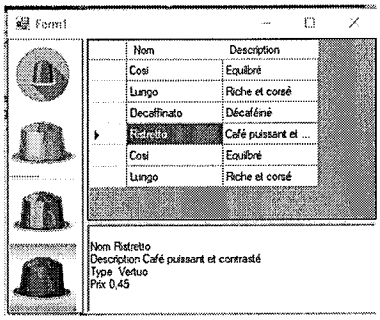
Licence Informatique **Lundi 4 janvier 2020**

Conception et développement avancé d'applications **Examen - Durée 2h**
Documents CM, TD et TP autorisés

Dans le cadre du développement d'une application de vente en ligne de capsules de café, il a été décidé de créer un prototype pour l'avant-vente du projet. Ce prototype a été développé en Windows Forms (Framework .net), en langage C#, et sous Visual Studio 2019.
L'application ne comporte que deux classes :
- une classe « Capsule » qui décrit une capsule avec le nom du café, sa description, un prix et une photo
- la classe principale nommée « Form1 » qui comporte une liste des capsules disponibles, et un panier d'achat qui est aussi une liste de capsules.

L'interface de l'application est la suivante :



L'application comporte une liste des photos (de type ListBox) des capsules qui sont proposées à gauche, en haut à droite une grille (de type DataGridView) qui représente le contenu du panier (capsules qui ont été sélectionnées dans la liste), et en bas à droite une zone d'édition (de type RichTextBox).
La sélection d'une capsule à ajouter dans le panier se fait en cliquant sur sa photo dans la liste des photos. La grille présente les capsules qui ont été ajoutées dans le panier, avec uniquement leur nom et leur description. Chaque ligne représente une capsule, il peut donc y avoir plusieurs lignes identiques, si la même capsule a été sélectionnée plusieurs fois.
Le clic sur une case de la grille provoque l'affichage des informations complètes de la capsule sélectionnée, dans la zone d'édition.

Exercice 1 : C# (5 points)

La classe « Capsule » - Annexe 1
L'annexe 1 décrit complètement la classe « Capsule ». En utilisant cette annexe :

1. **(0,5 pt)** Expliquer la portion de code suivante : « `public class Capsule : IEquatable<Capsule>, IComparable<Capsule>` »
2. **(2 pts)** La classe comporte deux implémentations de la méthode « `Equals` »
   a. Expliquer à quoi correspond chacune de ces deux méthodes et ce qu'elle fait.
   b. Donner un exemple d'usage de chacune de ces méthodes, si on souhaitait les tester dans le programme principal d'une application console. Donner les déclarations nécessaires, les créations des objets si nécessaire, et les appels.
3. **(1,5 pts)** L'opérateur == a été surchargé
   a. Expliquer précisément le code de cette surcharge en particulier en expliquant quelle est la méthode « `Equals` » appelée (parmi les 2 implémentées)
   b. Donner un exemple d'usage de cet opérateur comme pour la question 1b
4. **(1 pt)** En quoi l'implémentation de ces opérateurs de comparaison et de ces méthodes « `Equals` » et « `CompareTo` » sont intéressantes pour l'usage de cette classe (d'une façon générale, pas uniquement pour cette application)

Exercice 2 : Application Windows Forms (10 points)

La classe « Form1 » - Annexe 2
Note : Seules les méthodes proposées dans le code des annexes sont à utiliser. Si vous jugez nécessaire d'utiliser d'autres méthodes, décrivez leur code complet en C#.
L'annexe 2 décrit partiellement la classe principale nommée « Form1 ». L'ensemble des capsules disponibles et le panier sont gérés à l'aide de listes de capsules (de type List<Capsule>) nommées « *lc* » et « *panier* ». Les photos des capsules sont gérées en ressources dans le projet.
Le constructeur de la classe est la suivant :

```
public Form1()
        {   InitializeComponent();
            lc = new List<Capsule> ();
            panier = new List<Capsule>();
            InitListeCapsules();
            InitListePhotos();
            InitGrille();
        }
```

5. **(1,5 pts)** La méthode « *InitListeCapsules* » permet de remplir la liste des capsules avec des capsules à des fins de tests de l'application. Donner le code de cette méthode permettant l'ajout d'une capsule nommée « Lungo », avec la description « Riche et corsé », le prix de « 0,35 » et la photo en ressource nommée « Capsule1 ».
   `public void InitListeCapsules(){ à compléter}`

6. **(2,5 pts)** La méthode « *InitListePhotos* » permet l'initialisation de la liste des photos (*de type ListBox et nommée ListePhotos*) des capsules disponibles qui sont dans la liste « lc ». On supposera que cette liste est remplie correctement.
   a. Expliquer les différentes étapes nécessaires pour initialiser cette liste de photos.
   b. Donner les déclarations et les parties de code correspondantes en indiquant précisément où ces portions de code, méthodes ou gestionnaires d'événement doivent être décrits.
7. **(3 pts)** La méthode « *InitGrille* » permet de créer les colonnes et remplir la grille (*de type DataGridView et nommée Grille*) avec les informations (uniquement le nom et la description) des capsules du panier. A chaque ajout dans le panier, cette grille sera reconstruite. La grille a été paramétrée pour ne pas permettre les ajouts ni la suppression.
   Donner le code de cette méthode.

```
public void InitGrille()
{       Grille.Rows.Clear();
        Grille.Columns.Clear();
        // à compléter
}
```

8. **(1,5 pts)** Le clic sur une photo de la liste permet l'ajout de cette capsule dans le panier, et l'affichage de la grille à jour (avec la méthode *InitGrille*). Donner le code de ce gestionnaire d'évènement.
   `private void ListePhotos_SelectedIndexChanged(object sender, EventArgs e) { à completer}`
9. **(1,5 pts)** Le clic sur une case de la grille permet l'affichage des informations de la capsule correspondante dans la zone d'édition de type RichTextBox et nommée « Edition ». Donner le code de ce gestionnaire d'évènement.
   `private void Grille_CellContentClick(object sender, DataGridViewCellEventArgs e) { à compléter}`

Exercice 3 : Application ASP.NET MVC (3 points)

Une 2ème version du prototype a été implémentée sous forme d'application Web ASP.net selon le modèle MVC en utilisant l'environnement Visual Studio 2019. Cet environnement permet la génération d'un squelette d'application qui comporte une barre de menu avec 3 options : « Accueil » « A propos de » et « Contacts ».
Les annexes 3, 4 et 5 donnent le code des fichiers « Layout.cshmtl », « HomeController.cs » et « About.cshtml » générés dans ce squelette.

1. **(1,5 pts)** Expliquer le modèle d'architecture « Modèle – Vue – Controller » en détaillant le fonctionnement de la gestion du clic sur l'option « A propos de » du squelette de l'application.
   On précisera clairement sur cet exemple les parties de code concernées, la façon dont les événements sont déclenchés, les informations passées, etc.
   Vous pouvez utiliser les annexes pour les annoter si vous le souhaitez et les rendre avec votre copie.
2. **(1,5 pts)** Indiquer quelles seraient les étapes nécessaires pour mettre en place une nouvelle fonctionnalité dans cette architecture comme par exemple la visualisation de la liste des capsules (sans donner le code complet).

Exercice 4 : Bases de données (2 points)

Une 3ème version du prototype sera basée sur l'accès à une base de données pour stocker les informations sur les capsules, en utilisant les classes disponibles dans ADO.net.
L'annexe 6 propose un schéma de synthèse de ces classes. Utilisez ce schéma pour illustrer vos explications dans les réponses aux questions suivantes :

1. **(1 pt)** Expliquer la notion de classe abstraite et illustrer vos explications avec les classes issue de ADO.net
2. **(1 pt)** Expliquer la notion de classe fabrique et illustrer vos explications avec les classes issues de ADO.net

ANNEXE 1 – Classe « Capsule »

```
public enum TypeCapsule { Original, Vertuo }
```

```
public class Capsule : IEquatable<Capsule>, IComparable<Capsule>
    {   private string nom;
        private string description;
        private TypeCapsule typec;
        private float prix;
        private Image photo;

        public Capsule()
        {this.nom = ""; this.description = ""; this.typec = TypeCapsule.Original; this.prix = 0;}

        public Capsule(string n, string d, TypeCapsule ty, float p)
        {this.nom = n;this.description = d;this.typec = ty;this.prix = p;}

        public Image Photo { get { return this.photo; } set { this.photo = value; } }
        public string Nom{ get { return this.nom; } set { this.nom = value; } }
        public string Description {get {return this.description;} set {this.description = value;}}
        public TypeCapsule Typec { get { return this.typec; } set { this.typec = value; } }
        public float Prix { get { return this.prix; } set { this.prix = value; } }
        public string typecS
        {   get { return Enum.Format(typeof(TypeCapsule), this.typec, "g"); }
            set { this.typec = (TypeCapsule)Enum.Parse(typeof(TypeCapsule), value, false); }
        }

        public override string ToString()
        {   string s = "";
            s += "\nNom " + this.Nom;
            s += "\nDescription " + this.description;
            s += "\nType  " + this.typecS;
            s += "\nPrix " + this.prix;
            return s;
        }

        public override bool Equals(object obj)
        {   if (obj == null) return false;
            Capsule j = obj as Capsule;
            if (j == null) return false; else return this.Equals(j);
        }
        public bool Equals(Capsule other)
        {   if (other == null) return false; return (this.nom.Equals(other.nom));}


        public int CompareTo(Capsule other)
        { if (other == null) return 1; return this.nom.CompareTo(other.nom);}

        public int CompareTo(object obj)
        {   if (obj == null) return 1;
            Capsule j = obj as Capsule;
            if (j == null) return 1; else return this.CompareTo(j);


        public static bool operator ==(Capsule i1, Capsule i2)
        {   if ((Object)i1 == null) return ((Object)i2 == null); return i1.Equals(i2); }
        public static bool operator !=(Capsule i1, Capsule i2)
        {  if ((Object)i1 == null) return ((Object)i2 == null); return !i1.Equals(i2);}
        public static bool operator >(Capsule i1, Capsule i2)
        {   if (i1 == null) return false; return i1.CompareTo(i2) == 1; }
        public static bool operator <(Capsule i1, Capsule i2)
        {  if (i1 == null) return false; return i1.CompareTo(i2) == -1; }
    }
```

ANNEXE 2 – Classe « Form1»

```
public partial class Form1 : Form
{
    private List<Capsule> lc;
    private List<Capsule> panier;

    public Form1()
    {  InitializeComponent();
        lc = new List<Capsule> ();
        panier = new List<Capsule>();
        InitListeCapsules();
        InitListePhotos();
        InitGrille();
    }
    public void InitListeCapsules()
    { // A COMPLETER }

    public void InitGrille()
    { // A COMPLETER

    private void ListePhotos_SelectedIndexChanged(object sender, EventArgs e)
    { // A COMPLETER }

    private void Grille_CellContentClick(object sender, DataGridViewCellEventArgs e)
    { // A COMPLETER }
}
```

ANNEXE 3 – Application Web ASP.net - modèle MVC
Fichier « _Layout.cshtml » du squelette d'application

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - Mon application ASP.NET</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Nom de l'application", "Index", "Home", new { area = "" },
new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Accueil", "Index", "Home")</li>
                    <li>@Html.ActionLink("À propos de", "About", "Home")</li>
                    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
        </footer>
    </div>

    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
</html>
```

ANNEXE 4 – Application Web ASP.net - modèle MVC
Fichier « HomeController.cs » du squelette d'application

```
public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult About()
        {
            ViewBag.Message = "Your application description page.";

            return View();
        }

        public ActionResult Contact()
        {
            ViewBag.Message = "Your contact page.";

            return View();
        }
    }
```

ANNEXE 5 – Application Web ASP.net - modèle MVC
Fichier « About.cshtml » du squelette d'application

```
@{
    ViewBag.Title = "About";
}
<h2>@ViewBag.Title.</h2>
<h3>@ViewBag.Message</h3>

<p>Use this area to provide additional information.</p>
```

ANNEXE 6 – ADO.net