using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Text;

namespace CategoryTreeStructure

{

public class TreeNode<T>

{

readonly List<TreeNode<T>> \_children = new List<TreeNode<T>>();

public TreeNode(T data)

{

Data = data;

}

public T Data { get; set; }

public TreeNode<T> Parent { get; private set; }

public ReadOnlyCollection<TreeNode<T>> Children

{

get { return \_children.AsReadOnly(); }

}

public void AddChild(TreeNode<T> value)

{

value.Parent = this ;

\_children.Add(value);

}

}

}

================================================================================

using System;

using System.Collections.Generic;

using System.Text;

namespace CategoryTreeStructure

{

public class Category

{

public int ID { get; set; }

public string Name { get; set; }

public string Keywords { get; set; }

}

}

================================================================================

using System;

using System.Collections.Generic;

using System.Text;

namespace CategoryTreeStructure

{

public class CategoryNode : TreeNode<Category>

{

public CategoryNode(Category data) : base(data)

{

}

}

}

================================================================================

using System;

using System.Collections.Generic;

using System.Text;

namespace CategoryTreeStructure

{

public class CategoryTree : TreeNode<CategoryNode>

{

public CategoryTree(CategoryNode data) : base(data)

{

Root = data;

}

public CategoryNode Root { get; set; }

public CategoryNode FindByID(int id)

{

CategoryNode retObj = null;

Queue<CategoryNode> q = new Queue<CategoryNode>();

q.Enqueue(this.Root);

while (q.Count > 0)

{

var current = q.Dequeue();

if (current.Data.ID == id)

{

retObj = current;

retObj.Data.Keywords = GetKeywords(current);

break;

}

foreach (CategoryNode children in current.Children)

{

q.Enqueue(children);

}

}

return retObj;

}

private string GetKeywords(CategoryNode node)

{

if (!string.IsNullOrEmpty(node.Data.Keywords))

{

return node.Data.Keywords;

}

else

{

return this.GetKeywords(node.Parent as CategoryNode);

}

}

public int[] GetCategoryIDAtLevel(int level)

{

var node\_level = 0;

List<int> retCategories = new List<int>();

Queue<CategoryNode> q = new Queue<CategoryNode>();

q.Enqueue(this.Root);

q.Enqueue(new CategoryNode(new Category { ID = -1000}));

while (q.Count > 0)

{

var node = q.Dequeue();

if (node.Data.ID == -1000)

{

if (node\_level == level)

{

break;

}

else

{

node\_level++;

q.Enqueue(node);

}

}

else

{

if (node\_level == level)

{

retCategories.Add(node.Data.ID);

}

foreach (CategoryNode children in node.Children)

{

q.Enqueue(children);

}

}

}

return retCategories.ToArray();

}

}

}

================================================================================

// TESTS

using CategoryTreeStructure;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

namespace UnitTestProject1

{

[TestClass]

public class CategoriesUnitTest

{

CategoryTree \_categoryTree;

CategoryNode \_root;

ReadOnlyCollection<TreeNode<Category>> \_secondLevelChildren;

public CategoriesUnitTest()

{

\_root = new CategoryNode(new Category { ID = -1, Name = "", Keywords = "" });

var cat1\_1 = new CategoryNode(new Category { ID = 100, Name = "Business", Keywords = "Money" });

var cat1\_2 = new CategoryNode(new Category { ID = 200, Name = "Tutoring", Keywords = "Teaching" });

var cat2\_1\_1 = new CategoryNode(new Category { ID = 101, Name = "Accounting", Keywords = "Taxes" });

var cat2\_1\_2 = new CategoryNode(new Category { ID = 102, Name = "Taxation" });

var cat3\_1\_1 = new CategoryNode(new Category { ID = 103, Name = "Corporate Tax" });

var cat3\_1\_2 = new CategoryNode(new Category { ID = 109, Name = "Small Business Tax" });

var cat2\_2\_1 = new CategoryNode(new Category { ID = 201, Name = "Computer" });

var cat3\_2\_1 = new CategoryNode(new Category { ID = 202, Name = "Operating System" });

cat2\_1\_1.AddChild(cat3\_1\_1);

cat2\_1\_1.AddChild(cat3\_1\_2);

cat1\_1.AddChild(cat2\_1\_1);

cat1\_1.AddChild(cat2\_1\_2);

cat2\_2\_1.AddChild(cat3\_2\_1);

cat1\_2.AddChild(cat2\_2\_1);

\_root.AddChild(cat1\_1);

\_root.AddChild(cat1\_2);

\_categoryTree = new CategoryTree(\_root);

\_secondLevelChildren = \_categoryTree.Root.Children;

}

[TestMethod]

public void RootNodeShouldBeThere()

{

var root = new Category { ID = -1, Name = "", Keywords = "" };

Assert.AreEqual(\_root.Data.ID, root.ID);

}

[TestMethod]

public void RootCanAdd1stLevelCategories()

{

Assert.AreEqual(\_secondLevelChildren.Count, 2);

}

[TestMethod]

public void RootCanAdd2ndLevelCategories()

{

var \_3\_1\_children = \_secondLevelChildren[0].Children;

var \_3\_2\_children = \_secondLevelChildren[1].Children;

Assert.AreEqual(\_3\_1\_children.Count, 2);

Assert.AreEqual(\_3\_2\_children.Count, 1);

}

[TestMethod]

public void RootCanAdd3rdLevelCategories()

{

var \_3\_1\_children = \_secondLevelChildren[0].Children;

var \_3\_2\_children = \_secondLevelChildren[1].Children;

var \_4\_1\_1\_children = \_3\_1\_children[0].Children;

var \_4\_1\_2\_children = \_3\_1\_children[1].Children;

var \_4\_2\_1\_children = \_3\_2\_children[0].Children;

Assert.AreEqual(\_4\_1\_1\_children.Count, 2);

Assert.AreEqual(\_4\_1\_2\_children.Count, 0);

Assert.AreEqual(\_4\_2\_1\_children.Count, 1);

}

[TestMethod]

public void FindByIDShouldFetchExistingCategory()

{

var category\_201 = \_categoryTree.FindByID(201);

Assert.IsNotNull(category\_201);

Assert.AreEqual(category\_201.Data.ID, 201);

Assert.AreEqual(category\_201.Data.Keywords, "Teaching");

var category\_202 = \_categoryTree.FindByID(202);

Assert.IsNotNull(category\_202);

Assert.AreEqual(category\_202.Data.ID, 202);

Assert.AreEqual(category\_202.Data.Keywords, "Teaching");

}

[TestMethod]

public void GetCategoryIDAtLevelShouldReturnCategories()

{

Array cats\_2 = \_categoryTree.GetCategoryIDAtLevel(2);

Assert.AreEqual(cats\_2.Length, 3);

Array expected\_2 = new[] { 101, 102, 201 };

CollectionAssert.AreEqual(cats\_2, expected\_2);

Array cats\_3 = \_categoryTree.GetCategoryIDAtLevel(3);

Assert.AreEqual(cats\_3.Length, 3);

Array expected\_3 = new[] { 103, 109, 202 };

CollectionAssert.AreEqual(cats\_3, expected\_3);

}

}

}