListOfTasksCorrectly()
 {

     var random = new Random();
     var tasks = new List<Task>();
     for (int i = 0; i < 100000; i++)
     {
         tasks.Add(new Task(2, $"Task {i + 1}", random.Next(1, 5),
DateTime.Now.AddDays(1), Kategorija.sport, false));
     }


     var taskService = new TaskService();
     taskService.InsertionSortVaznost(tasks);


     for (int i = 1; i < tasks.Count; i++)
     {
         Assert.IsTrue(tasks[i - 1].vaznost <= tasks[i].vaznos);
     }
}
```

4. Refaktoring

Uradila sam refaktorisanje na nivou metoda, refaktorisanje na nivou iskaza i refaktorisanje na nivou podataka.

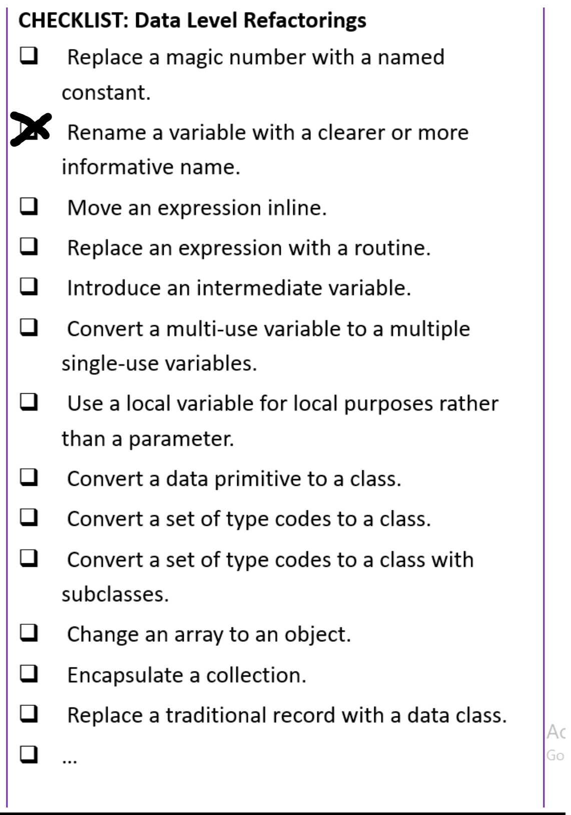
```
public void SortTasksByPriority(List<Task> tasks)
 {

     if (tasks == null || tasks.Count <= 1)
     {
         return;
     }


     ValidateTaskPriorities(tasks);


     for (int i = 1; i < tasks.Count; i++)
     {
         Task currentTask = tasks[i];
         int j = i - 1;

         while (j >= 0 && tasks[j].vaznost > currentTask.vaznost)
         {
             tasks[j + 1] = tasks[j];
             j--;
         }

         tasks[j + 1] = currentTask;
     }
 }

 private void ValidateTaskPriorities(List<Task> tasks)
 {
     foreach (var task in tasks)
     {
         if (task.vaznost < 1 || task.vaznost > 5)
         {
             throw new ArgumentOutOfRangeException(nameof(task.vaznost), "Vaznost
mora biti između 1 i 5.");
         }
     }
   }
```

Code Metrics:



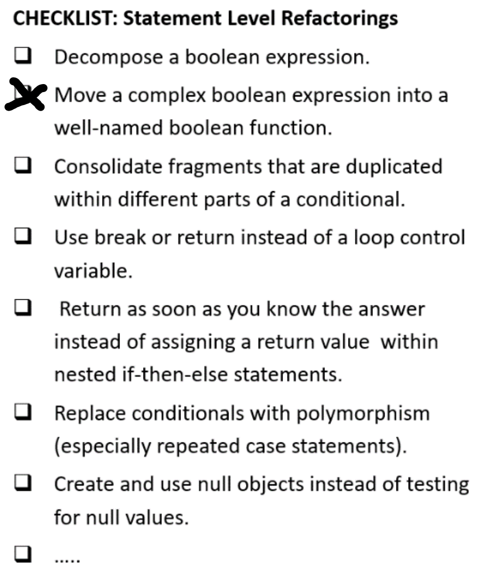
SortTasksByPriority(List<Ta	🟩	61	6		2	26	9

Dobili smo kod koji je lakse za shvatiti i odrzavati. Indeks održivosti se povećao sa 56 na 61, a ciklomatska kompleksnost se smanjila sa 10 na 6.



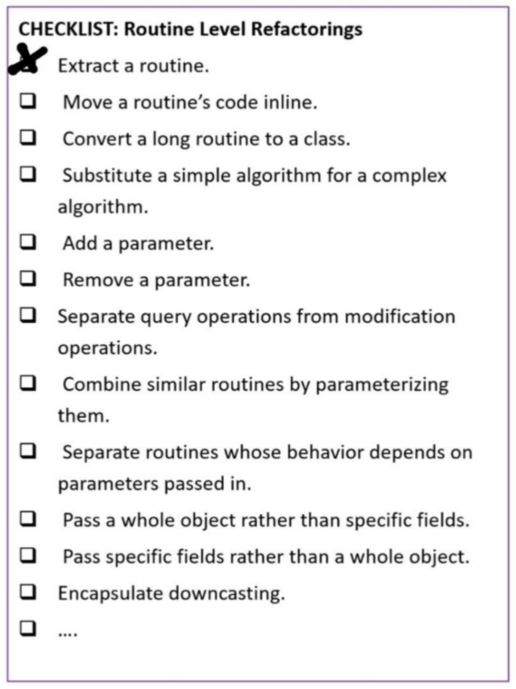
CHECKLIST: Data Level Refactorings

- ☐ Replace a magic number with a named constant.
- [✓] Rename a variable with a clearer or more informative name.
- ☐ Move an expression inline.
- ☐ Replace an expression with a routine.
- ☐ Introduce an intermediate variable.
- ☐ Convert a multi-use variable to a multiple single-use variables.
- ☐ Use a local variable for local purposes rather than a parameter.
- ☐ Convert a data primitive to a class.
- ☐ Convert a set of type codes to a class.
- ☐ Convert a set of type codes to a class with subclasses.
- ☐ Change an array to an object.
- ☐ Encapsulate a collection.
- ☐ Replace a traditional record with a data class.
- ☐ ...



CHECKLIST: Statement Level Refactorings

- ☐ Decompose a boolean expression.
- [✓] Move a complex boolean expression into a well-named boolean function.
- ☐ Consolidate fragments that are duplicated within different parts of a conditional.
- ☐ Use break or return instead of a loop control variable.
- ☐ Return as soon as you know the answer instead of assigning a return value within nested if-then-else statements.
- ☐ Replace conditionals with polymorphism (especially repeated case statements).
- ☐ Create and use null objects instead of testing for null values.
- ☐ .....



CHECKLIST: Routine Level Refactorings

- [✓] Extract a routine.
- ☐ Move a routine's code inline.
- ☐ Convert a long routine to a class.
- ☐ Substitute a simple algorithm for a complex algorithm.
- ☐ Add a parameter.
- ☐ Remove a parameter.
- ☐ Separate query operations from modification operations.
- ☐ Combine similar routines by parameterizing them.
- ☐ Separate routines whose behavior depends on parameters passed in.
- ☐ Pass a whole object rather than specific fields.
- ☐ Pass specific fields rather than a whole object.
- ☐ Encapsulate downcasting.
- ☐ ....