

Gebze Technical University  
Computer Engineering

CSE 222  
2017 Spring

### HOMEWORK 3 REPORT

FÜRKAN YILDIZ  
141044031

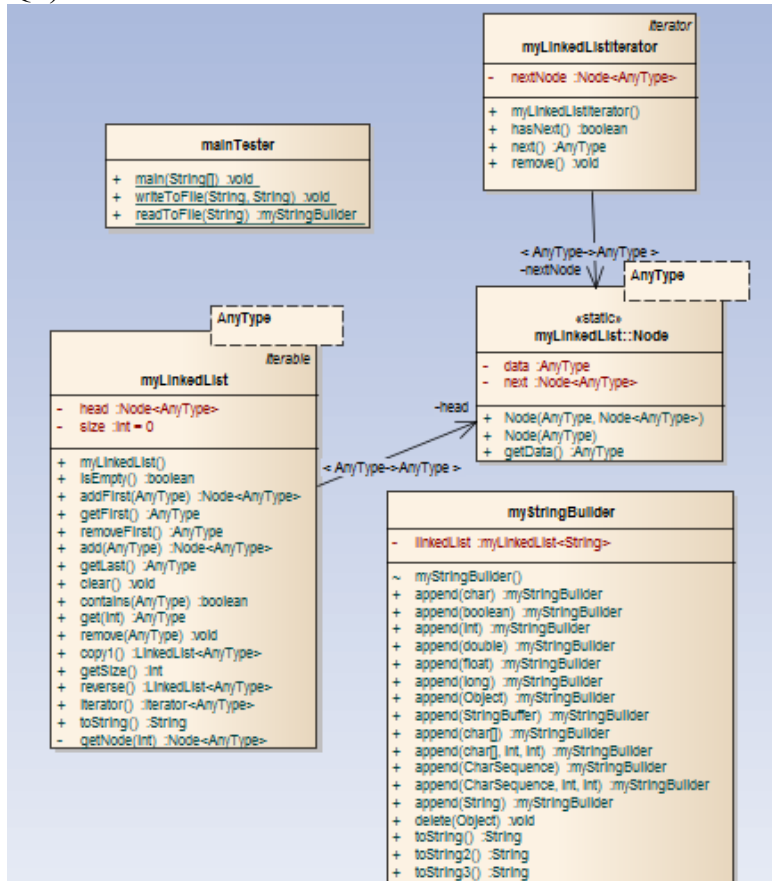
Course Assistant: NUR BANU ALBAYRAK

# 1. System Requirements

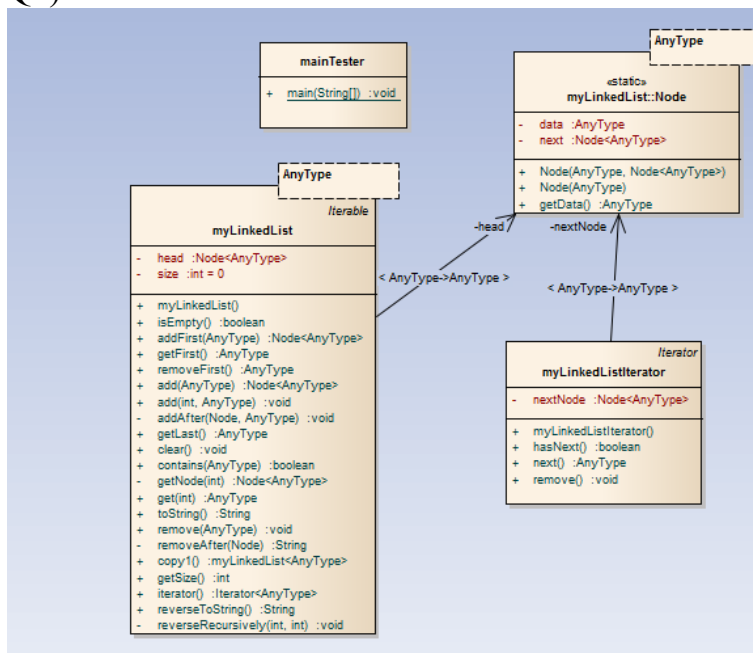
Java yüklü olmalı.

## 2. Class Diagrams

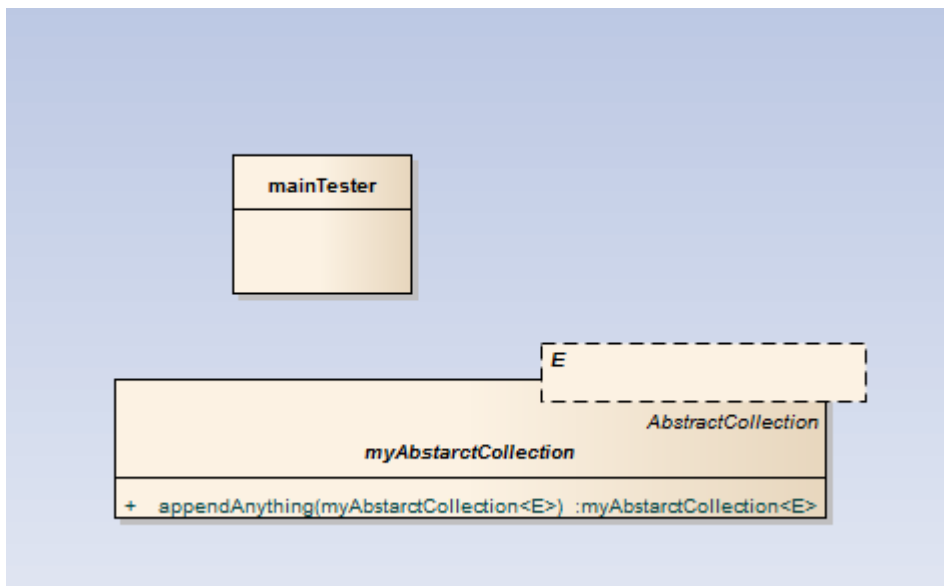
Q1)



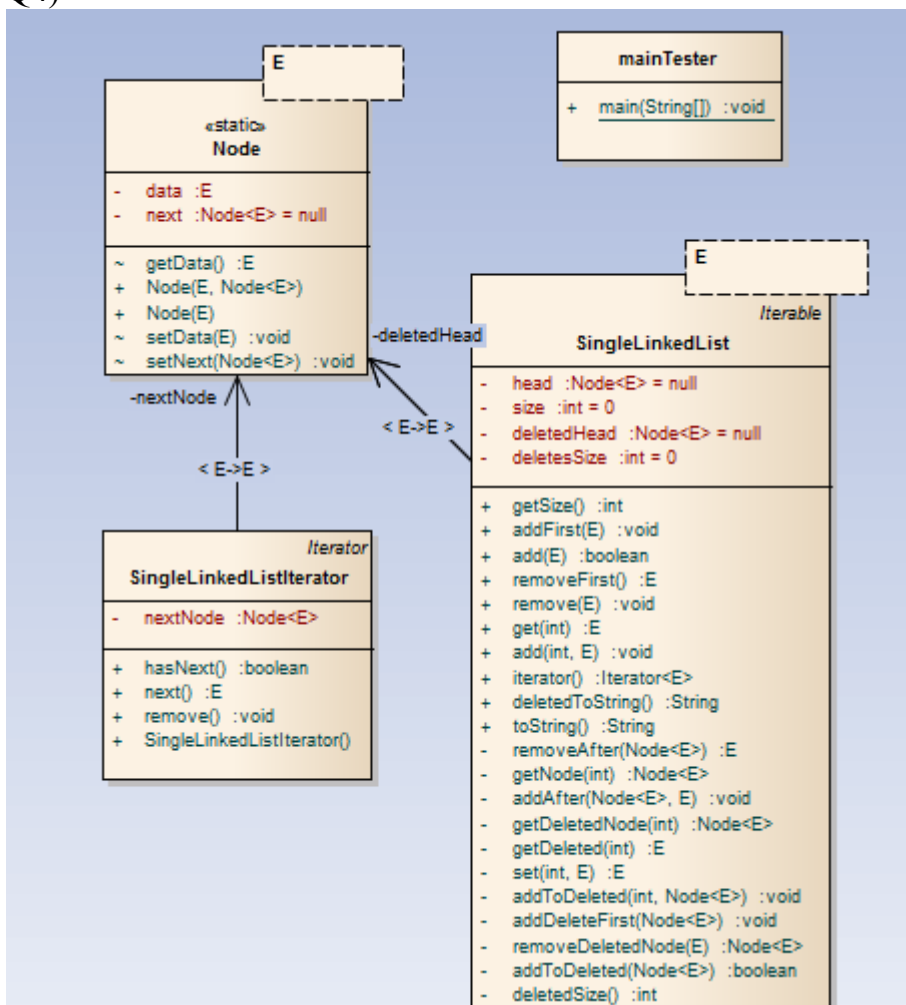
Q2)



Q3)



Q4)



### 3. Problem Solitions Approach

Q1) StringBuilder classına benzer bir class impliment edebilmek için arka planda tüm işleri yapmak üzere bir single linked list tuttuk. StringBuilder classı içerisinde Stringler tuttuğu için, myStringBuilder classında, Single linked listi String olarak instance ettik. Bu single linked list içerisindeki node yapısı sayesinde tüm elemanları tutacak. StringBuilder classının yaklaşık 15 adet farklı tip alan append metdounu impliment edebilmek için gelen referansları *String.valueOf* metodu ile *stringe* çevirdirdik ve *arkaplanda tuttuğumuz single linked listimize* ekledik. Daha sonra 3 farklı *toString* metodu impliment ederken, 1. *toString* için *single linked listin indexleri (aslında linked listin yapısında index yok ama biz ekliyoruz)* ve *bu indexler ile linked listin size'ına kadar giderken get metotlarını kullanıp her elemana teker teker erişerek bir string oluşturduk.*

*İkinci toStringi oluşturmak için single linked listimize iterator yazmamız gerekti, bu yüzden önce yazdığımız single linked list classını* Iterable interfacesinden impliment ettik ardından bir iterator classı yazarak bunu Iterator interfacesinden impliment ettik ve oluşturduğumuz bu iterator classına headi (ilk node referansı) , kendi içerisindeki head'a atan bir construc-ture, head.next'in null olup olmadığını kontrol ederek, ileride node olup olmadığını söyleyen hasNext methodu ve eğer ileride node varsa, yeni nodeyi eski nodenin nextine atarak ilerileyen bir next methodu yazdık. Remove methodunu ise impliment etmeye gerek olmadığı için exception fırlatacak şekilde bıraktık. Iterator classımızda hazır hale geldikten sonra ikinci toStringi impliment ederken önce single linked listimizin iterator objesini oluşturduk ardından bu obje üzerinden ileride node var mı kontrolü yaparak ilerleyip ilerlerken geçtiğimiz nodelerin stringlerini toplayarak return ettik. Üçüncü toString methodunda ise Single linked listimizin toString methodunu çağırdık, bu method ise for each döngüsü ile tüm nodeleri gezerek gezdiği referanslardaki stringleri topluyor ve son elemana gelince return ediyor.

*Bu toStringleri analiz edecek olursak,*

1. *toString (get kullanılarak yapılan):* her node ulaşmak için tek tek tüm nodları tekrar tekrar gezdiği için en yavaş çalışan algoritmaya sahip. Orneğin 40. Nodedeki değeri elde etmek için 40. Nodeye kadar tek tek her nodu gezmemiz lazım. Daha sonra 41. Nodedeki değeri elde etmek için ise tekrar geçtiğimiz 40 nodeden geçiceğiz ve 1 node daha geçiceğiz.  
Çalışma süresi quadratic time. ( $n^2$ )
2. *toString(iterator kullanılarak):* linked listimizin üzerinde iterator kullanarak geziyoruz, gezme işlemini yaparken her seferinde önce ileride node var mı diye kontrol ediyoruz, ardından eğer eleman varsa ilerleyip onun değerini alıyoruz. Tekrar tekrar başa dönme işlemi yok, tek seferde kontrollü bir biçimde ilerleniyor. İlk yaptığımız toString methodundan oldukça hızlı.  
Çalışma süresi linear time. ( $n$ )
3. *toString (linked list kullanılarak):* Linked list ise for each döngüsü ile her nodeyi kontrol falan olmaksızın (alt yapısında iterator kullanarak ve nerede duracağını bile-rek) ilerleyerek içerisindeki datayı bastırıyor.  
Çalışma süresi constant time (1)

Sonuç olarak en hızlı 3. Sonra 2. Ve en yavaş 1. toString metotları.

Q2)

Single linked listi reverse edebilmek için, ilk indexdeki eleman ile son indexdeki elemanı yer değiştiren bir kod yazdık, bu kodu sürekli durumuna getirmek için ise methoda ilk ve son indexi alan parametreler ekledik. Böylece 1. Defa ilk ve son indexdeki nodeleri yer değiştirdikten sonra 2. Defa da methodu çağırırken ilk indexi aslında ilkinden bir sonraki index son indexide sondan bir önceki index olarak verirsek ve bunu sürekli olarak ilk index size'ın yarısına kadar gelinceye dek yapar isek, tüm linked listi ters çevirmiş oluyoruz. Yazdığımız bu methodu, tüm linked list ters çevirilsin diye istendiği için bir wrapper classın içine koyduk. Aynı zamanda bu method ana linked listide ters çevirdiğinden bu methodu linked listin hard copy'si üzerinden yaptık böylece ana linked list bu işlemten etkilenmeden reverse edilmiş halini return etti.

Q3)

Önce AbstractCollection abstract classından myAbstractCollection abstract classını extend ettik, ardından iki tane myAbstractCollection objesini birbirine bir methodla eklemeye çalıştık bu method 1 adet myAbstractCollection referansı alacak, ikinci referans classın kendisinden gelecek bu ikisini birbirine ekleyerek (birincisine) return edecek. Bu ekleme işlemini yaparken, 2. myAbstractCollection referansı üzerinde bir iterator yardımıyla gezdik ve ileride eleman olduğu sürece o elemanı 1. myAbstractCollection obje referansına ekledik. 2. Referanstaki tüm elemanları geztikten sonra ise ilkinin referansını return ettik.

Q4)

Silinen Node ları tekrar kullanabilmemiz için bir yerde tutmamız gerekli. Tutacağımız yerin en uygun yine bir node yapısı olduğuna karar verdim ve silinen noddardanda bir linked list oluşturmaya karar verdim. Bunun için bir **deletedHead** adında Node referansı tuttum. Bu node aynı linked listin headi gibi iş görecek. Eğer ana single linked listimizden bir obje referansı silinirse bu objenin referansını silinen node lar ile oluşturduğumuz single linked listimize eklemeliyiz. Eğer ana single linked listimize bir obje referansı ekleyecek olur isek, ilk önce silinen Node lerde bir referans tutulup tutulmadığını kontrol etmeliyiz, eğer bir referans tutuluyorsa herhangi bir new (hafızadan yeni yer alma) işlemine girmeden deleted node için kullandığımız referansı direk kullanabiliriz. Referansı ana single linked listimize ekledikten sonra tek yapmamız gereken o referansın içerisindeki eski datayı, yenisi ile değiştirmek. Bu şekilde 2 farklı node uzantısı tutarak, birisi kullanacağımız datalar için diğeri silinen datalar için, single linked listimizden bir obje referansı sildiğimizde o obje referansını hiç kaybetmeden daha sonra tekrar memory allocation yapmamak için tuttuğumuz ikinci node uzantısına ekliyoruz ve single linked listimize bir eleman eklediğimize tekrar o tuttuğumuz referansı kullanıyoruz. Bu işlemleri yapabilmek için single linked listin tüm ekleme ve çıkartma methodları üzerinde değişiklikler yapmam gerekti. Örneğin bir şey çıkarttığımızda onu deleted node uzantımıza ekledim. Yeni bir obje referansı eklemek istediğimizde ise ilk önce deleted Nodeslerin boş olup olmadığını kontrol ediyoruz. Eğer boş ise yeni bir obje referansı oluşturuyoruz new ile, dolu ise oradaki referansları kullanıyoruz. Ayrıca deletedNodeleri eklemek ve silmek için methodlar oluşturmamız gerekti.

#### 4. Test Cases

Q1)

- 100.000 adet random integer üreten (1 ile 10 milyon arasında) ve bu ürettiği sayıları “numbers.txt” adında bir dosyaya yazan kod yazıldı.
- Yazılan 3 farklı toString metodu ile obje 3 farklı şekilde stringe çevirildi.
- toString metotlarının return ettikleri stringler “result1.txt”, “result2.txt” ve “result4.txt” adında 3 farklı dosyaya yazdırıldı.
- 3 farklı toString methodunu birbirleriyle karşılaştırmak için toString metotlarının içerisinde süre tutuldu.

Q2)

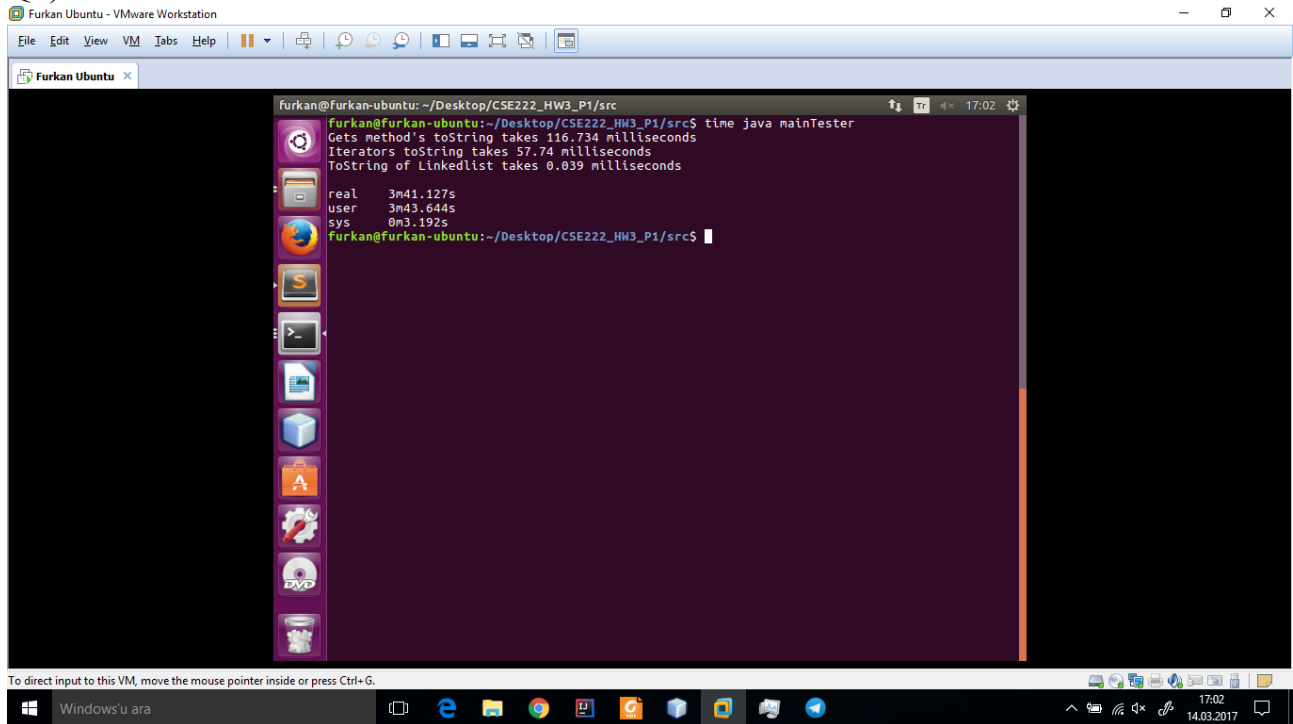
- myLinkedList referansını String olarak instance ettik.
- bu referansa 6 adet string ekledik.
- Ekleme işleminin ardından önce referansın toStringini, sonra reverse methodundan dönen stringi ve daha sonra tekrar referansın toStringini bastırdık.

Q4)

- Hiçbir şey yapılmadan önce linkedlist ve deletedNodes bastırıldı. Boş olmaları gerek ikisinde
- Single linked list objesi oluşturuldu.
- Objeye 100 adet integer eklendi.
- Eklenen 100 integerdan 50 si silindi.
- Objenin ve deletedLinkedListin size ları kontrol edildi. 50-50 olması gerek.
- Objeye 100 adet daha integer eklendi.
- Objenin ve deletedLinkedListin size ları kontrol edildi. 150-0 olması gerek.

#### 5. Running and Results

Q1)



The screenshot shows a VMware Workstation window titled "Furkan Ubuntu - VMware Workstation". Inside the window is a virtual machine named "Furkan Ubuntu". The terminal window shows the following output:

```
furkan@furkan-ubuntu: ~/Desktop/CSE222_HW3_P1/src
furkan@furkan-ubuntu:~/Desktop/CSE222_HW3_P1/src$ time java mainTester
Gets method's toString takes 116.734 milliseconds
Iterators toString takes 57.74 milliseconds
ToString of LinkedList takes 0.039 milliseconds

real    3m41.127s
user    3m43.644s
sys     0m3.192s
furkan@furkan-ubuntu:~/Desktop/CSE222_HW3_P1/src$
```

The terminal window is running on a Linux system (Ubuntu) with a desktop environment. The window title bar shows "Furkan Ubuntu" and "VMware Workstation". The terminal output shows the execution of a Java program named "mainTester" which measures the time taken by different toString methods. The results are displayed in a table format.

Q2)

```
CSE222_HW3_P2 - [C:\Users\yldzf\IdeaProjects\CSE222_HW3_P2] - [CSE222_HW3_P2] - ...src\mainTester.java - IntelliJ IDEA 2016.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
CSE222_HW3_P2 > src > mainTester >
Run mainTester
[C:\Program Files\Java\jdk1.8.0_111\bin\java] ...
LL: [Merhaba, ben, Furkan, YILDIZ, 141044031, GTU]
Reversed LL: [GTU, 141044031, YILDIZ, Furkan, ben, Merhaba]
After reversed it shouldn't be change LL: [Merhaba, ben, Furkan, YILDIZ, 141044031, GTU]
Process finished with exit code 0
```

Q4)

```
CSE222_HW3 - [C:\Users\yldzf\IdeaProjects\CSE222_HW3] - [CSE222_HW3] - ...src\mainTester.java - IntelliJ IDEA 2016.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
CSE222_HW3 > src > mainTester >
Run mainTester
[C:\Program Files\Java\jdk1.8.0_111\bin\java] ...
[]
[]
100 integer eklendikten sonra:
[1, 26, 51, 76, 101, 126, 151, 176, 201, 226, 251, 276, 301, 326, 351, 376, 401, 426, 451, 476, 501, 526, 551, 576, 601, 626, 651, 676, 701, 726, 751, 776, 801, 826, 851, 876, 901, 926, 951, 976, 1000]
Single Linked list size:100
[]
Deleted Nodes size:0
50 integer silindikten sonra:
[1251, 1276, 1301, 1326, 1351, 1376, 1401, 1426, 1451, 1476, 1501, 1526, 1551, 1576, 1601, 1626, 1651, 1676, 1701, 1726, 1751, 1776, 1801, 1826, 1851, 1876, 1901, 1926, 1951, 1976, 2000]
Single Linked list size:50
[1, 26, 51, 76, 101, 126, 151, 176, 201, 226, 251, 276, 301, 326, 351, 376, 401, 426, 451, 476, 501, 526, 551, 576, 601, 626, 651, 676, 701, 726, 751, 776, 801, 826, 851, 876, 901, 926, 951, 976, 1000]
Deleted Nodes size:50
Tekrar 100 integer eklendikten sonra:
[1251, 1276, 1301, 1326, 1351, 1376, 1401, 1426, 1451, 1476, 1501, 1526, 1551, 1576, 1601, 1626, 1651, 1676, 1701, 1726, 1751, 1776, 1801, 1826, 1851, 1876, 1901, 1926, 1951, 1976, 2000]
Single Linked list size:150
[]
Deleted Nodes size:0
Process finished with exit code 0
```