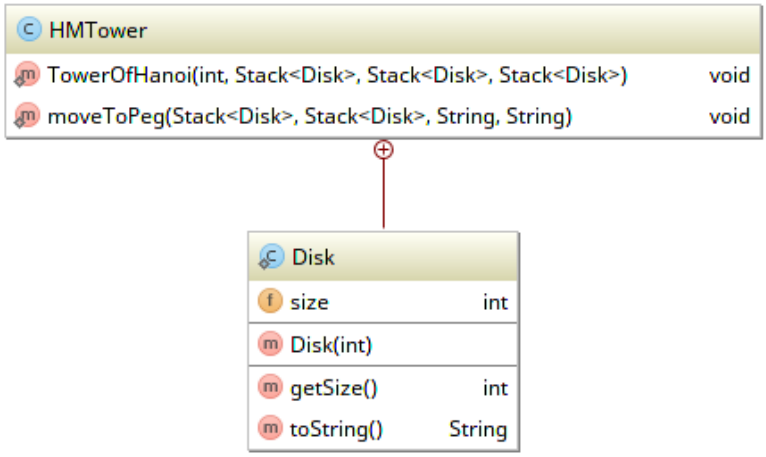
HASAN MEN – 131044009

GTU CSE 222 Data Structure and Algrorithm

HW5

https://github.com/hmenn/CSE222-2016HW5-Hanoi

Part 1. Iterative Tower of Hanoi



1. Gereksinimler
   1.1. Diskleri ilk basta ve daha sonraki adımlarda hareket ettirebilmek için 3 tane stack olmalı.
   1.2. Source stackine ilk basta disk sayisi kadar disk initialize edilip push edilecek. Destination tüm disklerin son konumu olacak. Auxiliary ise arada ki swap işlemleri için kullanılacak.
   1.3. Stackler arasi eleman cekme ve aktarma icin move metodu yazilacak.
   1.4. Bos stack durumunda direk eleman eklenebilir.
   1.5. Stackte eleman varsa küçük olan diskin üzerine büyük disk gelemez.

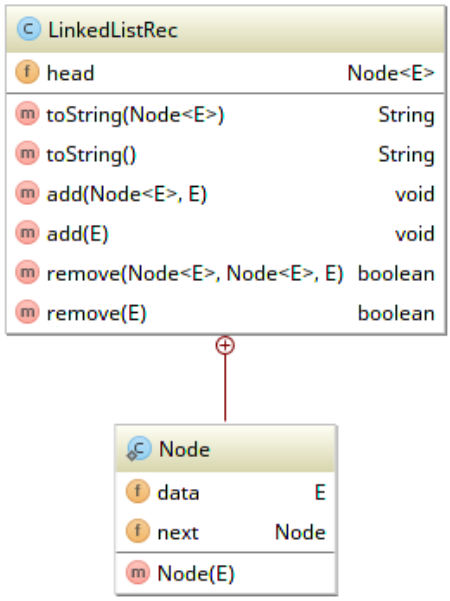
2. Testler
   2.1. 3 disk için örnek çözüm

```
###### TOWER OF HANOI ######
Move the disk 1 from Source peg to Destination peg
Move the disk 2 from Source peg to Auxiliary peg
Move the disk 1 from Destination peg to Auxiliary peg
Move the disk 3 from Source peg to Destination peg
Move the disk 1 from Auxiliary peg to Source peg
Move the disk 2 from Auxiliary peg to Destination peg
Move the disk 1 from Source peg to Destination peg
SOLVED IN 7 MOVEMENT FOR 3 DISKS.
```

2.2. 4 disk için örnek çözüm

```
###### TOWER OF HANOI ######
Move the disk 1 from Source peg to Auxiliary peg
Move the disk 2 from Source peg to Destination peg
Move the disk 1 from Auxiliary peg to Destination peg
Move the disk 3 from Source peg to Auxiliary peg
Move the disk 1 from Destination peg to Source peg
Move the disk 2 from Destination peg to Auxiliary peg
Move the disk 1 from Source peg to Auxiliary peg
Move the disk 4 from Source peg to Destination peg
Move the disk 1 from Auxiliary peg to Destination peg
Move the disk 2 from Auxiliary peg to Source peg
Move the disk 1 from Destination peg to Source peg
Move the disk 3 from Auxiliary peg to Destination peg
Move the disk 1 from Source peg to Auxiliary peg
Move the disk 2 from Source peg to Destination peg
Move the disk 1 from Auxiliary peg to Destination peg
SOLVED IN 15 MOVEMENT FOR 4 DISKS.
```

Part 2. Recursive Remove from list



2.1 Gereksinimler ve Soruna yaklaşım

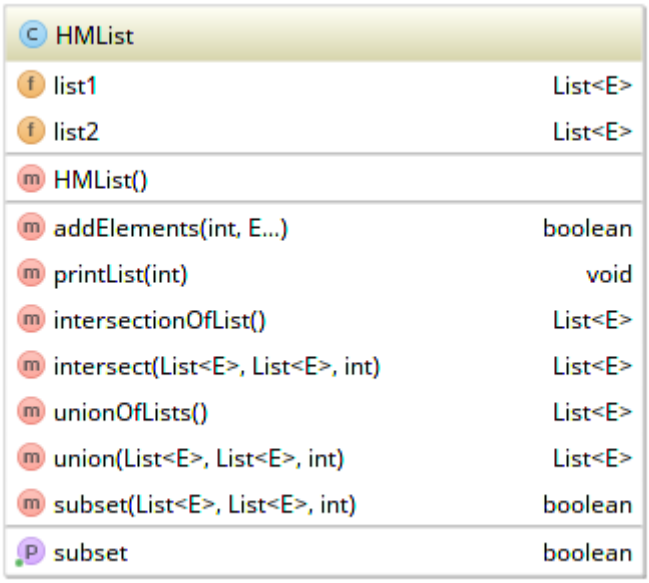
2.1.1 Silme işlemini yapabilmek için node bilgilerine ihtiyaç var. Normal kullanıcılar bunlara erişemediği içinde wrapper metod kullanılacak.

2.1.2. İşlerin %99unu node bilgisine ihtiyaç duyan remove yapacak.
2.1.3. Nodlar 2li kontrol edildiği için her nod için NULL kontrolu KESINLIKLE yapılmalı.
2.1.4. Art arda gelen aynı silinecek itemler icin this.head ile silme islemleri gercekleştirilmeli.

2.2 Testler

```
###### REMOVE TEST ######
Before remove :
2 -> 2 -> 3 -> 3 -> 4 -> 2 -> 1 -> 3 -> 2 -> 2
After remove 5 :
2 -> 2 -> 3 -> 3 -> 4 -> 2 -> 1 -> 3 -> 2 -> 2
After remove 2 :
3 -> 3 -> 4 -> 1 -> 3
After remove 4 :
3 -> 3 -> 1 -> 3
After remove 3 :
1
```

Part 3. List operations (intersection – union – subset)



3.2.1 Intersection - Union

3.2.1.0.Eğer listelerimiz sıralı ise daha kısa sürede çözüme ulaşırız.
3.2.1.1.Tüm listelerin elemanlari sonuna kadar sorgulanmali
3.2.1.2.Referans noktasi olarak küçük elemanlı listeyi alıyorum ki daha kisa sürede tamamlansın.
3.2.1.3.İlk başta recursive adımları yaparak listenin sonuna kadar gidip boş dizi ile geri geri geliyorum. Her gelmede list2de olan ve ortak (union için uniq olanlari) listeye(return edilen) daha once eklenmemişleri ortak(uniq) listeye ekliyip return ediyorum.
3.2.1.4 Metod için wrapper yazılmalı. Listlere dışarıdan direkt olarak ulaşım yok.

3.2.2 Subset

3.2.2.1 List2 nin list1in al kumesi mi diye bakacam.
3.2.2.2 list2 nin eleman sayisi fazla ise direk olarak false donder.
3.2.2.3 wrapper metdolar lazım. Ustte açıklanan nedenden dolayı
3.2.2.4 İlk elemandan başlayıp ilerliyorum dizinin sonuna kadar sorunsuz gidersem true, en ufak uygunsuz itemde (!list2.contains(list1.get(index))) subset özelliğini yitirdiği için return false ile geri recursivelere false donder.

3.3. Test

```
###### INTERSECT - UNION - SUBSET TEST ######
List1 : [3, 6, 8, 8, 8, 9, 11, 16]
List2 : [1, 3, 8, 10, 11, 14]
Intersect : [11, 8, 3]
Union : [14, 10, 1]
Subset : false
```