HASAN MEN

131044009

HW6 RAPOR

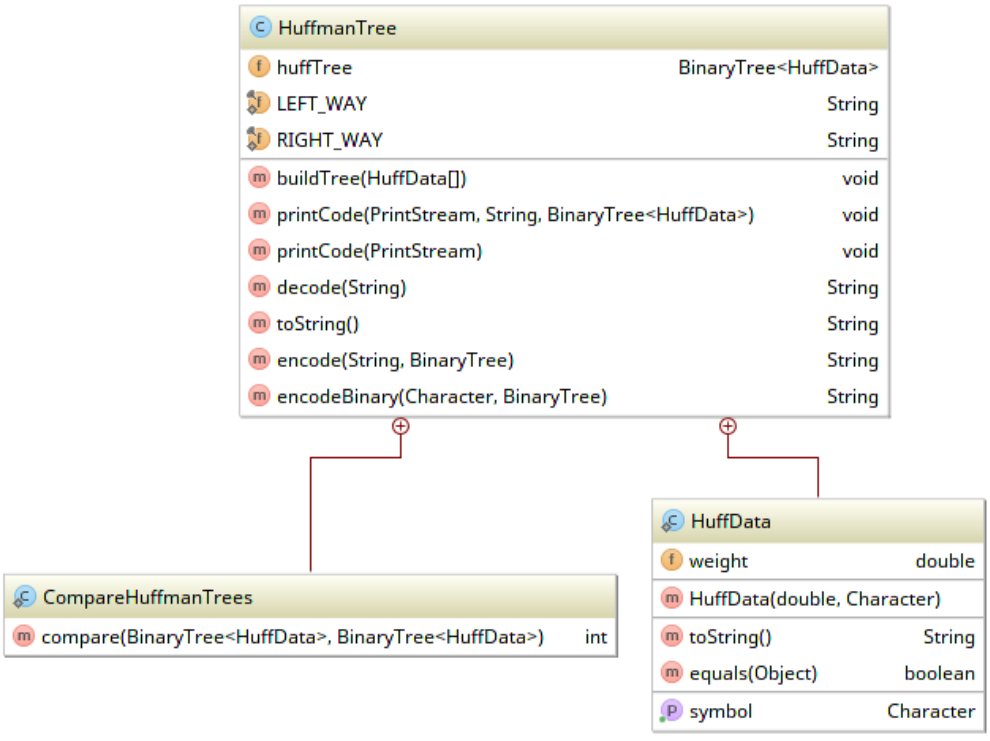
https://github.com/hmenn/CSE222-HW6-2016

Part1 – Huffman Tree Decoder

1.1. Genel bakış ve gereksinimler

-String olarak verilen bir metni şifrelememiz gerekli.
-Text parcalara (basamaklarina) ayrılacak her karakter huffman tree icinde aranacak.
-Tree uzerinde gidilen her adim kaydedilerek recursive metod yardımıyla yeri return edilecek.
-Sag yol 1, sol yol 0 olarak referans alinacak.

-



Part2 kapsamında encode metodu yazildi. Yardimci(helper) olarakta encodeBinary metodlari yazildi.

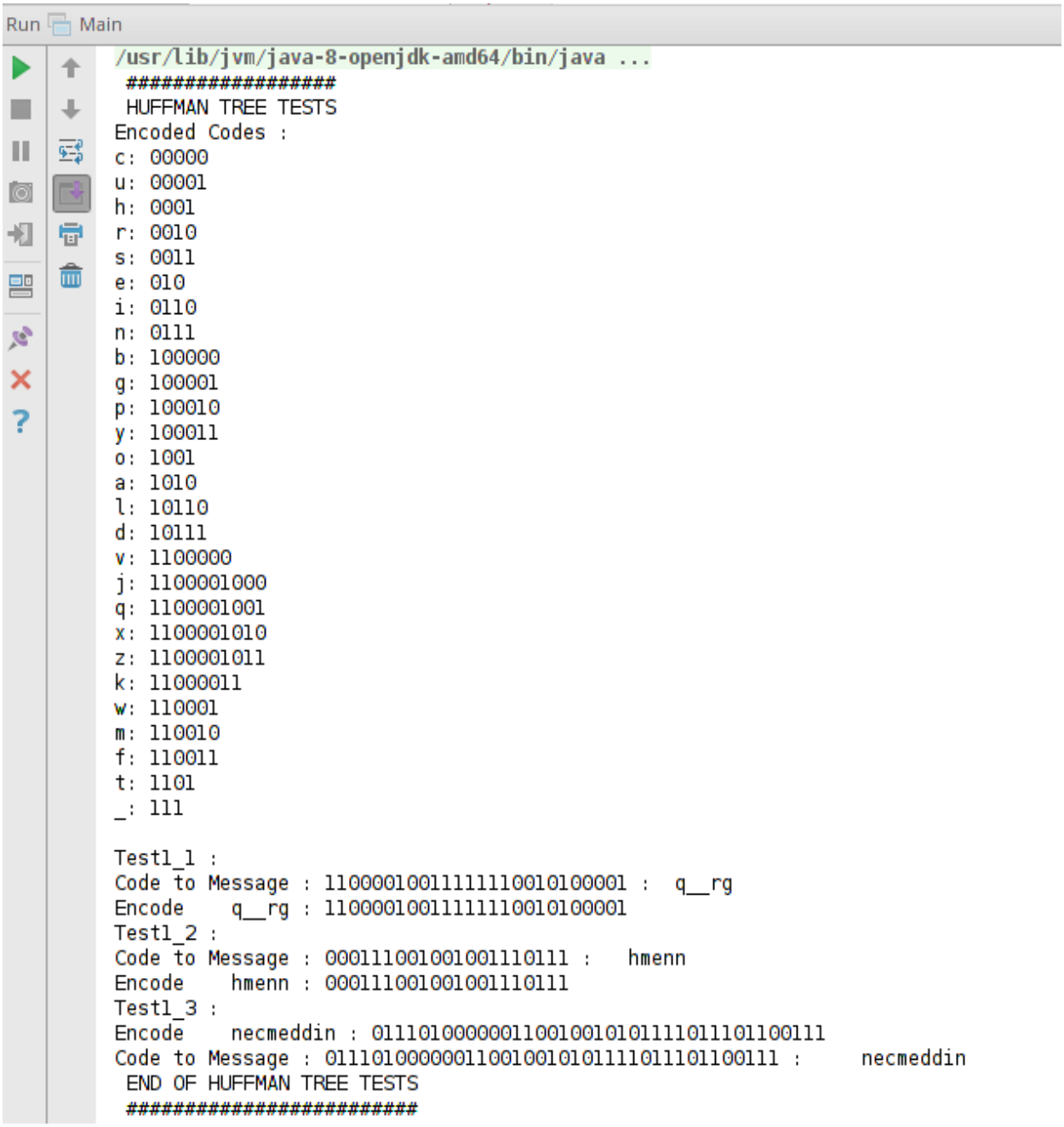
Metodlar :

encode(String,BinaryTree):String

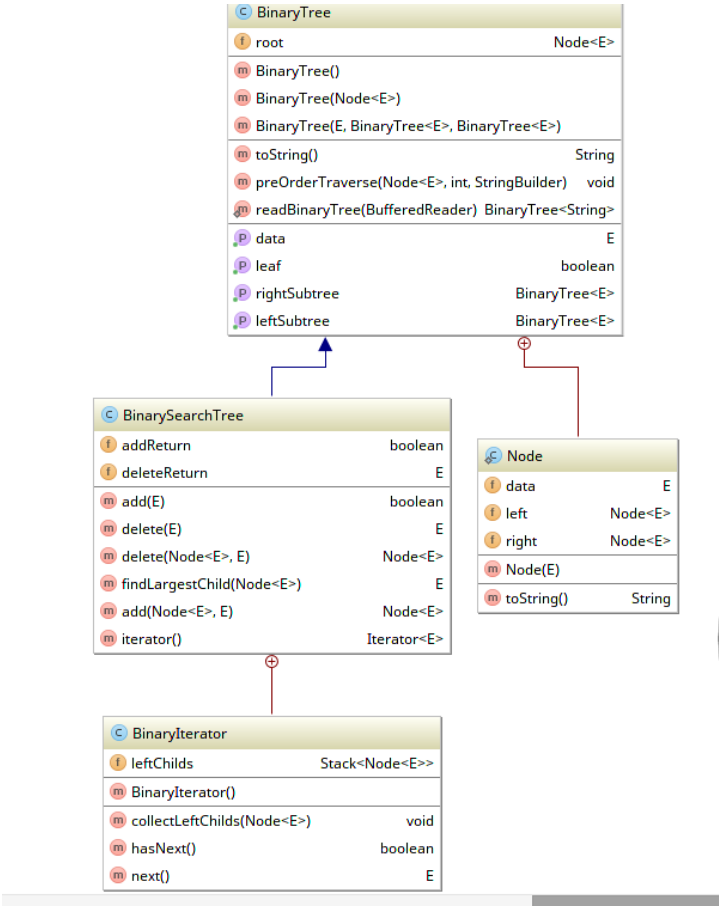
-Verilen stringi helper yardimiyla sifreler.

-BinaryTree parametresi eger daha onceden tree initialize edilmemis ise initialize etmemizi saglayip olası hatalardan kurtulmamizi saglar.

TESTLER :



PART2 : BinarySearchTree Iterator (ascading order)



- Iterator kucukten buyuge dogru elemanları gezecek.
- Iterator icin sadece next hasnext yazildi.

Temel Calisma mantigi :

Binary SeatchTree kullandıgımız icin her nodun sol childi stacklere atildi ve stackten elemanlar cekiliyor boylelikle her zaman sirali INORDER bir erisim saglaniyor. O(h) (h : height) kadar bir yer fazlalıgı var ama bu simdilik goz ardi edilebilir.

-Test edilirken random sekilde eleman eklenir sirali olarak ulasim saglanilmaya calisildi.

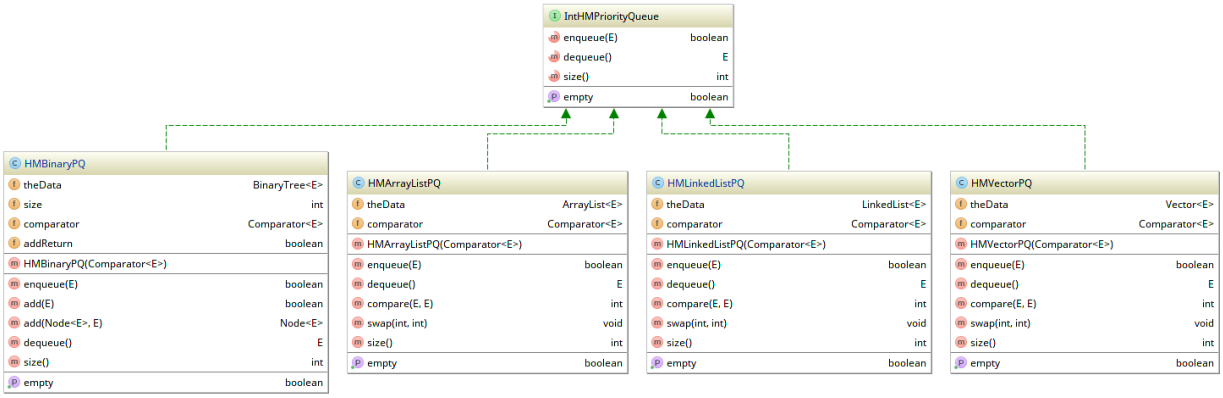
Test1 :

```
 ##################################
 BINARY SEARCH TREE ITERATOR TESTS
17
  7
    6
      2
        null
        3
          null
          null
      null
    16
      15
        13
          null
          null
        null
      null
  null

2 -> 3 -> 6 -> 7 -> 13 -> 15 -> 16 -> 17 ->
 END OF BINARY SEARCH TREE ITERATOR TESTS
 ########################################
```

Test2:

```
 ##################################
 BINARY SEARCH TREE ITERATOR TESTS
2
  1
    null
    null
  5
    3
      null
      null
    19
      6
        null
        11
          null
          null
      null

1 -> 2 -> 3 -> 5 -> 6 -> 11 -> 19 ->
 END OF BINARY SEARCH TREE ITERATOR TESTS
 ########################################
```

PART3 :



Tasarım :

Bu part kapsamında priority queue yapısı farklı adt ler ile implement edilip 1-10-100-1000-10000-100000-10000000 adet lik guncellemeler ile aralarındaki real time permansı test olculecek.
Test sonuclari Tablo.xlsx te yer almaktadir.

IntHMPriorityQueue: Interface miz 4 temel metottan oluşur. enqueue(offer), dequeue(remove),isEmpty,size.

HMArrayListPQ : Composition ile arraylist kullanılarak implement edildi.Test sonucları tabloya eklendi. Verilmli bir kullanıl olabilir.

HMVectorPQ : Composition ile vector ADT si kullanıldı. Test sonucları tabloya eklendi.

HMLinkedListPQ : LinkedList kullanılarak olusturuldu. Eleman sayisi arttıkca problemin suresinde artış yaşandı. Cok büyük veriler icin verimli degil. Guc yetmezliginden dolay buyuk sayilarda test edilmedi.

HMBinaryPQ : Binary Tree kullanildi ama implementasyon esnasinda binarySearch tree metodlari delege edilerek binary Search ADT sine uygun dizayn edildi. Metodlar buyuk oranda recursive yazildi. Recursive calışma verimi O(logn) olmasina karsin Stack yetersziliginden dolayı test asamalari dusuk sayilarda kaldı.

```
 ############################
 PART3 - PRIORITY QUEUE TESTS

ARRAYLIST PRIORITY QUEUE TESTS
 ## Test informations : [1,10,100,1000,100000,1000000,10000000] ##
Array Enqueue Times : [0, 1, 2, 7, 21, 47, 136, 3149]
Array Dequeue Times: [0, 0, 1, 10, 47, 202, 1302]
Vector Enqueue : [0, 0, 1, 4, 14, 367, 451, 2944]
Vector Dequeue : [0, 0, 1, 7, 53, 278, 2652]
LinkedList Enqueue : [0, 0, 1, 6, 287]
LinkedList Dequeue : [0, 3, 17, 142]
BinarySearchTree Enqueue : [0, 0, 0]

Process finished with exit code 0
```

**Test information arrayine paralel olarak belirtilen sayilardaki elemanlarin eklenme süreleri real time olarak (ms) verilmistir.