Développement d’un client REST, l’application Vélib

Description du thème

Propriétés	Description
Intitulé long	Application en C# qui utilise un service REST informant des disponibilités des stations de vélos –Vélib- de la ville de Paris.
Formation concernée	BTS Services informatiques aux organisations
Matière	SLAM4 - Réalisation et maintenance de composants logiciels
Présentation	Un service REST est présenté (gestion Vélib de la ville de Paris) ; une application en C# propose deux implémentations d’un client REST : synchrone et asynchrone.
Notions	- Programmation objet. - Développement C#. - XML. - Développement asynchrone.
Transversalité	GEOSI : les services WEB, XML comme langage d’échange entre applications.
Pré-requis	- Développement en C#.
Outils	Visual Studio (à partir de 2005).
Mots-clés	Service REST, C#, objet, XML, asynchrone.
Durée	
Auteur(es)	Patrice Grand avec la relecture attentive de Pierre Loisel.
Version	v 1.0
Date de publication	Février 2010

1. Présentation

L’application présentée permet de visualiser les disponibilités des vélos et des emplacements de parking à la disposition des parisiens dans le cadre du service Vélib géré par la ville de Paris.

Cette application développée en C# va récupérer un certain nombre d’informations en ligne fournies par la ville de Paris :

a) La liste des points d’accès Vélib et leurs localisations (1300 points)
b) La situation en temps réel de chaque point d’accès (disponibilités).

- La ressource à l’URI **http://www.velib.paris.fr/service/carto** retourne en XML les points d’accès.


```
  <?xml version="1.0" encoding="UTF-8" ?>
- <carto>
  - <markers>
      <marker name="20020 - PYRENEES RENOUVIER" number="20020" address="183 RUE DES PYRENEES -" fullAddress="183 RUE DES PYRENEES
        - 75020 PARIS" lat="48.861078931572" lng="2.4003620767512" open="1" bonus="1" />
      <marker name="20019 - DAVOUT SERPOLLET" number="20019" address="1 RUE SERPOLLET -" fullAddress="1 RUE SERPOLLET - 75020
        PARIS" lat="48.860597406723" lng="2.4095012861135" open="1" bonus="1" />
      <marker name="20017 - RUE SAINT BLAISE" number="20017" address="69 RUE SAINT BLAISE -" fullAddress="69 RUE SAINT BLAISE - 75020
        PARIS" lat="48.856813985451" lng="2.4090329301628" open="1" bonus="0" />
      <marker name="20016 - PYRENEES VITRUVE" number="20016" address="114 BIS RUE DES PYRENEES -" fullAddress="114 BIS RUE DES
        PYRENEES - 75020 PARIS" lat="48.856744663594" lng="2.4046199075916" open="1" bonus="0" />
```


 © CERTA - juin 2011 – v1.0

- Les URI de la *forme* ***http://www.velib.paris.fr/service/stationdetails/<number>*** fournissent les disponibilités des points d'accès.

Par exemple, **http://www.velib.paris.fr/service/stationdetails/20020** ***:***

```
<?xml version="1.0" encoding="UTF-8" ?>
- <station>
    <available>25</available>
    <free>1</free>
    <total>28</total>
    <ticket>1</ticket>
  </station>
```

Sur ce point d'accès, numéro 20020 (Pyrénnées Renouvier), 25 vélos sont disponibles, 1 emplacement est libre. Ce point dispose de 28 emplacements (2 emplacements semblent donc hors service). On peut payer par carte bleu : *ticket = 1*.

2. Service Web et technologie REST

Un service web présente l'interface de ses services dans un fichier de type WSDL. Consommer un service web consiste à pointer sur ce contrat et utiliser les fonctions proposées. Le protocole d'échange des services web est SOAP.

Dans notre cas, la ville de Paris ne propose pas de service web (pas de contrat WSDL) mais une architecture/technologie de type REST.

REST est l'acronyme de "Representational State Transfer" imaginée par Roy T. Fielding dans une thèse publiée en 2000.

Cette technologie est décrite en détail sur différents sites, en particulier dans une traduction d'un chapitre de la thèse : **http://opikanoba.org/tr/fielding/rest/**

Quelques points à souligner :

- Utilisation du seul protocole HTTP grâce auquel on récupère les ressources, directement à partir des URI.
- Ces ressources sont accessibles en XML, sans DTD ni schéma. D'autres formats de ressources sont possibles.
- Structuration des ressources : des URI signifiantes.

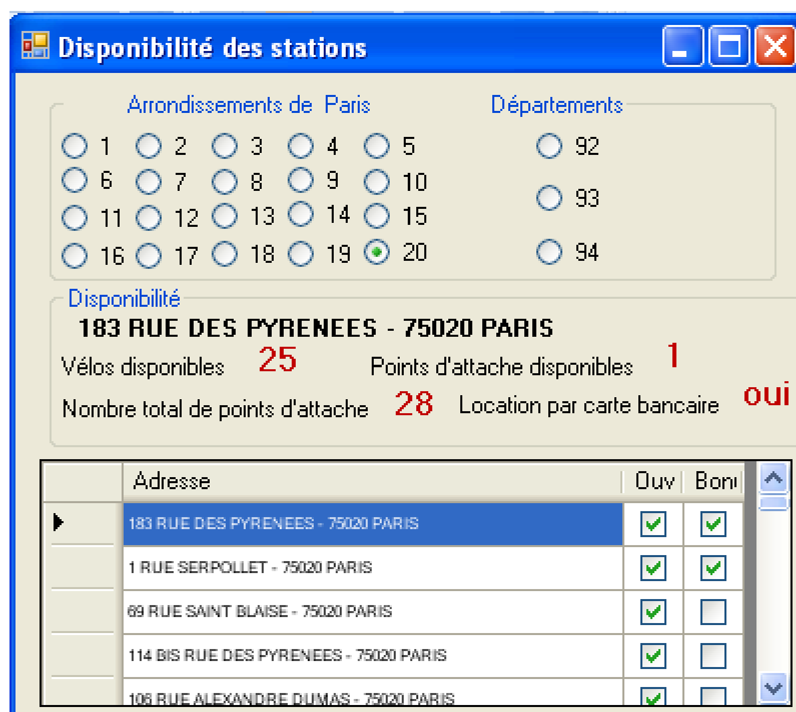
3. L'application à réaliser

L'application ne contient qu'un seul cas d'utilisation :

Cas d'utilisation : visualisation des disponibilités.

1. Le système présente la liste des arrondissements et départements.
2. L'utilisateur sélectionne un arrondissement (ou un département).
3. Le système retourne la liste des points d'accès de l'arrondissement (ou du département).
4. L'utilisateur sélectionne un point d'accès.
5. Le système retourne les disponibilités (vélos, points d'attache), les informations sur le nombre total de points d'attache, le type de location (carte bancaire ou non) ainsi que le rappel du lieu sélectionné.

L'interface peut prendre la forme suivante :

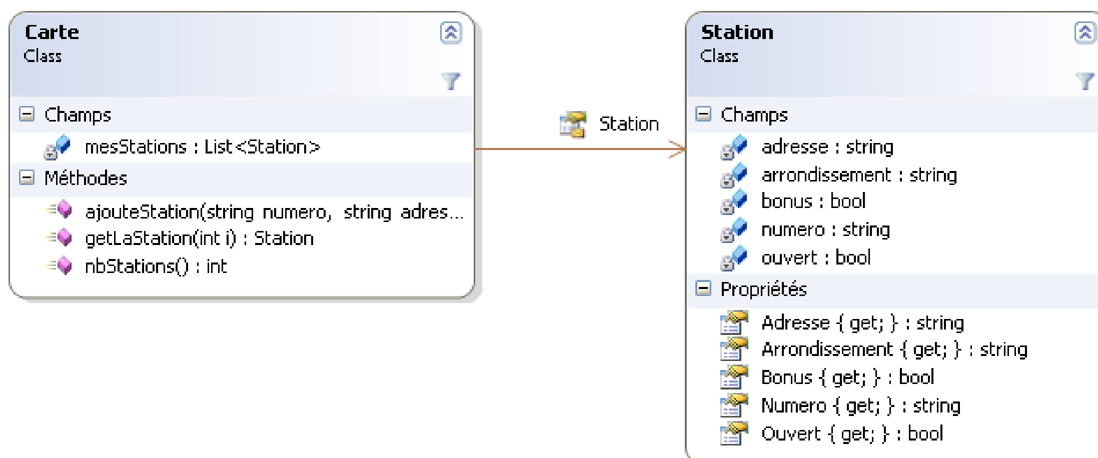


La liste des points d'accès est obtenue à partir de la première ressource évoquée, les disponibilités sont récupérées à partir de l'URI du point d'accès.

4. Réalisation

4.1 Les classes métiers

Pour charger la liste des points d'accès, nous allons utiliser le modèle métier suivant :



Remarques :

- La notion de bonus du point d'accès (station) correspond à un crédit horaire (15 minutes) attribué au point en fonction de sa pénibilité d'accès. La station peut être ouvert ou fermée.
- Le diagramme de classe est construit par Visual Studio (ici, après avoir défini les classes).
- Le champ arrondissement représente un arrondissement ou un département (dans le cas du 92, 93, et 94 –il n'y a pas de stations dans le 91-).
- Le lien entre la classe Carte et les stations est implémenté à l'aide du type générique List<Classe>.
- Dans la classe Station, chaque champ a une *propriété* correspondante (unique méthode get). Ce choix permettra de faciliter le *binding* entre la liste des stations et le composant graphique *DataGridView*.
- Les constructeurs ont été masqués ici.

Travail à faire.

Exercice 1

Créer la classe avec un constructeur qui valorise chaque champ :

```
public Station(string numero, string adresse, bool ouvert, bool bonus)
```

Le champ arrondissement (ou département) sera obtenu à partir du numero ; en effet dans le fichier xml (carto) l'attribut number permet d'extraire l'arrondissement (ou le département).Pour cela, observez bien la construction du numéro de station en vous connectant au site :
http://www.velib.paris.fr/service/carto.

Le test sera fait ainsi :

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Station s = new Station("20021", "15 rue petit", true, true);
    Station s1 = new Station("31023", "11 rue Blanche", true, true);
    Station s2 = new Station("8567", "45 rue Noire", true, true);
    Console.WriteLine(s.Arrondissement);
    Console.WriteLine(s1.Arrondissement);
    Console.WriteLine(s2.Arrondissement);

    // Application.Run(new FrmStations());
```

Regardez la sortie (onglet Sortie en bas) et vérifiez que vous obtenez bien : 20, 93 et 8.

La classe Carte possède un attribut privé *mesStations* de type List<Station>
Le code du constructeur est fourni :

```
public Carte()
{
    this.mesStations = new List<Station>();
}
```

La méthode *ajouteStation* a la signature suivante :

```
    public void ajouteStation(string numero, string adresse, bool ouvert,
bool bonus)
```

Travail à faire

Exercice 2
Ecrire le code de la classe Carte.
Tester avec :

```
static void Main()
{
     Application.EnableVisualStyles();
     Application.SetCompatibleTextRenderingDefault(false);
     Carte c = new Carte();
     c.ajouteStation("20021", "15 rue petit", true, true);
     c.ajouteStation("31023", "11 rue Blanche", true, true);
     c.ajouteStation("8567", "45 rue Noire", true, true);
     Console.WriteLine(c.getLaStation(1).Adresse);

     // Application.Run(new FrmStations());
```

*Vérifiez que vous obtenez bien **93**.*

4.2 La classe Passerelle

Pour charger la carte des stations, il faut se connecter au site et parser le fichier XML (*carto*) récupéré.
Pour cela, nous allons créer une nouvelle classe *Passerelle* ayant deux champs *static* :

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.IO;
using System.Xml;

namespace WindowsApplication1
{
    class Passerelle
    {
        private static string urlCarto = "http://www.velib.paris.fr/service/carto";
        private static string urlDispo = "http://www.velib.paris.fr/service/stationdetails/";
```

Et une méthode *static* :

```
public static Carte getCarte()
```

Cette méthode devra :

- Créer une requête HTTP, classe *HttpWebRequest*, à partir de la méthode statique *create* de la classe *WebRequest*. Attention au type attendu !
- Indiquer que la méthode est *Get*.
- Récupérer la réponse dans un objet *WebResponse*.
- Mettre cette réponse dans un flux *StreamReader*.
- Récupérer un objet *XmlReader (xml)* à partir de la méthode statique *Create* de cette classe.

A cette étape, on a récupéré un pointeur, *xml* de type *XmlReader* qui va parcourir le flux XML.
Pour terminer il faudra boucler à l'aide de la méthode *ReadToNextSibling* du pointeur. On vous fournit le corps de la boucle :

```
xml.ReadToFollowing("marker");
do
{
    // code qui extrait chaque valeur
    // des attributs number, fullAddress, open et bonus
    // et l'ajoute à la carte qui sera retournée
}
while (xml.ReadToNextSibling("marker"));
```

Travail à faire

Exercice 3

Ecrire la classe Passerelle et sa méthode getCarte().
Tester avec :

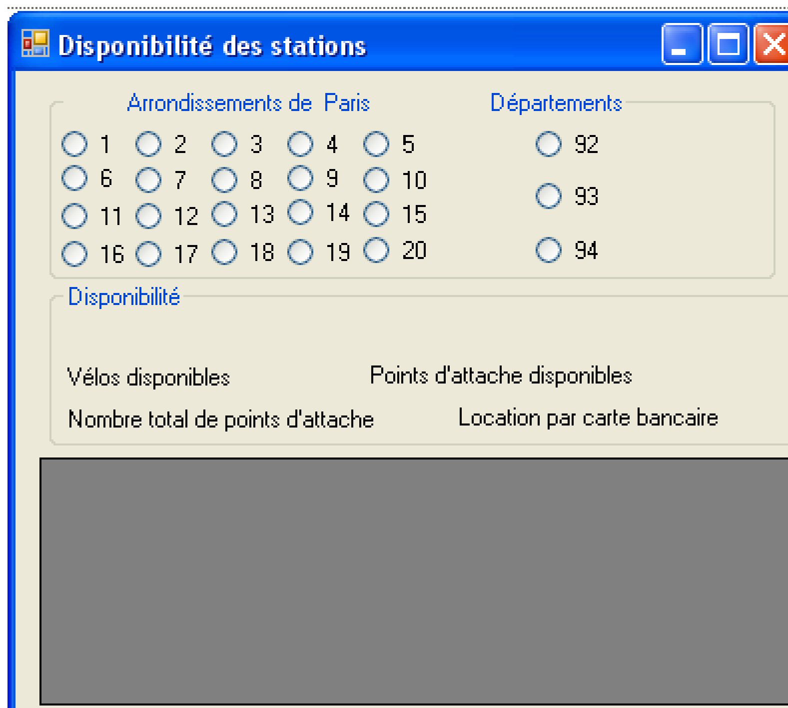
```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Carte c = Passerelle.getCarte();
    Console.WriteLine(c.getLaStation(1).Adresse);

    // Application.Run(new FrmStations());
```

Vérifier que vous obtenez bien :

```
1 RUE SERPOLLET - 75020 PARIS
```

D-3 L'interface

On peut maintenant construire l'interface :



Les propriétés *Text* des boutons radio correspondent aux arrondissements/départements. Dans la zone de disponibilité, des labels permettent d'afficher les informations pour chaque station, choisir une police et une couleur particulière –cf interface présentée plus haut-. Un DataGridView a été installé.

Un clic sur chaque bouton radio charge les stations concernées de la classe Carte.
Dans le formulaire, trois attributs privés sont déclarés et à la construction du formulaire la carte est chargée :

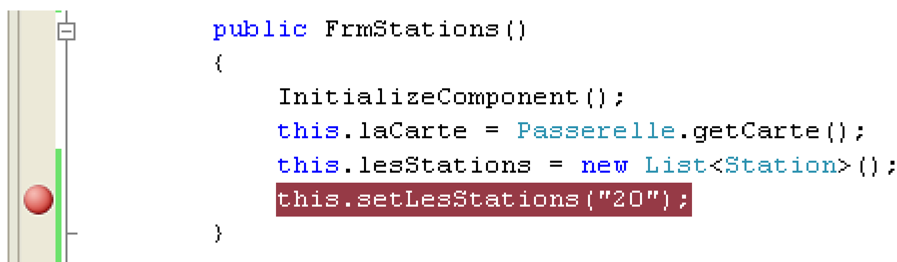
```
namespace Velib
{
    public partial class FrmStations : Form

    {
        private List<Station> lesStations;
        private Carte laCarte;
        private string[] lesInfos ;
        public FrmStations()
        {
            InitializeComponent();
            this.laCarte = Passerelle.getCarte();
            this.lesStations = new List<Station>();
        }
```

La liste *lesStations* contiendra les stations d'un arrondissement ou d'un département.
Le tableau *lesInfos* contiendra les valeurs des disponibilités demandées à la *passerelle.*
Une méthode, appelée à chaque clic sur un bouton radio, permet de valoriser la liste des stations concernées –*lesStations*- :

```
private void  setLesStations(string arrondissement)
```

Travail à faire

Exercice 4
Construire le formulaire. Ecrire la méthode setLesStations. Ne pas oublier de commencer par vider la liste.
Tester en appelant cette méthode, en plaçant un point d'arrêt et en observant le contenu du champ lesStations :



```
public FrmStations()
{
    InitializeComponent();
    this.laCarte = Passerelle.getCarte();
    this.lesStations = new List<Station>();
    this.setLesStations("20");
}
```

On peut écrire le gestionnaire d'événement du clic sur chaque bouton radio ; chaque clic appellera le même gestionnaire.
Ce gestionnaire doit :

- Mettre à *null* la propriété *DataSource* du *dataGridView*
- Récupérer la propriété Text de l'objet qui a été cliqué (sender)
- Appeler la méthode *setLesStations*
- Valoriser la propriété *DataSource* du *DataGridView* avec la liste des stations (champ *lesStations*)
- Conserver visibles les seules colonnes adresse, ouvert et bonus. Ajuster les largeurs des colonnes.

Travail à faire

Exercice 5
Ecrire le gestionnaire d'événement décrit ci-dessus. Tester.

Pour chaque chargement des disponibilités on utilise le gestionnaire d'événement suivant qui est appelé à chaque clic dans le DataGridView :

```
private void dtVStations_CellClick(object sender, DataGridViewCellEventArgs e)
{
    int ligne = e.RowIndex;
    string num = this.dtVStations[0, ligne].Value.ToString();
    string adresse = this.dtVStations[2, ligne].Value.ToString();
    this.lesInfos = Passerelle.getDispo(num, adresse);
    this.chargerLesLabels(this.lesInfos);
}
```

Ce gestionnaire récupère les valeurs des numéro et adresse (lignes 2 et 3) et demande à la Passerelle de retourner les informations disponibles (méthode *getDispo*). Ensuite, les zones de labels sont valorisées grâce à la méthode *chargerLesLabels*.

Travail à faire

Exercice 6
En s'inspirant du code déjà écrit (méthode getCarte), ajouter dans la classe Passerelle la méthode attendue ainsi que la méthode chargerLesLabels du formulaire. Tester.

5. Pour aller un peu plus loin.

Plusieurs problèmes peuvent survenir ; tout d'abord l'indisponibilité de la ressource, ensuite le mode bloquant (synchrone) du traitement.

5.1 Indisponibilité de la ressource (des disponibilités)

Ce cas est fréquent. La ressource se présente sous la forme :

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <station>
    <available />
    <free />
    <total />
    <ticket />
  </station>
```

On peut envisager dans ce cas de tester si le formulaire récupère bien le tableau de string ; dans l'hypothèse contraire la méthode *chargerLesLabels* affichera l'indisponibilité.

Travail à faire

Exercice 7
Ajouter dans le formulaire un label visible seulement dans le cas où les informations sont indisponibles, modifier le code de la méthode chargerLesLabels en conséquence.

5.2 Mode asynchrone

Le chargement des disponibilités prend un certain temps qui bloque l'utilisateur pour une autre tâche – par exemple la sélection d'un autre arrondissement ; nous sommes en mode synchrone.
Dans la méthode *getDipo* de la passerelle, le code :

```
string url = urlDispo + numero;
HttpWebRequest requete = (HttpWebRequest)WebRequest.Create(url);
requete.Method = WebRequestMethods.Http.Get;
WebResponse rep = requete.GetResponse(); // appel synchrone
StreamReader sr = new StreamReader(rep.GetResponseStream());
```

Utilise la méthode *GetResponse* synchone.

Pour faire un appel asynchrone (non bloquant) nous allons utiliser le couple *BeginGetResponse/EndGetResponse* appliqué à la requête HTTP.

Pour ce faire, nous allons un peu modifier l'architecture et effectuer l'appel asynchrone directement dans le formulaire.

```
private void dtVStations_CellClick(object sender, DataGridViewCellEventArgs e)
{
    int ligne = e.RowIndex;
    string num = this.dtVStations[0, ligne].Value.ToString();
    string adresse = this.dtVStations[2, ligne].Value.ToString();
    this.adresse = adresse;
    string url = Passerelle.getUrlDispo() + num;
    HttpWebRequest requete = (HttpWebRequest)WebRequest.Create(url);
    requete.Method = WebRequestMethods.Http.Get;
    requete.BeginGetResponse(new AsyncCallback(getValAsynchrone), requete);

}
```

La méthode *BeginGetResponse* attend une méthode *déléguée* dont la signature doit être :

```
public  void getValAsynchrone(IAsyncResult asyn)
```

Cette méthode doit :

- Récupérer la requête.
- Appeler la méthode *EndGetResponse.*
- Construire le fichier et le flux xml (cf mode synchrone).
- Construire le tableau de chaînes à partir du flux xml.
- Appeler le chargement des labels.

On vous fournit les deux premiers points :

```
public  void getValAsynchrone(IAsyncResult asyn)
{
     HttpWebRequest requete = (HttpWebRequest)asyn.AsyncState;
     HttpWebResponse rep = (HttpWebResponse)requete.EndGetResponse(asyn);
      ...
```

Travail à faire

Exercice 8
Terminer le code de cette méthode.

Si l'on lance l'application, tout doit se dérouler normalement ; par contre en mode débogage une alerte est lancée de type *InvalidOperationException* concernant une opération inter-thread.
En effet, si nous écrivons l'appel de *chargerLesLabels* dans la méthode asynchrone, c'est le thread associé qui modifiera les champs des labels ; or, ce n'est pas ce thread qui les a créés. Il y a conflit (potentiel) des threads sur l'accès aux composants graphiques. Visual Studio signale ce conflit ; les opérations *cross-threads* sont interdites.

Pour lever le conflit, il faut ***dans la fonction asynchrone demander au formulaire (thread principal) de modifier les labels***.
Ceci est possible grâce à la méthode *invoke* des composants.

On vous fournit cet appel :

```
   this.Invoke(new delegueChargeLabels(chargerLesLabels), new object[] {
valeurs });
```

Remarque : *valeurs* représente le tableau des 5 strings valorisés juste avant.

Travail à faire

Exercice 9
Déclarer le Delegate nécessaire à cet appel de la fonction chargerLesLabels.
Tester.

Corrigé

Exercice 1
Créer la classe avec un constructeur qui valorise chaque champ :
`public Station(string numero, string adresse, bool ouvert, bool bonus)`
Le champ arrondissement (ou département) sera obtenu à partir du numero ; en effet dans le fichier xml (carto) l'attribut number permet d'extraire l'arrondissement (ou le département).Pour cela, observez bien la construction du numéro de station en vous connectant au site.

```
class Station
    {
        public Station(string numero, string adresse, bool ouvert, bool
bonus)
        {
            this.numero = numero;
            this.adresse = adresse;
            if (this.Numero.Length == 5)
            {
                if (Convert.ToInt32(this.numero) >= 21000)
                    this.arrondissement = "9" + this.numero.Substring(0,
1);
                else
                    this.arrondissement = this.numero.Substring(0, 2);
            }
            else
                this.arrondissement = this.numero.Substring(0, 1);
            this.ouvert = ouvert;
            this.bonus = bonus;

        }

        public string Numero
        {
            get
            {
                return this.numero;
            }
        }

        public string Arrondissement
        {
            get
            {
                return this.arrondissement;
            }
        }
        public string Adresse
        {
            get
            {
                return this.adresse;
            }
        }
        public bool Ouvert
        {
            get
            {
                return this.ouvert;
            }
        }
        public bool Bonus
        {
            get
```

```
            {
            return this.bonus;
            }
        }
        private string numero;
        private string arrondissement;
        private string adresse;
        private bool ouvert;
        private bool bonus;
    }
```

Remarque : comme indiqué plus haut, le binding entre un objet List<*Classe*> et un composant graphique ne peut être mis en oeuvre que si la classe exposée possède des *propriétés* d'accès (pas seulement des méthodes). Dans le cas contraire il faudrait par le code définir chacune des colonnes du DataGridView.

Exercice 2

Ecrire le code de la classe Carte.

```
class Carte
    {
        public Carte()
        {
            this.mesStations = new List<Station>();
        }
        public void ajouteStation(string numero, string adresse, bool
ouvert, bool bonus)
        {
            Station s = new Station(numero, adresse, ouvert, bonus);
            this.mesStations.Add(s);
         }
        public Station getLaStation(int i)
        {
            return this.mesStations[i];
        }
        public int nbStations()
        {
            return this.mesStations.Count;
        }
        private List<Station>  mesStations;
    }
```

Exercice 3
Ecrire la classe Passerelle et sa méthode getCarte().

```
class Passerelle
    {
        private static string urlCarto =
"http://www.velib.paris.fr/service/carto";
        private static string urlDispo =
"http://www.velib.paris.fr/service/stationdetails/";
        public static Carte getCarte()

        {
            try
            {
                HttpWebRequest requete =
(HttpWebRequest)WebRequest.Create(urlCarto);
                requete.Method = WebRequestMethods.Http.Get;
                WebResponse rep = requete.GetResponse();
                StreamReader sr = new
StreamReader(rep.GetResponseStream());
                XmlReader xml = XmlReader.Create(sr);
                Carte c = new Carte();
                xml.ReadToFollowing("marker");
                do
                {
                    string num = xml.GetAttribute("number");
                    string adr = xml.GetAttribute("fullAddress");
                    string open = xml.GetAttribute("open");
                    string bonus = xml.GetAttribute("bonus");
                    bool o = (open == "1");
                    bool b = (bonus == "1");
                    c.ajouteStation(num, adr, o, b);
                }
                while (xml.ReadToNextSibling("marker"));
                return c;
            }
            catch (Exception ex)
            {
                Console.Write(ex.Message);
                return null;
            }
        }
```

Exercice 4
Construire le formulaire. Ecrire la méthode setLesStations. Ne pas oublier de commencer par vider la liste.

```
private void  setLesStations(string n)
        {
            this.lesStations.Clear();
            for(int i = 0; i< this.laCarte.nbStations(); i++)
            {
                Station s =this.laCarte.getLaStation(i);
                if (s.Arrondissement == n)
                    this.lesStations.Add(s);
            }
        }
```

Exercice 5
Ecrire le gestionnaire d'événement décrit ci-dessus.

```
private void radioButton2_Click(object sender, EventArgs e)
        {
            this.dtVStations.DataSource = null;
            string s = ((RadioButton)sender).Text;
            this.setLesStations(s);
            this.dtVStations.DataSource = this.lesStations;
            this.dtVStations.Columns["Numero"].Visible = false;
            this.dtVStations.Columns["Arrondissement"].Visible = false;
            this.dtVStations.Columns["Adresse"].Width = 250;
            this.dtVStations.Columns["Ouvert"].Width = 30;
            this.dtVStations.Columns["Bonus"].Width = 30;        }
```

Exercice 6
En s'inspirant du code déjà écrit (méthode getCarte), ajouter dans la classe Passerelle la méthode attendue ainsi que la méthode chargerLesLabels du formulaire.

```
public static string[] getDispo(string numero,string adresse)
        {
            try
            {
                string url = urlDispo + numero;
                HttpWebRequest requete =
(HttpWebRequest)WebRequest.Create(url);
                requete.Method = WebRequestMethods.Http.Get;
                WebResponse rep = requete.GetResponse();
                StreamReader sr = new
StreamReader(rep.GetResponseStream());
                XmlReader xml = XmlReader.Create(sr);
                string[] valeurs = new string[5];
                int i = 1;
                valeurs[0] = adresse;
                while (xml.Read())
                {
                    if (xml.NodeType == XmlNodeType.Text)
                        valeurs[i++] = xml.Value;
                }
                return valeurs;
            }
            catch (Exception ex)
            {
                Console.Write(ex.Message);
                return null;
            }
        }
private void chargerLesLabels(string[] lesInfos) // méthode du formulaire
        {
            lblPoint.Text = this.lesInfos[0];
            this.lblDispo.Text = this.lesInfos[1];
            this.lblPointsDispo.Text = this.lesInfos[2];
            this.lblPointsTotal.Text = this.lesInfos[3];
            if (this.lesInfos[4] == "1")
                this.lblCarte.Text = "oui";
            else
                this.lblCarte.Text = "non";

        }
```

Exercice 7
Ajouter dans le formulaire un label visible seulement dans le cas où les informations sont indisponibles, modifier le code de la méthode chargerLesLabels en conséquence.

```
private void chargerLesLabels(string[] lesInfos)
        {
            if (lesInfos[1] == null)
            {
                label5.Visible = true;
                label5.Height = this.grpDispo.Height;
                label5.Width = this.grpDispo.Width;
            }
            else
            {
                label5.Visible = false;
                lblPoint.Text = lesInfos[0];
                this.lblDispo.Text = lesInfos[1];
                this.lblPointsDispo.Text = lesInfos[2];
                this.lblPointsTotal.Text = lesInfos[3];
                if (lesInfos[4] == "1")
                    this.lblCarte.Text = "oui";
                else
                    this.lblCarte.Text = "non";
            }
        }
```

Exercice 8
Terminer le code de cette méthode.

```
public  void getValAsynchrone(IAsyncResult asyn)
        {
            HttpWebRequest requete = (HttpWebRequest)asyn.AsyncState;
            HttpWebResponse rep =
(HttpWebResponse)requete.EndGetResponse(asyn);
            StreamReader sr = new StreamReader(rep.GetResponseStream());
            XmlReader xml = XmlReader.Create(sr);
            string[] valeurs = new string[5];
            valeurs[0] = this.adresse;
            int i = 1;
            while (xml.Read())
            {
                string s = xml.Name;
                if (xml.NodeType == XmlNodeType.Text)
                    valeurs[i++] = xml.Value;
            }
            this.Invoke(new delegueChargeLabels(chargerLesLabels), new
object[] { valeurs });

        }
```

Exercice 9
Déclarer le Delegate nécessaire à cette appel de la fonction chargerLesLabels..

```
private delegate void delegueChargeLabels(string[] str);
```