Druid

**Для работы с деревьями и создания**

**псевдографического изображения дерева.**

**Преимущества**

1. Положение узла в дереве определяется только одним параметром: Address.
2. В таблицу, содержащую дерево, нельзя записать больше одного корня.

*Как правило, узел в дереве определяется двумя параметрами: Id и ParentId. Для корня ParentId == null, следовательно, столбец "ParentId" нужно задавать со свойством NULL. null != null, поэтому в таблице может оказаться несколько корней. Address узла, не являющегося корнем, содержит Address его родителя, поэтому столбец "ParentId" не требуется.*

1. Класс Node играет роль базового класса для ваших специфичных классов и содержит все необходимые свойства и методы (Add, Search, Sort, Save, Load и др.), поэтому он простой и эффективный.
2. Метод ToString возвращает псевдографический рисунок дерева. Это очень удобно для быстрой работы с деревьями в консольном приложении. Чтобы увидеть дерево, просто воспользуйтесь Console.WriteLine(<Node>).

**class Node**

Узел дерева. Узел правильный, если его Address правильный.

1. **Свойства**
   1. **List<int> Address { get; set; }**

Адрес узла в дереве. Последовательность (...,n2,n1,n0), где n – Number. n0 принадлежит этому узлу, остальные элементы принадлежат его предкам: n1 – родителю, n2 – прародителю и т.д., первый элемент – первопредку. Если последовательность содержит один элемент, то соответствующий узел не имеет предков. Address должен быть уникален в пределах дерева. Address правильный, если описывает положение узла в дереве. На Address может ссылаться только один узел. "set" нужен для того, чтобы создавать локальные деревья. После обработки локальные деревья вписываются в нужное место других деревьев.

* 1. **string AddressAsString { get; set; }**

Address в виде "(...,n2,n1,n0)".

* 1. **List<Node> Children { get; }**

Дочерние узлы этого узла.

* 1. **List<Node> DescendantCount { get; }**

Количество потомков этого узла.

* 1. **bool HasIncorrectDescendant { get; }**

Результат проверки наличия неправильного потомка у этого узла.

* 1. **int InnerLevelCount { get; }**

Количество непустых уровней с потомками этого узла. Потомки этого узла должны быть правильными.

* 1. **bool IsCorrect { get; }**

Результат проверки наличия неправильного потомка у этого узла.

* 1. **int Level { get; }**

Уровень иерархии дерева, на котором находится этот узел. Количество предков этого узла, служащее общим признаком узлов, объединяющим их в поколение. Узлы на уровне L являются следующим поколением по отношению к узлам на уровне L-1.

* 1. **virtual string Name { get; set; }**

Имя этого узла. Level дерева может содержать только узлы с уникальными именами. Имена, в отличие от Address, нельзя задавать любые, т.к. невозможно создать метод автоматического исправления имен.

* 1. **int Number { get; }**

Номер этого узла на его Level. Номера на уровне дерева должны быть уникальными, чтобы адреса в этом дереве были уникальными.

* 1. **Node Parent { get; }**

Ссылка на родитель этого узла.

1. **Конструкторы**
   1. ()
   2. (string name)
   3. (IEnumerable<int> address)
   4. (IEnumerable<Node> children)
   5. (string name, string address)
   6. (string name, IEnumerable<int> address)
   7. (string name, IEnumerable<Node> children)
   8. (IEnumerable<int> address, IEnumerable<Node> children)
   9. (string name, string address, IEnumerable<Node> children)
   10. (string name, IEnumerable<int> address, IEnumerable<Node> children)
2. **Делегаты**
   1. string **Titile**(Node node)

**summary**

Возвращает подпись узла в псевдографическом изображении дерева. Аргумент ToString().

**node**

Ресурс подписи. Аргумент лямбда-выражения, передающегося в делегат.

**returns**

Подпись узла в псевдографическом изображении дерева.

1. **Методы**
   1. Node **Add**(Node newChild)

**summary**

Добавляет новый узел в Children этого узла.

**newChild**

Узел, добавляемый в Children этого узла.

**returns**

Этот узел с новым узлом в Children.

* 1. Node **AddChildren**(Node[] newChildren)

**summary**

Добавляет новые узлы в Children этого узла.

**newChild**

Узлы, добавляемые в Children этого узла.

**returns**

Этот узел с новыми узлами в Children.

* 1. Node **AddChildren**(List<Node> newChildren)

**summary**

Добавляет новые узлы в Children этого узла.

**newChild**

Узлы, добавляемые в Children этого узла.

**returns**

Этот узел с новыми узлами в Children.

* 1. Node **AddChildrenCopy**(Node[] newChildren)

**summary**

Добавляет новые узлы в Children этого узла.

**newChild**

Узлы, копии которых добавляются в Children этого узла.

**returns**

Этот узел с новыми узлами в Children.

* 1. Node **AddChildrenCopy**(List<Node> newChildren)

**summary**

Добавляет новые узлы в Children этого узла.

**newChild**

Узлы, копии которых добавляются в Children этого узла.

**returns**

Этот узел с новыми узлами в Children.

* 1. Node AddCopy(Node newChild)

**summary**

Добавляет новый узел в Children этого узла.

**newChild**

Узел, копия которого добавляется в Children этого узла.

**returns**

Этот узел с новым узлом в Children.

* 1. Node **Copy**()

**summary**

Возвращает копию этого узла.

**returns**

Копия этого узла.

* 1. Node **DescendantsCorrect**()

**summary**

Делает потомки этого узла правильными.

**returns**

Этот узел с правильными потомками.

* 1. Node **DescendantCount**()

**summary**

Возвращает количество потомков этого узла.

**returns**

Количество потомков этого узла.

* 1. Node **Find**(int[] address)

**summary**

Возвращает узел с указанным Address из дерева, корнем которого является этот узел.

**address**

Address искомого узла.

**returns**

Искомый узел.

* 1. Node **Find**(List<int> address)

**summary**

Возвращает узел с указанным Address из дерева, корнем которого является этот узел.

**address**

Address искомого узла.

**returns**

Искомый узел.

* 1. List<Node> **Find**(bool isCorrect)

**summary**

Возвращает узлы с указанным IsCorrect из дерева, корнем которого является этот узел.

**isCorrect**

IsCorrect искомых узлов.

**returns**

Искомые узлы.

* 1. List<Node> **Find**(string name)

**summary**

Возвращает узлы с указанным Name из дерева, корнем которого является этот узел.

**name**

Name искомых узлов.

**returns**

Искомые узлы.

* 1. List<Node> **Find**(int number)

**summary**

Возвращает узлы с указанным Number из дерева, корнем которого является этот узел.

**number**

Number искомых узлов.

**returns**

Искомые узлы.

* 1. bool **HasIncorrectDescendant**()

**summary**

Проверяет наличие неправильного потомка этого узла.

**returns**

Результат проверки наличия неправильного потомка этого узла.

* 1. int **InnerLevelCount**()

**summary**

Возвращает количество непустых уровней потомков этого узла. Потомки этого узла должны быть правильными.

**returns**

Количество непустых уровней потомков этого узла.

* 1. Node **SaveAsCsv**(string path)

**summary**

Сохраняет дерево, корнем которого является этот узел, как таблицу со столбцами "Name", "Address", "Parent Addres", разделенными символом ";".

**path**

Путь к файлу, в который будет сохранено дерево.

**returns**

Этот узел.

* 1. List<Node> **Search**(string nameSubstring)

**summary**

Возвращает узлы из дерева, корнем которого является этот узел, Name которых содержит указанную подстроку.

**nameSubstring**

Подстрока в Name искомых узлов.

**returns**

Искомые узлы.

* 1. List<Node> **Search**(string[] nameSubstrings)

**summary**

Возвращает узлы из дерева, корнем которого является этот узел, Name которых содержит хотя бы одну из указанных подстрок.

**nameSubstrings**

Подстроки в Name искомых узлов.

**returns**

Искомые узлы.

* 1. List<Node> **Search**(List<string> nameSubstrings)

**summary**

Возвращает узлы из дерева, корнем которого является этот узел, Name которых содержит хотя бы одну из указанных подстрок.

**nameSubstrings**

Подстроки в Name искомых узлов.

**returns**

Искомые узлы.

* 1. Node **SetChildren**(Node[] newChildren)

**Summary**

Задает новый Children этому узлу.

**newChildren**

Узлы, которые составят новый Children этого узла.

**returns**

Этот узел с новым Children.

* 1. Node **SetChildren**(List<Node> newChildren)

**summary**

Задает новый Children этому узлу.

**newChildren**

Узлы, которые составят новый Children этого узла.

**returns**

Этот узел с новым Children.

* 1. Node **SetChildrenCopy**(Node[] newChildren)

**summary**

Задает новый Children этому узлу.

**newChildren**

Узлы, копии которых составят новый Children этого узла.

**returns**

Этот узел с новым Children.

* 1. Node **SetChildrenCopy**(List<Node> newChildren)

**summary**

Задает новый Children этому узлу.

**newChildren**

Узлы, копии которых составят новый Children этого узла.

**returns**

Этот узел с новым Children.

* 1. Node **Sort**(IComparer<Node> comparer)

**summary**

Сортирует потомки этого узла на их уровнях согласно указанному компаратору.

**comparer**

Компаратор, описывающий механизм сравнения узлов.

**returns**

Этот узел с отсортированными потомками на их уровнях.

* 1. Node **Sort**(SortingSign sign)

**summary**

Сортирует потомки этого узла на их уровнях согласно указанному признаку сравнения узлов.

**sign**

Признак сравнения узлов.

**returns**

Этот узел с отсортированными потомками на их уровнях.

* 1. override string **ToString**()

**summary**

Возвращает строку с псевдографическим изображением дерева, корнем которого является этот узел. Изображение состоит из соединительных линий от предков к потомкам и подписей узлов. Узлы подписаны соответствующими именами.

**returns**

Строка с псевдографическим изображением дерева.

* 1. string **ToString**(Title title)

**summary**

Возвращает строку с псевдографическим изображением дерева, корнем которого является этот узел. Изображение состоит из соединительных линий от предков к потомкам и подписей узлов. Подпись узла создается делегатом Title.

**title**

Подпись узла в псевдографическим изображением дерева.

**returns**

Строка с псевдографическим изображением дерева.

Загружает дерево из указанного файла CSV в этот узел, который будет являться его корнем. Предполагается, что дерево в файле имеет правильные потомки.

Сохраняет дерево, корнем которого является этот узел, как таблицу со столбцами "Name", "Address", разделенными символом ";", в файл CSV. Этих столбцов достаточно, чтобы описать узел, т.к. Address узла содержит Address его родителя. Потомки этого узла должны быть правильными.