using System;

namespace Probleme

{

class Arbre\_N\_aire

{

#region Attributs

protected Noeud\_N racine;

#endregion

#region Constructeurs

public Noeud\_N Racine

{

get { return racine; }

set { racine = value; }

}

public Arbre\_N\_aire(Noeud\_N racine)

{

this.racine = racine;

}

public Arbre\_N\_aire(Salarie val)

{

this.racine = new Noeud\_N(val);

}

#endregion

#region Insérer frère

public bool InsererFrere(Noeud\_N aine, Noeud\_N nouv)

{

if (aine == racine)

{

return false; //c'est une façon de protéger la racine qui ne doit pas avoir de frère

}

if (aine == null)

{

return false;

}

if (aine.Frere == null)

{

aine.Frere = nouv;

return true;

}

return InsererFrere(aine.Frere, nouv);

}

public bool InsererFrere(Noeud\_N aine, Salarie val)

{

if (aine == racine)

{

return false; //c'est une façon de protéger la racine qui ne doit pas avoir de frère

}

if (aine == null)

{

return false;

}

if (aine.Frere == null)

{

aine.Frere = new Noeud\_N(val);

return true;

}

return InsererFrere(aine.Frere, val);

}

#endregion

#region Insérer successeur

public bool InsererSuccesseur(Noeud\_N parent, Noeud\_N nouv)

{

if (parent == null)

{

return false;

}

if (parent.Successeur! != null)

{

return false;

}

parent.Successeur = nouv;

return true;

}

public bool InsererSuccesseur(Noeud\_N parent, Salarie val)

{

if (parent == null)

{

return false;

}

if (parent.Successeur! != null)

{

return false;

}

parent.Successeur = new Noeud\_N(val);

return true;

}

#endregion

#region AFFICHAGE ARBORESCENCE

public void AfficheEspace(int res) // Permet de faire res \* tabulation pour l'affichage

{

for (int i = 0; i < res; i++)

{

Console.Write("\t");

}

}

public void AffichageArborescence(Noeud\_N n, int res) // Ce code permet d'afficher l'arborescence de l'arbre en respectant bien la hiérarchie

{

if (n.Salarie == null) // Si le salarié est null, il n'y a rien à afficher

{

Console.Write("");

}

else // sinon

{

Console.Write("\n");

AfficheEspace(res); // On affiche l'espace par rapport au directeur d'entreprise qui est le plus haut gradé

if (res != 0)

{

Console.Write("||=> ");

}

Console.Write(n.Salarie.Nom.ToUpper() + " / " + n.Salarie.Poste); // Affiche le nom et le poste du salarié pour l'arbre

}

if (n.Successeur != null) // Si le successeur n'est pas null

{

AfficheEspace(res + 1); // Ajoute des tabulations

AffichageArborescence(n.Successeur, res + 1);

}

if (n.Frere != null) // Si le frère n'est pas null

{

AfficheEspace(res); // Ajoute des tabulations

AffichageArborescence(n.Frere, res);

}

}

#endregion

#region RECHERCHE SALARIE

public Noeud\_N RechercheSalarie(Noeud\_N racine, string nom)

{

if (racine != null)

{

if (racine.Salarie.Nom.ToUpper() == nom.ToUpper())

{

Console.Write(racine.ToString());

return racine;

}

RechercheSalarie(racine.Successeur, nom);

RechercheSalarie(racine.Frere, nom);

}

return null;

}

#endregion

#region INCLURE DANS ARBRE

public void AjoutArbre(Noeud\_N arbre, Salarie salarie, string nom) // Ce code permet d'ajouter un élément dans l'arbre

{

Noeud\_N parent = new Noeud\_N();

Noeud\_N salN = new Noeud\_N(salarie);

if (this.racine == null) // Vérifie si null ou non

{

arbre = salN; // Ajout du salarié

this.racine = arbre;

}

else

{

if (arbre != null) // Vérifie si null ou non

{

if (arbre.Salarie != null) // Vérifie si null ou non

{

if (arbre.Salarie.Nom.ToUpper() == nom.ToUpper()) // Si le nom du salarié de l'arbre est le même que celui qu'on souhaite supprimer

{

Console.Write(arbre.ToString());

parent = arbre; // On prend le noeud qui correspond

}

}

AjoutArbre(arbre.Successeur, salarie, nom); // Parcours l'arbre pour trouver l'employé

AjoutArbre(arbre.Frere, salarie, nom); // Parcours l'arbre pour trouver l'employé

}

while (parent.Frere != null) // On recherche le dernier frère null

{

parent = parent.Frere;

}

parent.Frere = salN; // On ajoute au dernier frere un nouveau frere qui est le salarié qu'on veut ajouter

}

}

#endregion

#region SUPPRESSION DANS ARBRE

public void SuppressionArbre(Noeud\_N arbre, string nom) // Ce code permet de supprimer un élément dans l'arbre

{

Noeud\_N parent = new Noeud\_N();

if (this.racine != null) // Vérifie si null ou non

{

if (arbre != null) // Vérifie si null ou non

{

if (arbre.Salarie != null) // Vérifie si null ou non

{

if (arbre.Salarie.Nom != null) // Vérifie si null ou non

{

if (arbre.Salarie.Nom.ToUpper() == nom.ToUpper()) // Si le nom du salarié de l'arbre est le même que celui qu'on souhaite supprimer

{

Console.WriteLine(arbre.ToString());

parent = arbre; // On prend le noeud qui correspond

}

}

}

SuppressionArbre(arbre.Successeur, nom); // Parcours l'arbre pour trouver l'employé

SuppressionArbre(arbre.Frere, nom); // Parcours l'arbre pour trouver l'employé

}

if (parent.Salarie != null) // Vérifie si null ou non

{

if (parent.Successeur == null && parent.Frere == null) // Si parent est une feuille de l'arbre

{

parent = null; // Mets à null

}

else if (parent.Successeur != null) // Sinon regarde si le successeur est null

{

parent.Salarie = parent.Successeur.Salarie; // Transforme parent en son successeur

parent.Successeur.Salarie = null; // Mets à null

}

else if (parent.Frere != null) // Sinon regarde si le frère est null

{

parent.Salarie = parent.Frere.Salarie; // Transforme parent en son frere

parent.Frere.Salarie = null; // Mets à null

}

}

}

}

#endregion

}

}