Gebze Technical University
Computer Engineering

CSE 222
2017 Spring

HOMEWORK 5 REPORT

FÜRKAN YILDIZ
141044031

Course Assistant:NUR BANU ALBAYRAK

Problem Solutions Approach

1.1)
Öncelikle Binary Tree'yi oluşturmak için kitaptan yardım alındı. Binary tree'yi oluşturuktan sonra oluşturulan bu classı Itarable interface'sinden implement edildi. Daha sonra ise Binary Tree için Iterator interfacesini implement eden bir iterator classı oluşturuldu. Bu iterator class'ın methodlarını override etmeye geldiğinde sıra, bu aşamada bizden Binary tree'yi pre-order olarak gezmemiz isteniyor, yani aşamalar şu şekilde olmalı : 1)root – 2) left tree – 3)right tree. Bunu sağayabilmek için Binary Tree'nin iterator class'ının içerisinde private bir Stack tutuldu ve bu stack'e eğer root null değil ise root eklendi. Next metodunda ise stack'in en üstündeki değer (root) çıkartıldı. Ve temp bir Node'ye atıldı, daha sonra bu temp'in eğer sağ çocuğu varsa önce sağ çocuğu sonra ise sol çocuğu varsa sol çocuğu eklendi (sağ çocuk önce eklendi çünkü en son sağ da işlem yapılacak) daha sonra ise çıkarılmış ve temp'e atılmış root return edildi. Böylece ilk next methodumuz bizim rootumuzu return etti. Daha next'i çağıracağımızda ise root'un sol çocuğu üstte olacak o çıkartılacak çıkartılan node'nin sağ sol çocukları sırasıyla eklenecak stack'a ve çıkartılan node return edilecek böylece left tree'de return edilmiş oldu, daha sonra ise left tree tamamen bitirildiğinde right tree stacktan çıkartılacak ve return edilecek böylece right tree'yide traverse etmiş olacağız. Bu operasyonlar ile tree'miz üzerinde sırasıyla root,left tree ve right tree traversesini yapabilmiş olduk.

1.2)
Öncelikle Binary Search Tree'yi oluşturmak için kitaptan yardım alındı. Bu tree oluşturulurken Comparable, BinaryTree classlarını extend, kendimizin yazdığı SearchTree interfacesini ise implement edildi. Daha sonra Level order olarak traverse yapabilmek için LevelOrderIteratorClass iterator class'ı yazıldı Binary Search Tree'de. LevelOrderIteratorClass 'ın içerisinde private bir queue ve bir iterator tutuldu. Bizden oluşan ağacın her deep'ine (level) göre gezen bir iterator yazmamız isteniyor. Bunu sağlayabilmek için Binary Search Tree'nin iterator classında private bir queue tuttuldu. Iterator oluşturulduğunda bu queue'ye root eğer null değilse ekleniyor. Null ise eklenmiyor ve queue boş oluyor.

Next methodunda ise eklenmiş olan root'un lefti boş değil ise lefti queue'ye ekleniyor daha sonra right'ı boş değil ise right'ı queue 'ye ekleniyor ve daha sonra queue'nin root'u çıkartılarak return ediliyor. Böylece next ilk elemanı return ediyor ve bir sonraki next'i çağırmada root, bir önceki root'un left'i oluyor, onun sol ve sağ çocukları var ise queue'ye ekleniyor ve ardından çıkartılıyor böylece ilk root'un left'i return ediliyor. Bir sonraki aşamada ise queue'nin ilk elemanı olan 1. Root'un sağ çocuğu root oluyor ve yine sağ ve sol çocukları var ise queue'ye ekleniyor ve 1. Rootun sağ çocuğu return ediliyor. Daha sonraki aşamalarda ise üst rootun, leftinin sol çocukları daha sonra üsteki rootun leftinin sağ çocukları root oluyor.İşlem bu şekilde queue boş oluncaya kadar devam ediyor. Her next'e remove'de yapığımız için queue 'den hasNext methodunda next'in olup olmadığını anlamak için queue'nin isEmpty methodu kullanılıyor eğer queue boş ise tree de eleman kalmamış olacaktır. Bu şekilde level order olarak iterator gezdirilebiliniyor.

2)

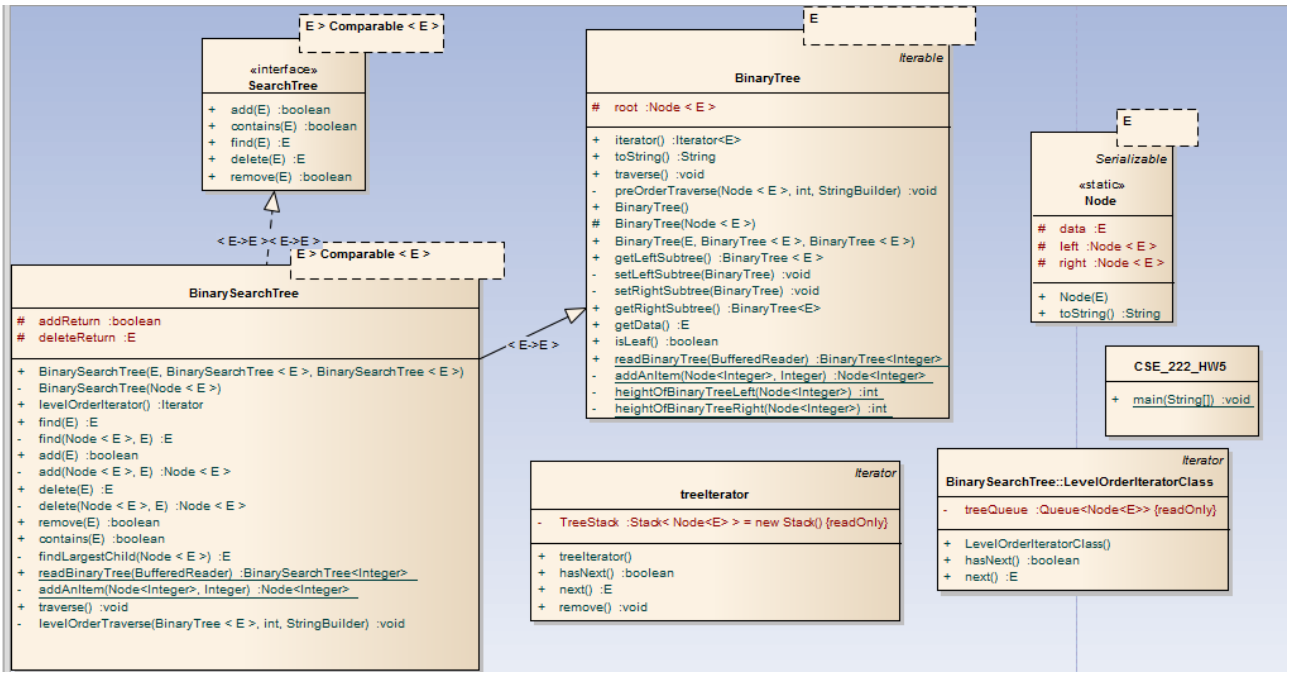
FamilyTree oluşturmak için BinaryTree classının String instancesi extend edildi. Constructurede gelen string parametresi direk node çevirilerek eklendi. Böylece ağacın rootu yapılmış oldu. Daha sonraki elemanları ekleyebilmek için ise add metodu yazıldı. Bu metodda gelen nickname stringi, "ibn" ya da "ebu" olarak ve karşısına gelen isim olmak üzere iki parçaya ayırılıyor. Böylece "ibn" ve "ebu" ya göre farklı işlemler yapılıyor. Eğer okunan line "Ayşe, Hasan, ebu-Ayşe" şeklinde nicknamede ve isimde bir eşitlik içeriyorsa, "addForEquality" metoduna root, parent, ve eklenecek data yollanıyor ve burada recursive kullanılarak parent aranıyor, bulunan tüm aynı dataya sahip parentlere verilen string node olarak ekleniyor. (ekleme işlemi bulunan parentin lefti boşsa leftine, boş değil ise left'inin right'ının ilk boş olduğu noda yapılıyor) ve her eklemeden sonra count arttırılıyor count eğer 1 den büyük ise exception fırlatılıyor (birden fazla aynı nickname – parent koşulunu sağlayan ikili olduğu için)

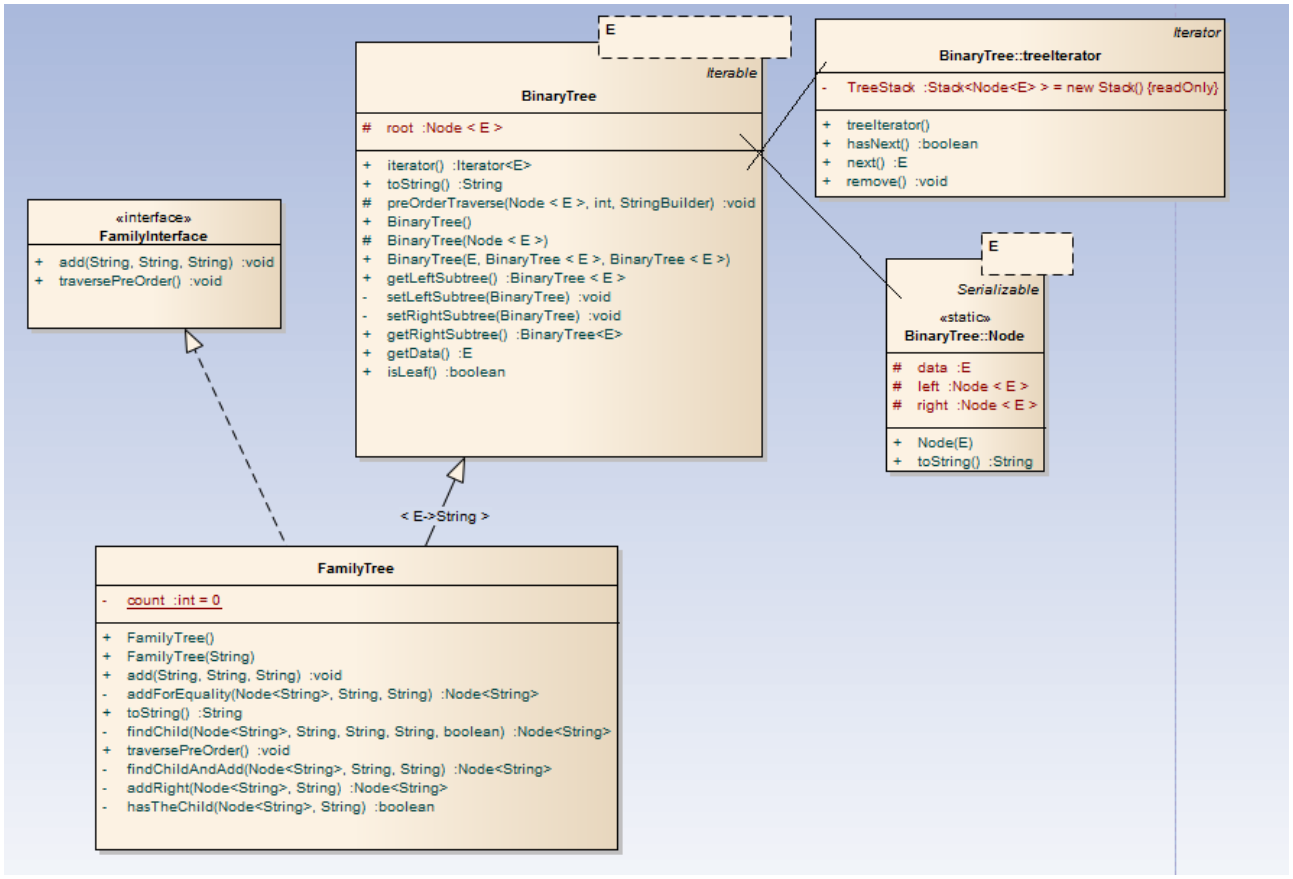
Eğer böyle bir durum yok ise ve nickname "ibn" ise nickname önce ibn'den sonra gelen isim treede aranacak o bulunduktan sonra onun, parent stringinde verilen bir çocuğu olup olmadığı kontrol edilecek eğer var ve verilen parentin boşsa leftine lefti boş değilse ilk boş olan leftinin rightına ekleniyor. (bu ekleme işlemi yapılırken count tutuluyor, eğer count 1 den fazla gelir ise exception fırlatılıyor)

Eğer "ebu" ise nickname, önce parentten gelen isim treede aranıyor daha nicknameden gelen isim aranıyor ve birbirleri ile parentlik ilişkisi var ise verilen parentin boşsa leftine lefti boş değilse ilk boş olan leftinin rightına ekleniyor. (bu ekleme işlemi yapılırken count tutuluyor, eğer count 1 den fazla gelir ise exception fırlatılıyor)

Node'ler üzerinde işlemler yapılırken recursive metotlar kullanıldı çünkü iterative kullanılsaydılar methodun içerisinde değişim yapacaklardı, main root bundan etkilenmeyecekti.

Class Diagrams




E
Iterable
BinaryTree
root :Node < E >
+ iterator() :Iterator<E>
+ toString() :String
# preOrderTraverse(Node < E >, int, StringBuilder) :void
+ BinaryTree()
# BinaryTree(Node < E >)
+ BinaryTree(E, BinaryTree < E >, BinaryTree < E >)
+ getLeftSubtree() :BinaryTree < E >
- setLeftSubtree(BinaryTree) :void
- setRightSubtree(BinaryTree) :void
+ getRightSubtree() :BinaryTree<E>
+ getData() :E
+ isLeaf() :boolean
Iterator
BinaryTree::treeIterator
- TreeStack :Stack<Node<E> > = new Stack() {readOnly}
+ treeIterator()
+ hasNext() :boolean
+ next() :E
+ remove() :void
«interface»
FamilyInterface
+ add(String, String, String) :void
+ traversePreOrder() :void
E
Serializable
«static»
BinaryTree::Node
data :E
left :Node < E >
right :Node < E >
+ Node(E)
+ toString() :String
< E->String >
FamilyTree
- count :int = 0
+ FamilyTree()
+ FamilyTree(String)
+ add(String, String, String) :void
- addForEquality(Node<String>, String, String) :Node<String>
+ toString() :String
- findChild(Node<String>, String, String, String, boolean) :Node<String>
+ traversePreOrder() :void
- findChildAndAdd(Node<String>, String, String) :Node<String>
- addRight(Node<String>, String) :Node<String>
- hasTheChild(Node<String>, String) :boolean