

Gebze Technical University
Computer Engineering

CSE 222
2017 Spring

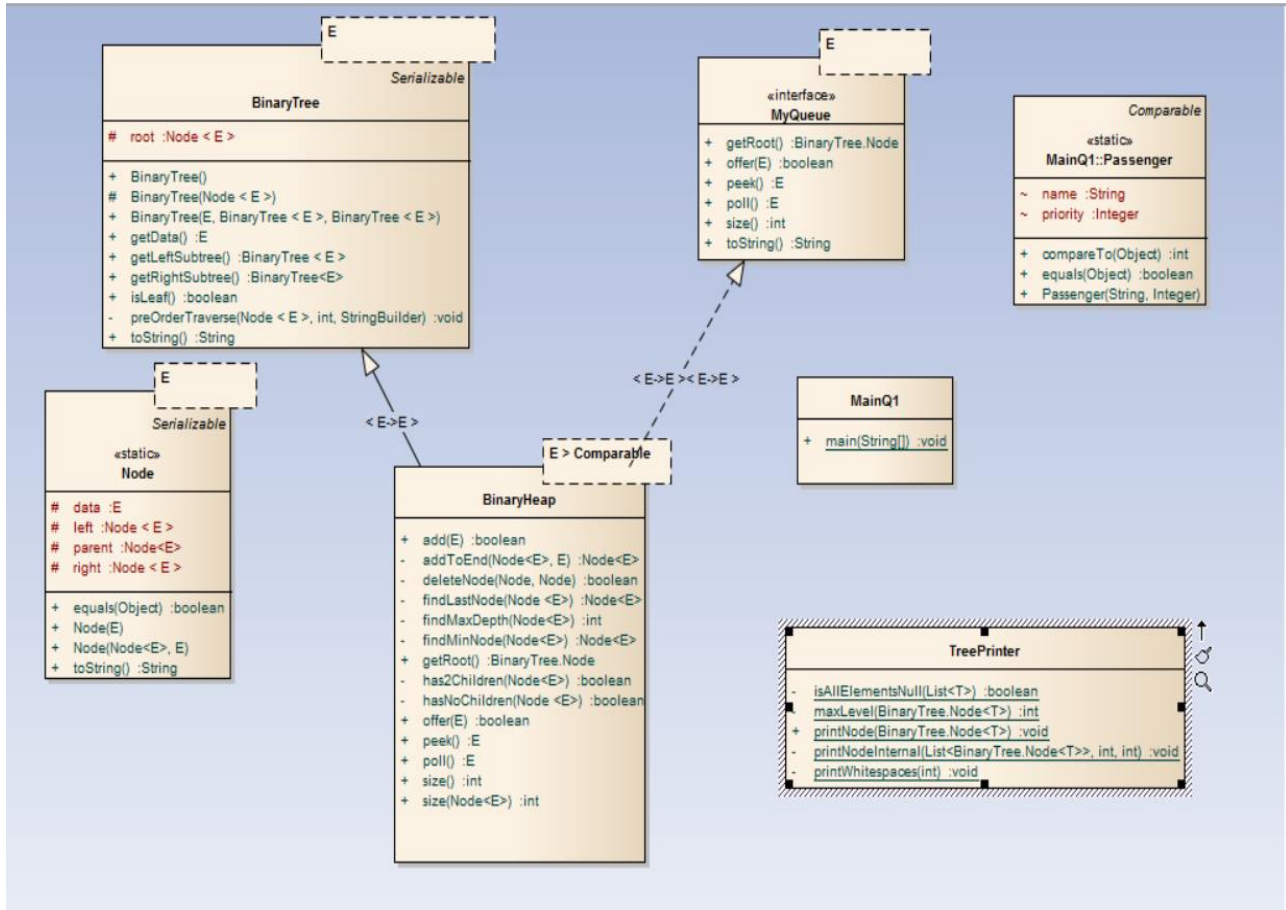
HOMEWORK 06 REPORT

Elif Şeyma ARMAĞAN
151044042

Course Assistant:
Nur Banu ALBAYRAK

Q1

1. Class Diagrams



2. Problem Solutions Approach

Soruda istenen **BinaryHeap** class ini yazabilmek için öncelikle implement etmem gereken **MyQueue** interface ini yazdım. Bunun nedeni Java'nın **Queue** interface inin birçok kullanmayacağım fonksiyona sahip olmasıydı. Kendi yazdığım **MyQueue** interface inde **poll**, **peek**, **offer** ve **size** fonksiyonlarını implement ettim. Kullanmış olduğum **BinaryTree** class ı için kitapta yer alan **BinaryTree** class ını kullandım. Ama bu class ta **BinaryHeap** e eleman eklenmesi ya da eleman çıkarılması durumunda ağaçtaki diğer nodeların yeni duruma göre şekillenebilmesi için **parent** node u da tuttum. Böylelikle ağaçta eklemek ya da silmek istediğim yeri bulmam daha kolay oldu.

Yazmış olduğum **BinaryHeap** class ında ekleme yaparken öncelikle elemanı **levelOrder** prensibinde olduğu gibi ağaçta soldan sağa düşünerek ilk boş yere ekledim. Daha sonra gerekli karşılaştırmaları yaparak eklemiş olduğum node un doğru yerini buldum. Her node un kendisinin parentı olan node lardan daha yüksek olmasına dikkat ettim. Böylelikle her leveldaki node kendi alt level ındaki nodelardan daha küçük olmuş oldu.

Silme işlemini yaparken ise sadece ağacın root node unun silinmesine izin verdim. Root node u sildikten sonra ağacın sonundaki elemanı root a koyup daha sonra ağacı yukarıda anlattığım eleman ekleme algoritmasına göre tekrar şekillendirdim.

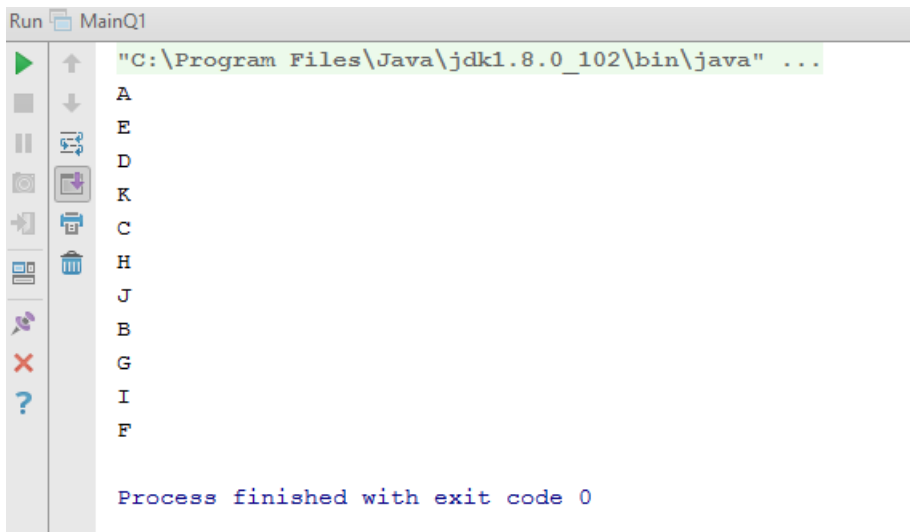
3. Test Cases

BinaryHeap class ımın test kısmı için bir adet passengers.txt dosyası hazırladım. Bu dosyada her satırda ilk harf passenger ismini ikinci sayı ise o passenger ın priority sini belirtmektedir. Test ederken öncelikle bu txt yi okudum ve her satırını bir passenger olarak oluşturdum sonra oluşturdüğüm bir BinaryHeap objesine attım. Bu obje BinaryHeap algortimasındaki en küçük sayı en tepede olur mantığına göre priority si en düşük olanı ağacın en tepesine koydu. Bu durumu engellemek için her passenger ın priority sini 1 ile böldüm. Böylelikle priority si yüksek olan yolcu 1 ile bölündüğü için aslında daha düşük priority e sahipmiş gibi oldu ve ağacın en yukarısında yer aldı. Bu işlem sonucunda her bir yolcuyu işleme almak istediğimde BinaryHeap imden poll yaparak en yüksek priority ye sahip yolcuyu elde etmiş oldum.

Bu durumu göstermek için yazdığım passengers.txt şu şekildedir :

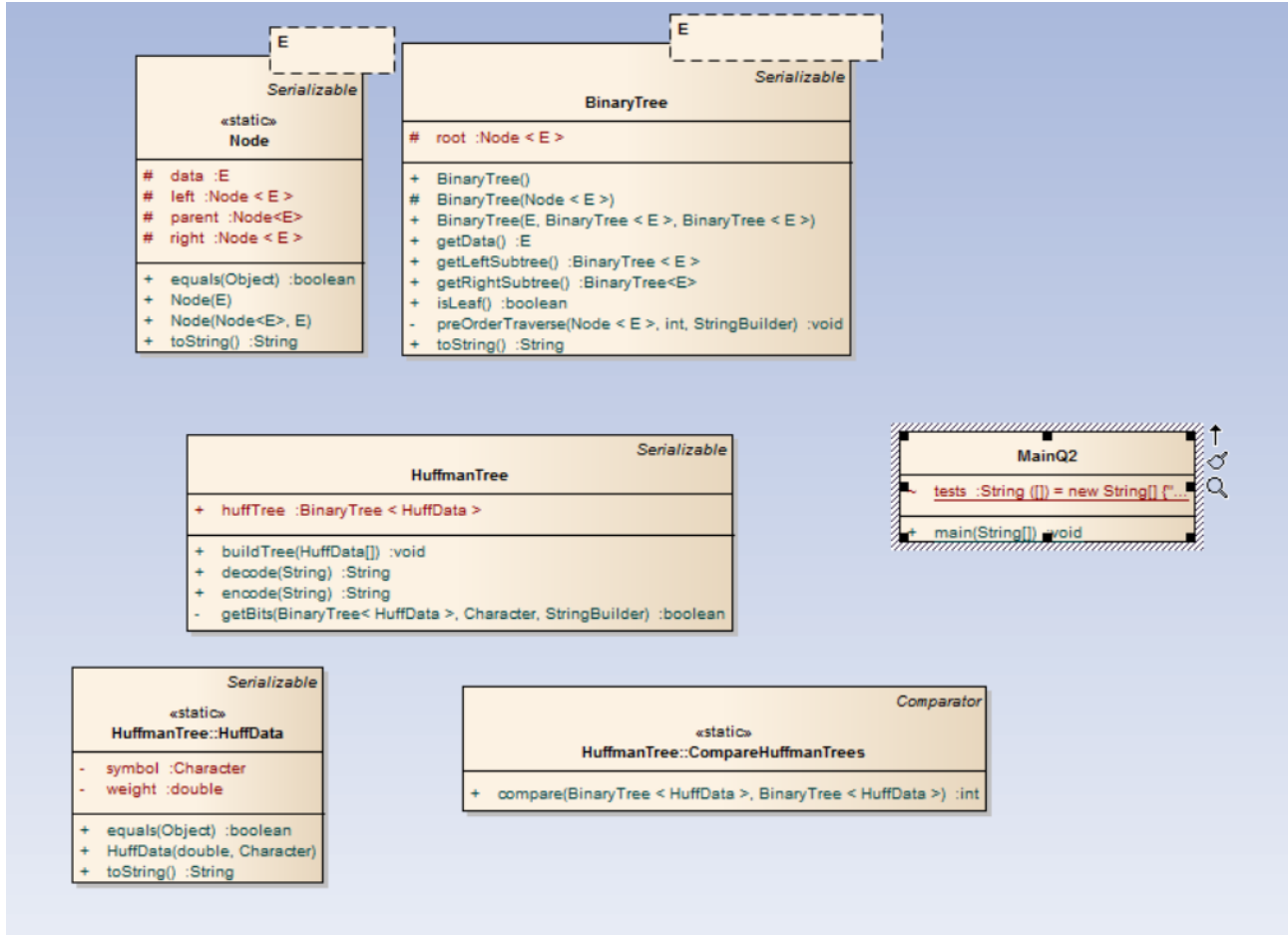
passengers.txt	
1	A 100
2	B 50
3	C 77
4	D 81
5	E 87
6	F 10
7	G 45
8	H 63
9	I 24
10	J 55
11	K 78

Program çalıştığında ise işleme alınma sırası şu şekildedir :



Q2

1. Class Diagrams

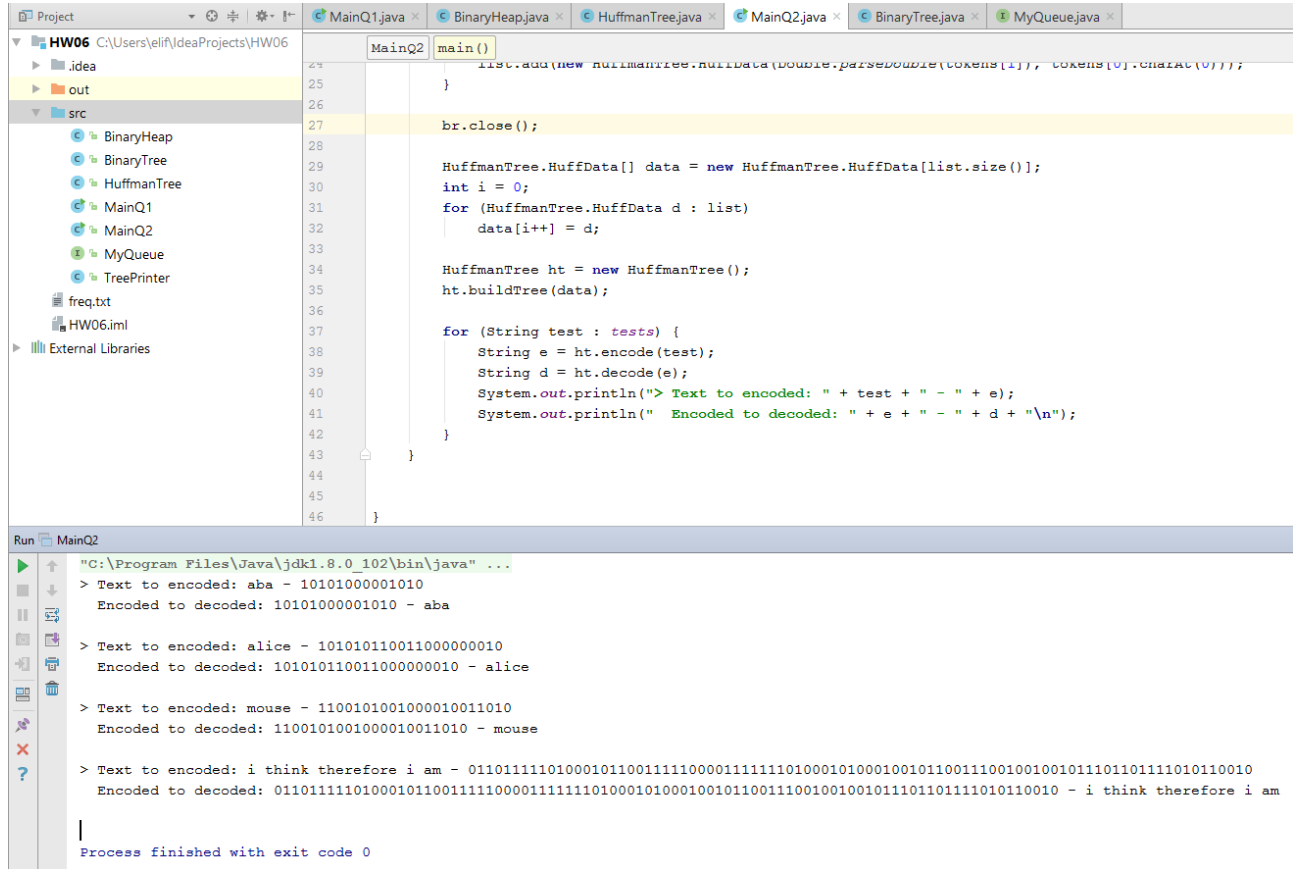


2. Problem Solutions Approach

Kitapta yer alan huffmanTree kodlarından yararlanıp eklemeler ve değişiklikler yaptım ve encode metodunu ekledim. Encode ederken verilen karakter tree içerisinde recursive olarak aranır. Eğer bir recursive çağrı sonrasında bulunabiliyor ise o recursive çağrı için geçerli olan karakter string e eklenir (sol ağaç için 0 ve sağ ağaç için 1) . En son ise oluşan string in tersi alınarak encode edilmiş olur. Her bir harf için bu işlem tekrarlanıp çıkan string ler birleştirilir.

3. Test Cases

Test ederken ilk başta verdiğim string i şifreledim. Daha sonra o şifrelenmiş ifadeyi decode fonksiyonuma verdim ve tekrar string imi elde ettim. Freq.txt yi oluştururken kitapta yer alan ingilizce için hazırlanmış tabloyu esas aldım.



The screenshot displays an IDE with a project named 'HW06'. The 'src' folder contains several files: BinaryHeap, BinaryTree, HuffmanTree, MainQ1, MainQ2, MyQueue, and TreePrinter. The 'MainQ2.java' file is open, showing the 'main()' method. The code reads a file 'freq.txt', processes its contents, and tests the HuffmanTree encoding and decoding functionality. The output window shows the results of these tests for various input strings.

```
24      list.add(new HuffmanTree.HuffData(Double.parseDouble(tokens[1]), tokens[0].charAt(0)));
25  }
26
27  br.close();
28
29  HuffmanTree.HuffData[] data = new HuffmanTree.HuffData[list.size()];
30  int i = 0;
31  for (HuffmanTree.HuffData d : list)
32      data[i++] = d;
33
34  HuffmanTree ht = new HuffmanTree();
35  ht.buildTree(data);
36
37  for (String test : tests) {
38      String e = ht.encode(test);
39      String d = ht.decode(e);
40      System.out.println("> Text to encoded: " + test + " - " + e);
41      System.out.println("    Encoded to decoded: " + e + " - " + d + "\n");
42  }
43  }
44
45
46  }
```

Run MainQ2

```
"C:\Program Files\Java\jdk1.8.0_102\bin\java" ...
> Text to encoded: aba - 10101000001010
    Encoded to decoded: 10101000001010 - aba

> Text to encoded: alice - 101010110011000000010
    Encoded to decoded: 101010110011000000010 - alice

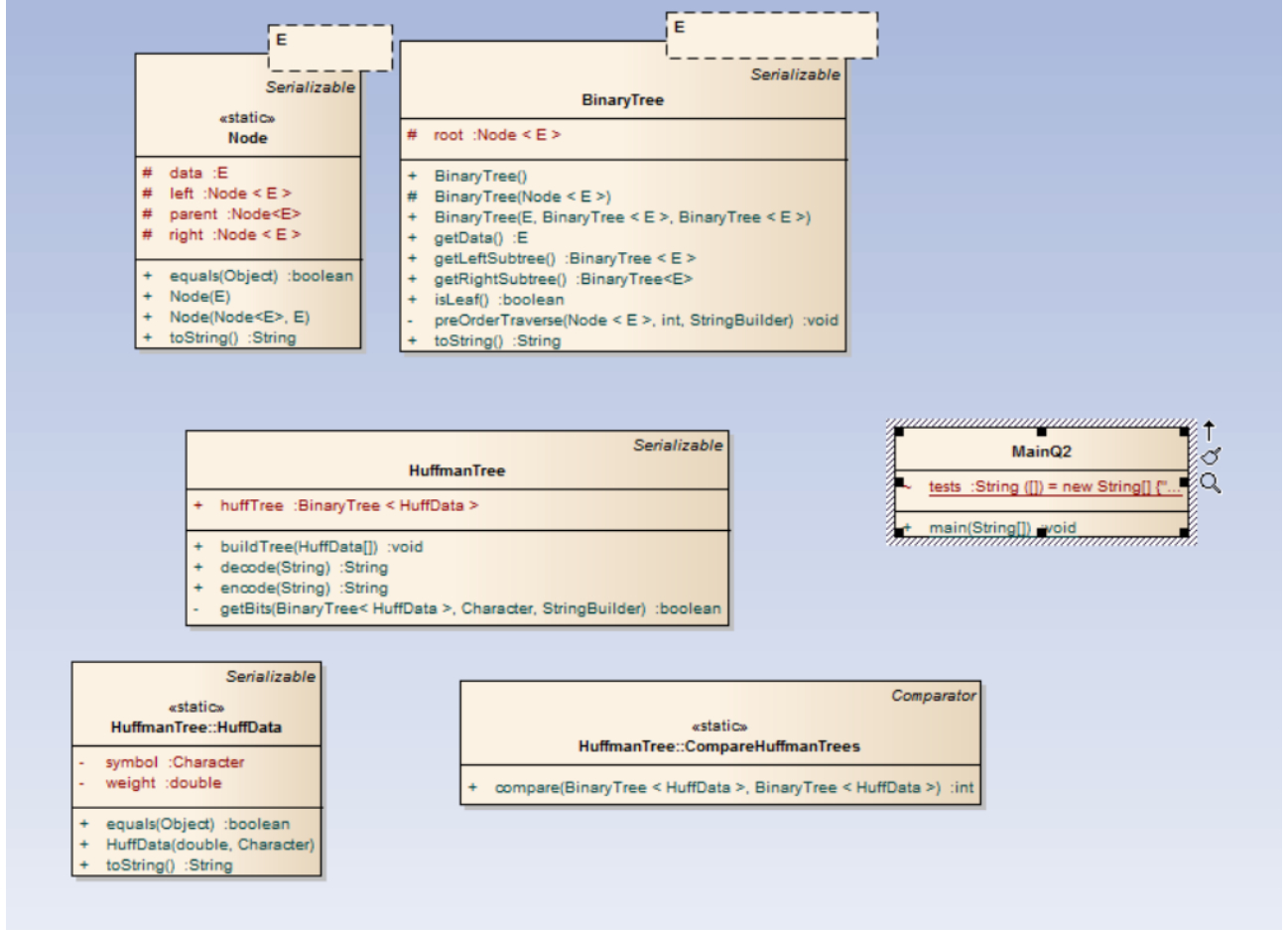
> Text to encoded: mouse - 1100101001000010011010
    Encoded to decoded: 1100101001000010011010 - mouse

> Text to encoded: i think therefore i am - 011011111010001011001111100001111110100010100010010011001100100101110101111010110010
    Encoded to decoded: 011011111010001011001111100001111110100010100010010011001100100101110101111010110010 - i think therefore i am

Process finished with exit code 0
```

Q3

1. Class Diagrams



2. Problem Solutions Approach

İteratörümün level order şekilde traversal yapabilmesi için queue veri yapısını kullandım. Bu yapıyı kullanmamın nedeni FIFO prensibine göre çalışmasıydı. İteratörün `next()` metodunu yazarken öncelikle constructor da root node umu queue ya ekledim. Daha sonra `next()` metodunda queue mdan öncelikle bir eleman çıkardım. Bu çıkardığım elemanın sağ ve sol çocukları varsa onları sırayla queue ya ekledim. Next() metodunu her çağırdığımda bu işlemi tekrarlayarak ağacın tamamını level level queya ekleyerek FIFO prensibi ile yaptım.

Ağacın daha güzel görünmesi için kullandığım `treePrinter` class ını

<http://stackoverflow.com/questions/4965335/how-to-print-binary-tree-diagram> sayfasından aldım.

3. Test Cases

The screenshot displays an IDE with a project named 'HW06'. The file explorer on the left shows the project structure, including a 'src' directory with files like 'BinaryHeap', 'BinaryTree', 'FamilyTree', 'HuffmanTree', 'MainQ1', 'MainQ2', 'MainQ3', and 'MyQueue'. The main editor shows a Java file 'MainQ3.java' with a list of names: Hasan, Ayşe, Hasan, ebu-Ayşe, Ali, Ayşe, ibn-Hasan, Sema, Hasan, ebu-Ayşe, Merve, Ali, ibn-Ayşe, and Beyza, Hasan, ebu-Ayşe. The console output shows the level order traversal of a binary tree: Hasan, Ayşe, Ali, Sema, Merve, Beyza. The process finished with exit code 0.

```

graph TD
    Hasan --> Ayşe
    Hasan --> Sema
    Ayşe --> Ali
    Ayşe --> Merve
    Sema --> Beyza
  
```

Level order iterator for family tree

Hasan
Ayşe
Ali
Sema
Merve
Beyza

Process finished with exit code 0