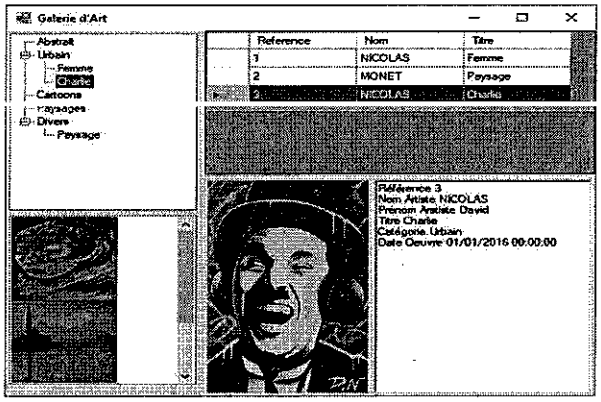
Licence Informatique **Lundi 7 janvier 2019**
Conception et développement avancé d'applications **Examen - Durée 2h**
Documents CM, TD et TP autorisés

Exercice 1 : C# et Application Windows Forms (16 points)

Une galerie d'art a décidé de mettre à disposition de ses visiteurs un outil numérique de recherche des œuvres de la galerie. Dans un premier temps, un prototype a été implémenté pour effectuer des tests. L'interface de ce prototype est la suivante.



Le prototype a été développée en langage C# sous forme d'une application Windows Forms.
Son interface comporte plusieurs zones :

- La 1ère zone en haut à gauche contient un arborescence (TreeView nommé ArbreCat) qui permet de présenter les œuvres selon leur catégorie (art abstrait, art urbain, etc.)
- La 2ème zone en bas à gauche contient une liste d'images des œuvres de la galerie (ListBox nommé ListePhotos) remplie avec des images.
- La 3ème zone en haut à gauche permet d'afficher sous forme d'une grille la référence, le nom de l'artiste et le titre de l'œuvre pour les œuvres de la galerie, sous forme d'une grille (DataGridView nommée Grille)
- La 4ème zone en bas à droite comporte à gauche la photo de l'œuvre dont le détail textuel est affiché à droite. Elle contient à gauche un composant PictureBox nommé PanPhoto et à droite un composant RichTextBox nommé Edition.

Dans un 1er temps, l'application comporte 2 classes qui modélisent les œuvres et une application principale qui gère une liste d'œuvres.

- La classe « **Oeuvre** » modélise une œuvre avec une référence, le nom et le prénom de l'artiste, une catégorie (sélectionnée dans une liste énumérée), une date de création et une photo. L'annexe 1 donne une description partielle de cette classe.
- La classe « **OeuvreHistorique** » modélise une œuvre d'un peintre célèbre (décédé), une description complémentaire est gérée dans ce cas. L'annexe 1 donne une description partielle de cette classe.
- La classe principale appelée « **Form1** » gère l'ensemble des œuvres à l'aide d'une liste. L'annexe 2 donne une description partielle de cette classe.

Les classes « Oeuvre » et « OeuvreHistorique » - Annexe 1

1. **(1 pt)** Donner le code du constructeur avec paramètre de la classe « **OeuvreHistorique** » qui initialise tous les attributs excepté la photo qui a une valeur par défaut (« Properties.Resources.Defaut »).
2. **(1 pt)** Donner le code de la méthode « ToString » de la classe « **OeuvreHistorique** ».
3. **(1,5 pt)** Compléter la classe « Oeuvre» pour qu'elle implémente l'interface « **IEquatable<T>** » qui ne comporte qu'une méthode « **public bool Equals (T other)** » en expliquant le code ajouté. La comparaison des œuvres est faite sur leur référence et leur catégorie, c'est-à-dire que deux œuvres sont considérées comme identiques si elles ont la même référence et sont de la même catégorie.

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, décrire leur code complet en C#.

4. **(1,5 pt)** Expliquer quels composants conteneurs ont été utilisés selon vous, pour construire l'interface décrite ci-dessus. Indiquer brièvement leur organisation sous une forme au choix (phrases ou schéma).

La liste des oeuvres est gérée dans une liste générique de type List<Oeuvre>.
Lors du lancement de l'application, plusieurs méthodes d'initialisation « InitArbre », « InitListePhotos » et « InitGrille » sont appelées dans le constructeur pour « remplir » les composants. Une méthode « InitListe » permet de créer des œuvres pour tester l'application.
L'annexe 2 décrit partiellement la classe « Form1 ». En utilisant cette annexe, répondre aux questions suivantes.

5. **(1,5 points)** Expliquer précisément les 4 premières instructions de la méthode « InitListe ».
6. **(2 points)** Expliquer précisément comment est « remplie » la liste des photos de la galerie (ListBox nommée ListePhotos). Expliquer tous les attributs, méthodes, instructions gérés pour la construction de cette liste d'images.
7. **(2 pts)** Compléter le code de la méthode « InitGrille » qui remplit la grille avec les informations (référence, nom de l'artiste et titre de l'œuvre) des œuvres de la galerie.
8. **(3 pts)** Donner le code de la méthode « InitArbre » qui remplit le composant de type TreeView nommé ArbreCat, avec les titres des œuvres classés par catégorie. On conservera pour chaque nœud de titre, la référence de l'œuvre dans la propriété « Name » du nœud.
   On rappelle si besoin que l'instruction « int i = (int) o.Categorie; » permet de connaître l'indice de la catégorie dans le type énuméré, 0 pour le 1er, 1 pour le 2ème, etc.

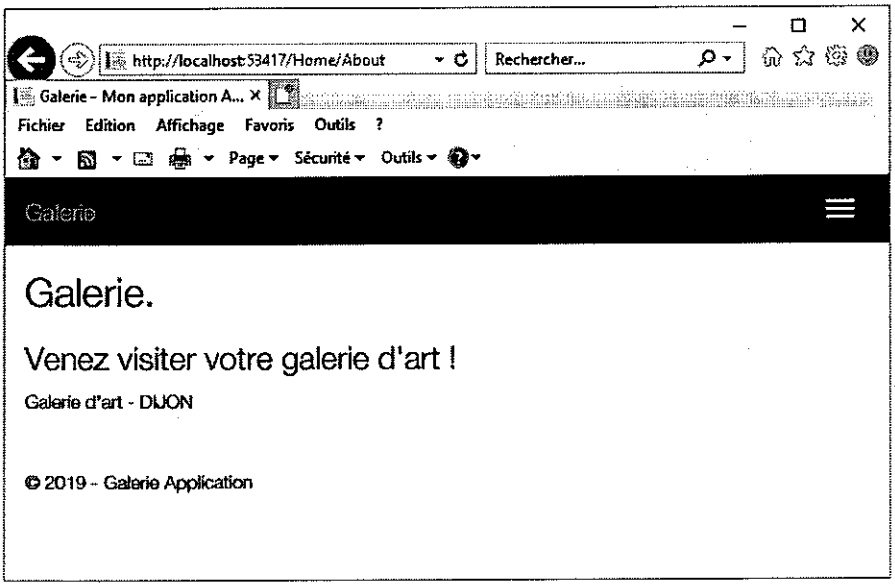
Le clic sur le titre d'une œuvre dans l'arborescence, dans une cellule de la grille ou lors de la sélection d'une photo dans la liste de photos produit toujours l'affichage des informations textuelles de l'œuvre dans la zone d'édition (de type RichTextBox nommée Edition) et de sa photo dans le PictureBox nommé PanPhoto.

9. **(2,5 pts)** Donner le code C# du gestionnaire de clic sur un nœud de l'arbre et expliquer ses paramètres.
   « private void ArbreCat_AfterSelect(object sender, TreeViewEventArgs e) »

Exercice 2 : Application APS.NET MVC (4 points)

Afin de rendre l'application de gestion de la galerie d'art plus facilement accessible, une version a été implémentée sous forme d'application Web ASP.net selon le modèle MVC en utilisant l'environnement Visual Studio 2015.
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. **(2,5 points)** 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 points)** Expliquer clairement les modifications à effectuer pour que le clic sur cette option « A propos de » affiche la page décrite ci-dessous.



ANNEXE 1 – Classes « Oeuvre » et « OeuvreHistorique »

```
public enum CategorieArt { Abstrait, Urbain, Cartoons, Paysages, Divers } ;
```

```
public class Oeuvre
    {
        private int reference;
        private string nomArtiste, prenomArtiste;
        private string titre;
        private CategorieArt cat;
        private DateTime dateCreation;
        private Image photo;

        public Oeuvre() { . . . }

        public Oeuvre(int r, string n, string p, string t, CategorieArt ca, DateTime dc) { . . . }

        public int Reference { get { return this.reference; } set { this.reference = value; } }
        public Image Photo { get { return this.photo; } set { this.photo = value; } }
        public string Nom { get { return this.nomArtiste; } set { this.nomArtiste = value; } }
        public string Prenom {get { return this.prenomArtiste;} set { this.prenomArtiste = value;} }
        public string Titre{ get { return this.titre; } set { this.titre = value; } }
        public CategorieArt Categorie { get { return this.cat; } set { this.cat = value; } }
        public string CategorieS
        {
            get { return Enum.Format(typeof(CategorieArt), this.cat, "g"); }
            set { this.cat = (CategorieArt)Enum.Parse(typeof(CategorieArt), value, false); }
        }

        public DateTime DateCreation
        { get { return this.dateCreation; } set { this.dateCreation = value; } }

        public override string ToString()
        {
            string s = "";
            s += "Référence " + this.Reference;
            s += "\nNom Artiste " + this.Nom + "\nPrénom Arstiste " + this.Prenom;
            s += "\nTitre " + this.Titre;
            s += "\nCatégorie " + this.CategorieS;
            s += "\nDate Oeuvre " + this.DateCreation + "\n";
            return s;
        }

       . . .
}
```

```
public class OeuvreHistorique : Oeuvre
    {    private string histoire;
        public string Histoire {  get { return this.histoire; } set { this.histoire = value; } }

       . . .

    }
```

ANNEXE 2 – Classe « Form1»

```
public partial class Form1 : Form
{
    private List<Oeuvre> listo;
    private ImageList listPh;

    public Form1()
    {
        InitializeComponent();
        this.listo = new List<Oeuvre>();
        this.listPh = new ImageList();
        listPh.ImageSize = new Size(100, 100);
        PanPhoto.SizeMode = PictureBoxSizeMode.StretchImage;
        this.InitListe();
        this.InitArbre();
        this.InitListePhotos();
        listePhotos.DrawMode = DrawMode.OwnerDrawFixed;
        this.InitGrille();
    }

    public void InitGrille()
    {
        DataGridViewTextBoxColumn Reference = new DataGridViewTextBoxColumn();
        DataGridViewTextBoxColumn Nom = new DataGridViewTextBoxColumn();
        DataGridViewTextBoxColumn Titre = new DataGridViewTextBoxColumn();

        Reference.HeaderText = "Reference";
        Reference.Name = "Reference";
        Reference.ReadOnly = true;
        Nom.HeaderText = "Nom";
        Nom.Name = "Nom";
        Nom.ReadOnly = true;
        Titre.HeaderText = "Titre";
        Titre.Name = "Titre";
        Titre.ReadOnly = true;
        Grille.Columns.AddRange(new System.Windows.Forms.DataGridViewColumn[] {
               Reference, Nom, Titre});
        Grille.AllowUserToDeleteRows = false;
        Grille.AllowUserToAddRows = false;

        // A COMPLETER
    }

    public Oeuvre rechOeuvre(int r)
    {   foreach (Oeuvre o in listo)
            if (o.Reference == r) return o;
           return null;
    }

    public void InitListe()
    {   // A EXPLIQUER
        Oeuvre o = new Oeuvre(1, "NICOLAS", "David", "Femme", CategorieArt.Urbain, new DateTime(2015, 1, 1));
        o.Photo = Properties.Resources.Femme;
        listPh.Images.Add(Properties.Resources.Femme);
        listo.Add(o);

        o = new OeuvreHistorique(2, "MONET", "Claude", "Paysage", CategorieArt.Divers, new DateTime(1920, 1, 1), "1840-1926");
        o.Photo = Properties.Resources.Paysage;
        listo.Add(o);
        listPh.Images.Add(Properties.Resources.Paysage);

        o = new Oeuvre(3, "NICOLAS", "David", "Charlie", CategorieArt.Urbain, new DateTime(2016, 1, 1));
        listPh.Images.Add(Properties.Resources.DN3);
        o.Photo = Properties.Resources.DN3;
        listo.Add(o);
    }
```

```
        public void InitArbre()
        {   // A COMPLETER       }

        public void InitListePhotos()
        {   listePhotos.ItemHeight = 100;
            foreach (Oeuvre o in listo) { listePhotos.Items.Add(o.Reference);}
        }

        private void listePhotos_DrawItem(object sender, DrawItemEventArgs e)
        {   Point p = e.Bounds.Location;
            listPh.Draw(e.Graphics, p, e.Index);
        }

        private void listePhotos_SelectedIndexChanged(object sender, EventArgs e)
        {   . . .  }

        private void ArbreCat_AfterSelect(object sender, TreeViewEventArgs e)
        {   // A COMPLETER }

        private void Grille_CellClick(object sender, DataGridViewCellEventArgs e)
        {   . . . }
    }
```

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>
```