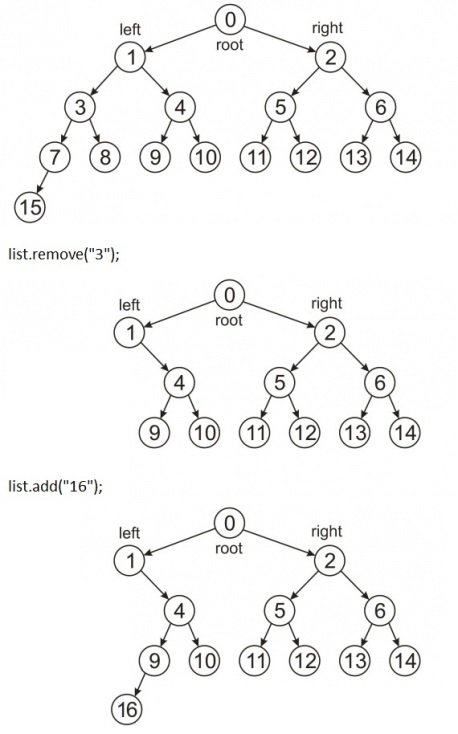
**Построй дерево(1)**

Амиго, похоже ты уже достаточно окреп. Самое время проверить свои навыки в большой задаче!

Сегодня реализуем свое дерево немного нестандартным способом(на базе **AbstractList**).

Вводную информацию можешь получить используя свой любимый поисковик и текст ниже.

Элементы дерева должны следовать так как показано на картинке:  


Для начала сделаем наше дерево потомком класса **AbstractList** с параметром типа String, а также реализуем интерфейсы **Cloneable** и **Serializable**.

Реализацию методов get(int index) и size() пока оставь стандартной.

**Построй дерево(2)**

Несмотря на то что наше дерево является потомком класса **AbstractList**, это не список в привычном понимании.  
В частности нам недоступны принимающие в качестве параметра индекс элемента.  
Такие методы необходимо переопределить и бросить новое исключение типа **UnsupportedOperationException**.

**Вот их список:**  
public String get(int **index**)  
public String set(int **index**, String **element**)  
public void add(int **index**, String **element**)  
public String remove(int **index**)  
public List<String> subList(int **fromIndex**, int **toIndex**)  
protected void removeRange(int **fromIndex**, int **toIndex**)  
public boolean addAll(int **index**, Collection<? extends String> **c**)

**Требования:**  
1. При попытке вызвать метод get(int index) должно возникать исключение типа UnsupportedOperationException.  
2. При попытке вызвать метод set(int index, String element) должно возникать исключение типа UnsupportedOperationException.  
3. При попытке вызвать метод add(int index, String element) должно возникать исключение типа UnsupportedOperationException.  
4. При попытке вызвать метод String remove(int index) должно возникать исключение типа UnsupportedOperationException.  
5. При попытке вызвать метод subList(int fromIndex, int toIndex) должно возникать исключение типа UnsupportedOperationException.  
6. При попытке вызвать метод removeRange(int fromIndex, int toIndex) должно возникать исключение типа UnsupportedOperationException.  
7. При попытке вызвать метод addAll(int index, Collection c) должно возникать исключение типа UnsupportedOperationException.

### Построй дерево(3)

Класс описывающий дерево мы создали, теперь нужен класс описывающий тип элементов дерева:  
1.  В классе **CustomTree** создай вложенный статический параметризированный класс Entry<T> с модификатором доступа по умолчанию.  
2. Обеспечь поддержку этим классом интерфейса **Serializable**.  
3. Создай такие поля (модификатор доступа по умолчанию):  
- String elementName;  
- int lineNumber;  
- boolean availableToAddLeftChildren, availableToAddRightChildren;  
- Entry<T> parent, leftChild, rightChild;  
4. Реализуй публичный конструктор с одним параметром типа String:  
- установи поле elementName равным полученному параметру;  
- установи поле availableToAddLeftChildren равным **true**;  
- установи поле availableToAddRightChildren равным **true**;  
5. Реализуй метод void checkChildren, устанавливающий поле availableToAddLeftChildren в **false**, если leftChild не равен **null** и availableToAddRightChildren в **false**, если rightChild не равен **null**.  
6. Реализуй метод boolean isAvailableToAddChildren, возвращающий дизъюнкцию полей availableToAddLeftChildren и availableToAddRightChildren.

Любое дерево начинается с корня, поэтому не забудь в класс **CustomTree** добавить поле root типа Entry<String> c модификатором доступа по умолчанию.

**Требования:**  
1. Класс CustomTree.Entry должен быть объявлен с модификатором доступа по умолчанию.  
2. Класс CustomTree.Entry должен поддерживать интерфейс Serializable.  
3. В классе CustomTree.Entry должно существовать поле elementName типа String.  
4. В классе CustomTree.Entry должно существовать поле availableToAddLeftChildren типа boolean.  
5. В классе CustomTree.Entry должно существовать поле availableToAddRightChildren типа boolean.  
6. В классе CustomTree.Entry должно существовать поле parent типа Entry.  
7. В классе CustomTree.Entry должно существовать поле leftChild типа Entry.  
8. В классе CustomTree.Entry должно существовать поле rightChild типа Entry.  
9. В классе CustomTree.Entry должен быть корректно реализован конструктор с одним параметром типа String (смотри условие).  
10. В классе CustomTree.Entry должен корректно реализован метод isAvailableToAddChildren (смотри условие).

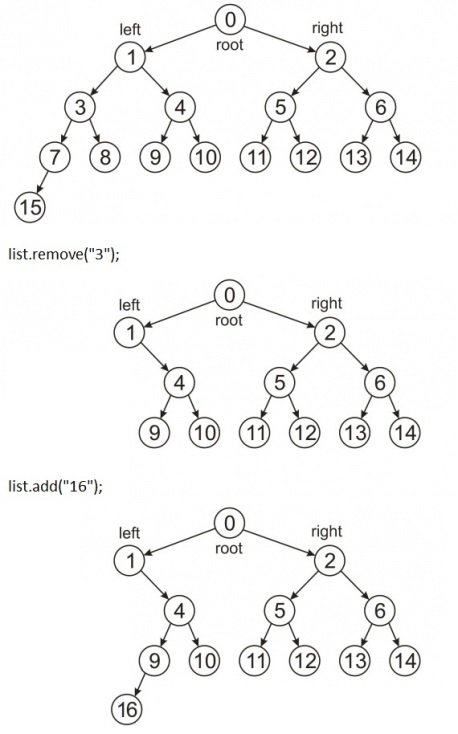
### Построй дерево(4)

Любое дерево начинается с корня, поэтому не забудь в класс **CustomTree** добавить поле root типа **Entry<String>** c модификатором доступа по умолчанию.

Инициируй его в конструкторе **CustomTree**, имя (elementName) не важно.

Итак, основа дерева создана, пора тебе поработать немного самому.

Вспомним как должно выглядеть наше дерево.

**Элементы дерева должны следовать так как показано на картинке:**  


Необходимо написать методы, которые бы позволили создать такую структуру дерева и проводить операции над ней.

Тебе необходимо:  
1) переопределить метод add(String s) - добавляет элементы дерева, в качестве параметра принимает имя элемента (elementName), искать место для вставки начинаем слева направо.  
2) переопределить метод size() - возвращает текущее количество элементов в дереве.  
3) реализовать метод getParent(String s) - возвращает имя родителя элемента дерева, имя которого было полученного в качестве параметра.

Если необходимо, можешь вводить дополнительные методы и поля, не указанные в задании.

**Требования:**  
1. В классе CustomTree должно существовать поле root типа Entry.  
2. В классе CustomTree должны быть переопределены методы add(String s) и size().  
3. После добавления N элементов в дерево с помощью метода add, метод size должен возвращать N.  
4. Метод getParent должен возвращать имя родителя для любого элемента дерева.

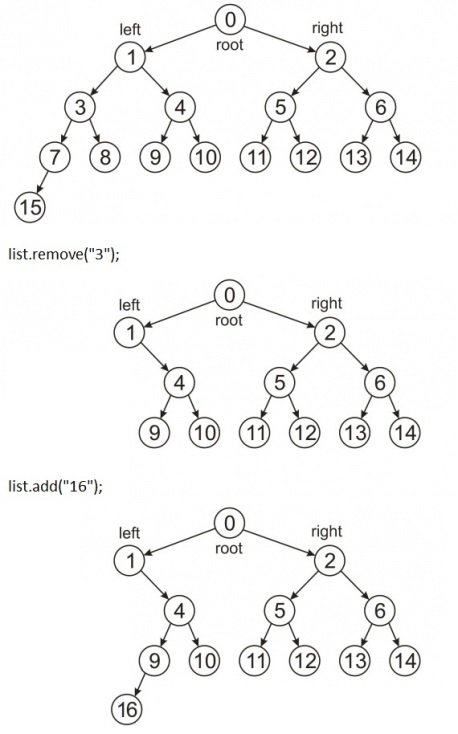
### Построй дерево(5)

Добавлять в дерево элементы мы можем, теперь займись удалением:

Необходимо реализовать метод remove(Object o), который будет удалять элемент дерева имя которого было полученного в качестве параметра.

Если переданный объект не является строкой, метод должен бросить **UnsupportedOperationException**.

Если в дереве присутствует несколько элементов с переданным именем - можешь удалить только первый найденный.

Не забывай сверять поведение своего дерева с картинкой:  


Что будет если удалить из дерева элементы "**3**", "**4**", "**5**" и "**6**", а затем попытаемся добавить новый елемент?

В таком случае элементы "**1**" и "2" должны восстановить возможность иметь потомков (возможно придется внести изменения в метод add()).

**Требования:**  
1. После удаления последнего добавленного элемента из дерева с помощью метода remove, метод size должен возвращать N-1.  
2. После удаления второго элемента добавленного в дерево, метод size должен возвращать N/2 + 1 (для случаев где N > 2 и является степенью двойки), N - размер дерева до удаления.  
3. Если переданный объект не является строкой, метод remove() должен бросить UnsupportedOperationException.  
4. Если ни один элемент не способен иметь потомков, необходимо восстановить такую возможность.