**CSE222\_HW05**

**ERCAN UCA 091044011**

* **LinkedListRecursive** Classımda
* İnner **Node** Classı
* Node tipinde head,
* Method olarak private recursive methodlar ve onları çağıran Warapper methodlar vardır.

\***Private** recursive methodlar;

-> private int size(Node<E> head)

-> private void add(Node<E> head, E newValue)

-> private boolean remove(Node<E> head, Node<E> prev, E outData)

-> private String toString(Node<E> head)

\* **Public** Wrapper methodlar;

-> public int size() // linkedList size

-> public void add(E newValue) // Linked list'e ekleme

->public boolean remove(E outData) // linkedlisten silinmek istenen tüm değerleri siler.

-> public String toString() // Sonuçlar gösterir.

**\* Tüm elemanları silmek için her elemen silişimden sonra sonraki kısmı yolladım eğer sona gelindiyese return true yani son silmeden sonra true döner.**

* **ArrayListRecursive** Classımda;
* ArrayList tiplerinde list1 ve list2 vardır.
* ArrayList constructorları mevcuttur(Default ve iki list parametreli olarak.)
* **public List<E> IntersectionOfLists();**
* Class içinde olan list1 ve list2'yi kullanarak, private recursive methodu çağırır. Ve iki listede ortak olan elemanları bir liste olarak geri döndüren Wrapper methoddur.
* Listelerden biri boş ise **NULL** döndürür.
* Listelerden biri boş değilse; yeni liste oluşturup; list1, list2 ile sizelarını parametre olarak recursive method çağırılır.
* Çağırılma şekli;
* ArrayList<Integer> list1 = new ArrayList<>();
* Collections.sort(list1); // sorted
* ArrayList<Integer> list2 = new ArrayList<>();
* Collections.sort(list2); // sorted
* ArrayListRecursive<Integer> arr = new ArrayListRecursive<>(list1,list2);
* System.out.println(arr.IntersectonOfLists().toString());
  + - **Tüm elemanlara bakma işlemi için recursive method kullanıldı.**
    - **Bu methodla tüm elemanlara bakmak için listelerin sizeları 1er 1er düşürülerek tüm elemenlar sağlandı ve eğer sizelardan biri 0 olmamışsa size yeniden atayıp tüm elemanlar bulunup returnedList'e eklendi.**
* **private List<E> IntersectionOfList(List<E> list1,List<E>**

**list2,List<E> returnedList, int size1, int size2);**

* Recursive private method return Intersection of two lists.
* **@param** list1 The first list object.
* **@param** list2 The second list object.
* **@param** returnedList The result list.
* **@param** size1 The size of list1.
* **@param** size2 The size of list2.
* **@return** result list that intersection of list1 and list2.
* Recursive method çalışma şekli,
  + İlk önce BaseCaselere bakılır tek elemanlılar mı,
  + Sonra değilse; gelen size parametreleri ile son elemanın index ile **equal** method ile kıyas yapılır.
  + Ona göre Wrapper method da oluşturulmuş listeye ekleme yapılır, sonraki kısım için sizelar değiştirilip yeniden dönderilir method.
  + **Try-catch** blokları ile exception yakalanır. **Add,get** ile gelebilecek herhangi bir exception
* **private List<E> UnionOfLists(List<E> list1,List<E> list2,**

**List<E> returnedList,**

**int size1, int size2);**

Recursive private method return Union of two lists.

**@param** list1 The first list object.

**@param** list2 The second list object.

**@param** returnedList The result list.

**@param** size1 The size of list1.

**@param** size2 The size of list2.

**@return** result list that union of list1 and list2.

Recursive method çalışma şekli,

İlk önce BaseCaselere bakılır tek elemanlılar mı,

Sonra değilse; gelen size parametreleri ile son elemanın index ile **equal** method ile kıyas yapılır.

Ona göre Wrapper method da oluşturulmuş listeye ekleme yapılır, sonraki kısım için sizelar değiştirilip yeniden gönderilir method.

**Try-catch** blokları ile exception yakalanır. Add,get ile gelebilecek herhangi bir exception.

* **public List<E> UnionOfLists();**
  + - Class içinde olan **list1** ve **list2**'yi kullanarak, **private** recursive methodu çağırır. Ve iki listede elemanlardan birleşim kümesi olan bir liste olarak geri döndüren Wrapper methoddur.
    - Listelerden biri boş ise **NULL** döndürür.
    - Listelerden biri boş değilse; yeni liste oluşturup, **list1**, **list2** ile sizelarını parametre olarak recursive method çağırılır.
    - Çağırılma şekli;
      * ArrayList<Integer> list1 = new ArrayList<>();
      * Collections.sort(list1); // sorted
      * ArrayList<Integer> list2 = new ArrayList<>();
      * Collections.sort(list2); // sorted
      * ArrayListRecursive<Integer> arr = new ArrayListRecursive<>(list1,list2);
      * System.out.println(arr.UnionOfLists().toString());
    - **Tüm elemanlara bakma işlemi için recursive method kullanıldı.**

**Bu methodla tüm elemanlara bakmak için listelerin sizeları 1er 1er düşürülerek tüm elemenlar sağlandı ve eğer sizelardan biri 0 olmamışsa size yeniden atayıp tüm elemanlar bulunup returnedList'e eklendi.**

* **private boolean isSubSet(List<E> list1,List<E> list2,**

**int size1, int size2)**

* + - Recursive **private** method return boolean if **list2** is subset of **list1**
    - **@param** list1 The first list object.
    - **@param** list2 The second list object.
    - **@param** returnedList The result list.
    - **@param** size1 The size of list1.
    - **@param** size2 The size of list2.
    - **@return** true if list2 is subset of list1,otherwise false.
    - Recursive method çalışma şekli,
      * İlk önce BaseCaselere bakılır tek elemanlılar mı,
      * Sonra değilse; gelen size parametreleri ile son elemanın index ile **equal** method ile kıyas yapılır.
      * Ona göre sonraki kısım için sizelar değiştirilip yeniden gönderilir method.
      * **Try-catch** blokları ile exception yakalanır. Add,get ile gelebilecek herhangi bir exception.
* **public boolean isSubSet()**
  + - Class içinde olan **list1** ve **list2**'yi kullanarak, private recursive methodu çağırır. Ve iki liste içinde bir diğer listenin içindeki bir listenin kısmı mı, eğer alt listesi ise **TRUE**, değilse **FALSE** döndüren Wrapper methoddur.
    - Listelerden biri boş ise **NULL** döndürür.
    - Listelerden biri boş değilse; **list1**,**list2** ile sizelarını parametre olarak recursive method çağırılır.
    - Çağırılma şekli;
      * ArrayList<Integer> list1 = new ArrayList<>();
      * Collections.sort(list1); // sorted
      * ArrayList<Integer> list2 = new ArrayList<>();
      * Collections.sort(list2); // sorted
      * ArrayListRecursive<Integer> arr = new ArrayListRecursive<>(list1,list2);
      * System.out.println(arr.isSubSet()); // print TRUE or FALSE.
    - **Tüm elemanlara bakma işlemi için recursive method kullanıldı.**

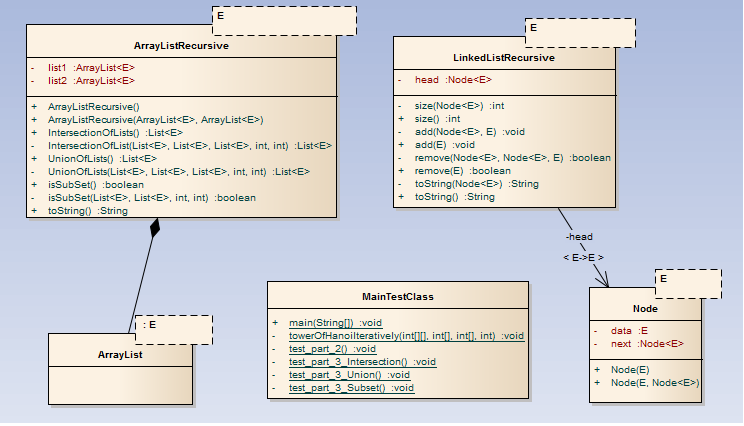
**Bu methodla tüm elemanlara bakmak için listelerin sizeları 1er 1er düşürülerek tüm elemenlar sağlandı ve eğer sizelardan biri 0 olmamışsa size yeniden atayıp tüm elemanlar bulunup returnedList'e eklendi.**

* **private static void towerOfHanoiIteratively(int pegs[][], int pegA[],**

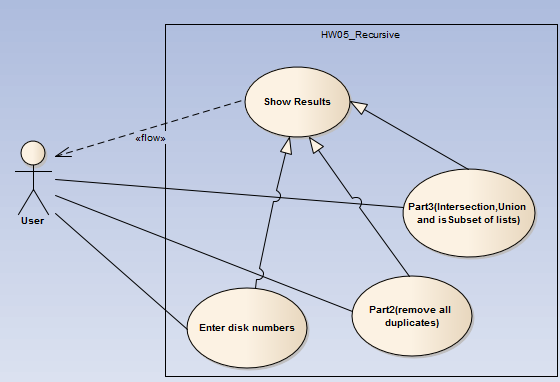
**int pegB[], int numberOfDisk);**

* + Tower of hanoi implement iterative.
  + **@param** pegs store weight and disk number.
  + **@param** pegA store swaps for pegA.
  + **@param** pegB store sawps for pegB.
  + **@param** numberOfDisk total disks number.
  + **Pegs[][]** ile tüm disklerimiz ve onların büyüklükleri tutulur.
  + Başlangıçta ağırlıklar verilen ile ağırıklar belirlenir, yani disklerin büyüklükleri.
  + Ve Hareket etme konumları disk sayısının çift ya da tek olmasına göre belirlenir.
  + En son disk kalmayana dek döngü içinde hareketler gösterilip ve en son kaç ms sürmüş hesap tutulur.

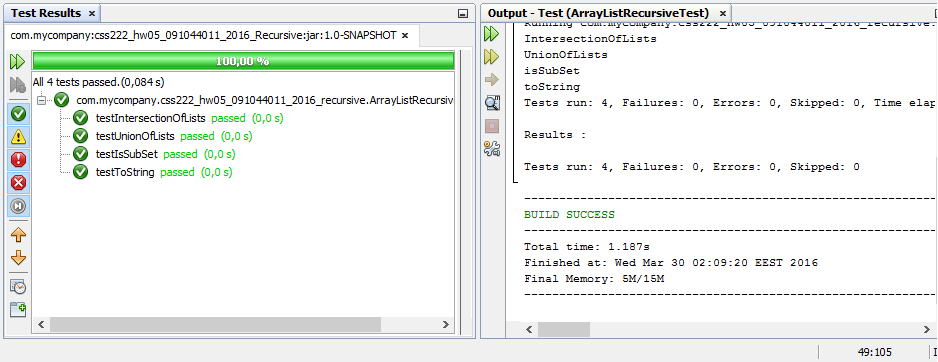
**Class Diagramı**

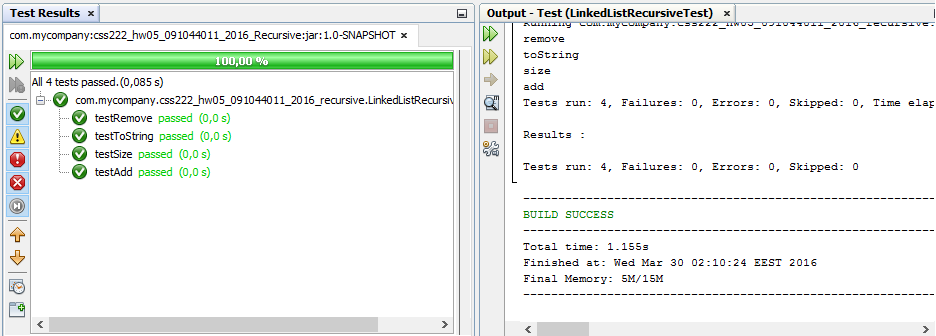


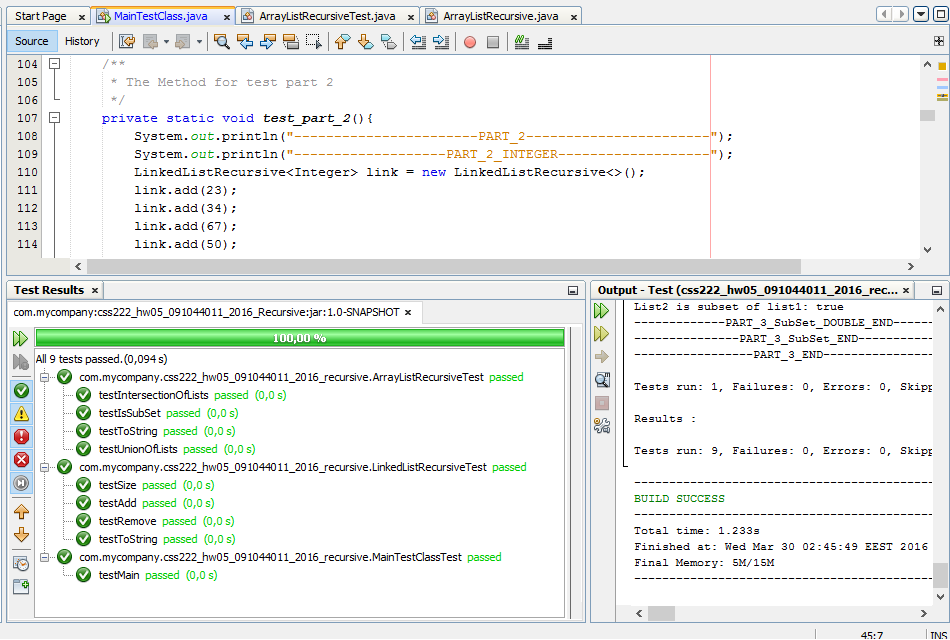
**Use Case Diagrams**



**Tests**







**Main tests**

**-----------------------PART\_1-----------------------**

pegs[0][0] = 1

pegs[1][0] = 99

pegs[2][0] = 99

pegs[0][1] = 2

pegs[1][1] = 99

pegs[2][1] = 99

pegs[0][2] = 3

pegs[1][2] = 99

pegs[2][2] = 99

Moving disk 1 from peg 0 to peg 2

Moving disk 2 from peg 0 to peg 1

Moving disk 1 from peg 2 to peg 1

Moving disk 3 from peg 0 to peg 2

Moving disk 1 from peg 1 to peg 0

Moving disk 2 from peg 1 to peg 2

Moving disk 1 from peg 0 to peg 2

Took approximately 31 ms to solve Towers of Hanoi with 3 disks

Number of moves: 7

---------------------PART\_1\_END---------------------

-----------------------PART\_2-----------------------

-------------------PART\_2\_INTEGER-------------------

Before removing!

23

34

67

50

67

67

67

67

67

67

67

67

22

67

67

11

67

67

Remove all dublicates.

After removing!

23

34

50

22

11

-----------------PART\_2\_INTEGER\_END-----------------

-------------------PART\_2\_STRING-------------------

Before removing!

ercan

can

sercan

arif

elif

elif

serdar

elif

Ayse

elif

elif

Halis

Yusuf

After removing!

ercan

can

sercan

arif

serdar

Ayse

Halis

Yusuf

-----------------PART\_2\_STRING\_END-----------------

-------------------PART\_2\_DOUBLE-------------------

Before removing!

12.1

12.3

12.3

12.7

12.3

12.5

12.4

After removing!

12.1

12.7

12.5

12.4

-----------------PART\_2\_DOUBLE\_END-----------------

---------------------PART\_2\_END---------------------

-----------------------PART\_3-----------------------

-----------------PART\_3\_Intersection----------------

-------------PART\_3\_Intersection\_Integer------------

ArrayListRecursive{list1=[1, 2, 3, 5], list2=[0, 1, 2, 3, 4, 8, 88]}

[3, 2, 1]

---------PART\_3\_Intersection\_Integer\_END------------

---------PART\_3\_Intersection\_STRING-----------------

ArrayListRecursive{list1=[Ali, Can, elif, ercan], list2=[Ali, Can, ahmet, elif, ercan, kenan, sertap]}

[ercan, elif, Can, Ali]

---------PART\_3\_Intersection\_String\_END------------

---------PART\_3\_Intersection\_DOUBLE-----------------

ArrayListRecursive{list1=[1.5, 1.6, 1.9, 2.2], list2=[0.1213, 1.5, 1.6, 1.9, 4.223, 33.4, 99.99]}

[1.9, 1.6, 1.5]

---------PART\_3\_Intersection\_DOUBLE\_END------------

---------PART\_3\_Intersection\_END------------------

------------------PART\_3\_Union\_List-----------------

--------------PART\_3\_Union\_List\_Integer-------------

ArrayListRecursive{list1=[1, 2, 3, 5], list2=[0, 1, 2, 3, 4, 8, 88]}

[5, 88, 3, 8, 2, 4, 1]

---------PART\_3\_Union\_List\_Integer\_END--------------

---------PART\_3\_Union\_List\_STRING-----------------

ArrayListRecursive{list1=[Ali, Can, elif, ercan], list2=[Ali, Can, ahmet, elif, ercan, kenan, sertap]}

[ercan, sertap, elif, kenan, Can, Ali]

---------PART\_3\_Union\_List\_String\_END------------

---------PART\_3\_Union\_List\_DOUBLE-----------------

ArrayListRecursive{list1=[1.5, 1.6, 1.9, 2.2], list2=[0.1213, 1.5, 1.6, 1.9, 4.223, 33.4, 99.99]}

[2.2, 99.99, 1.9, 33.4, 1.6, 4.223, 1.5]

---------PART\_3\_Union\_List\_DOUBLE\_END---------------

---------PART\_3\_Union\_List\_END----------------------

-----------------PART\_3\_SubSet----------------------

-------------PART\_3\_SubSet\_Integer------------------

ArrayListRecursive{list1=[1, 2, 3, 5], list2=[0, 1, 2, 3, 4, 8, 88]}

List2 is subset of list1: true

---------PART\_3\_SubSet\_Integer\_END------------------

---------PART\_3\_SubSet\_STRING-----------------------

ArrayListRecursive{list1=[Ali, Can, elif, ercan], list2=[Ali, Can, ahmet, elif, ercan, kenan, sertap]}

List2 is subset of list1: true

---------PART\_3\_SubSet\_String\_END------------

---------PART\_3\_SubSet\_DOUBLE-----------------

List2 is subset of list1: true

-------------PART\_3\_SubSet\_DOUBLE\_END---------------

---------------PART\_3\_SubSet\_END--------------------

-----------------PART\_3\_END-------------------------

Ödev githup linki

<https://github.com/erccanuca/cse222_hw05_TowerOfHanoi_LinkedListRecursive_ArrayListRecursive.git>

(Ödev teslim süresi geçince public yapılacak.)

**(Herhangi bir ödevde 2 günlük ek süremi bu ödevde kullandım.**

**Son saatlerde pc çöktü neyseki githupa eklemiştim.)**