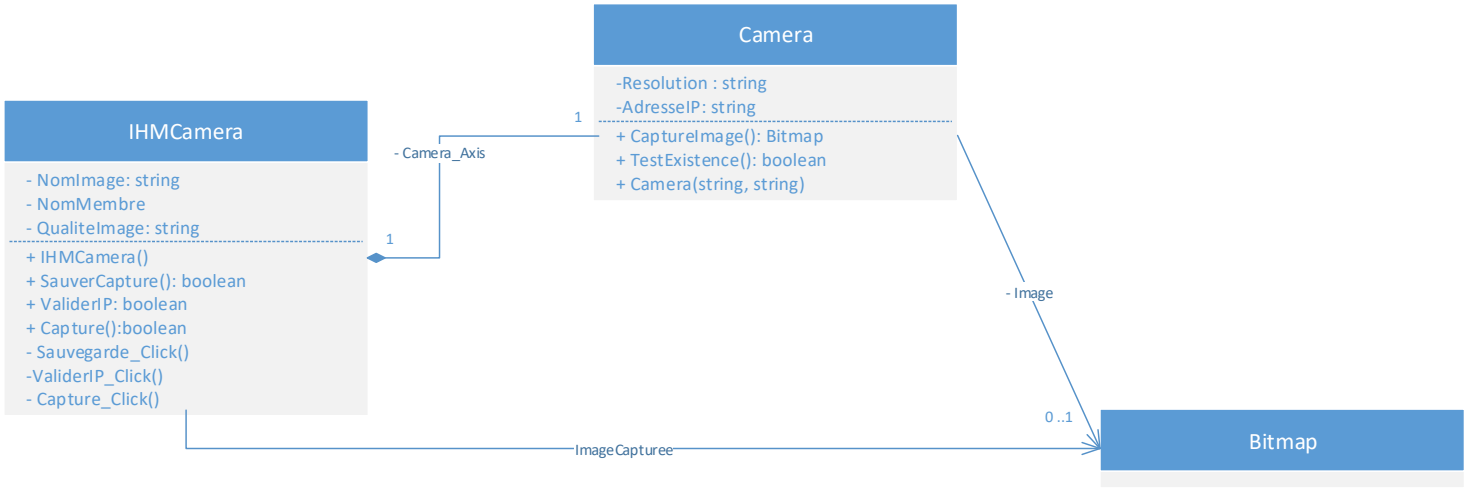
Durée: 6h

1. Introduction.

Notre télescope sera équipé d'une caméra wi fi montée en parallèle sur le tube. Cette caméra est de type Axis 1004 w, elle embarque un serveur web auquel nous connecterons notre application afin de récupérer les images.

2. Diagramme de classes du système "Contrôle Caméra".



Une aide sur les diagrammes de classes c'est ici

2.1. Lecture du diagramme de classe.

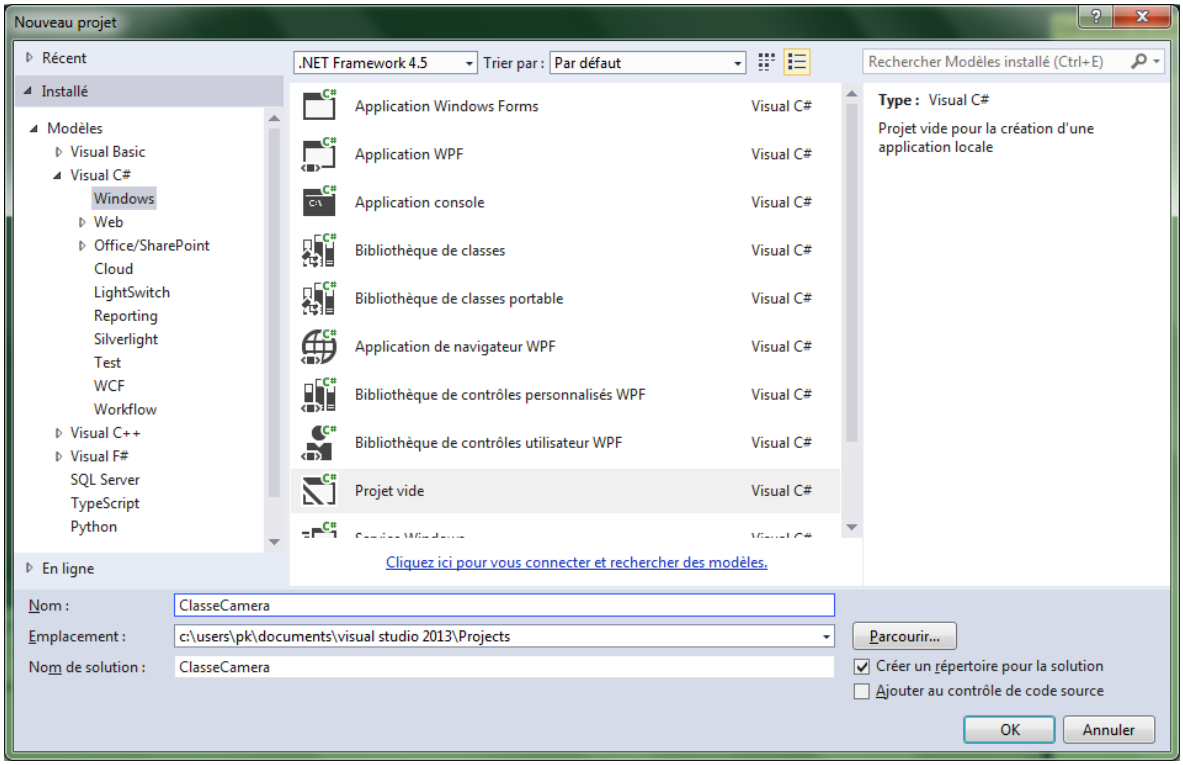
Afin de vous familiariser avec la lecture d'un diagramme de classes, répondez au QCM suivant:

Question	Réponses possibles
La représentation d'une classe comporte 3 parties	□ **oui** □ **non**
Le symbole + indique une visibilité publique	□ **oui** □ **non**
Le symbole moins indique une visibilité privée	□ **oui** □ **non**
Un item dont la visibilité est privée n'est accessible que…	□ **dans la classe à laquelle il appartient** □ **si on dispose d'accesseurs et mutateurs** □ **lorsque l'objet existe** □ **si c'est un entier**
La méthode CaptureImage …	□ **Ne reçoit aucun paramètre** □ **Reçoit un paramètre de type bitmap** □ **Retourne un objet de type bitmap**
Le lien Image matérialise	□ **La taille de l'image** □ **La présence d'un attribut Image dans la classe Caméra** □ **La couleur de l'image** □ **L'adresse où l'image sera stockée**
Le lien Camera_Axis signifie que	□ **La classe IHMCaméra comporte un objet Camera** □ **La classe IHMCamera va instancier un objet Camera** □ **La classe Camera est un attribut de la classe IHMCamera**

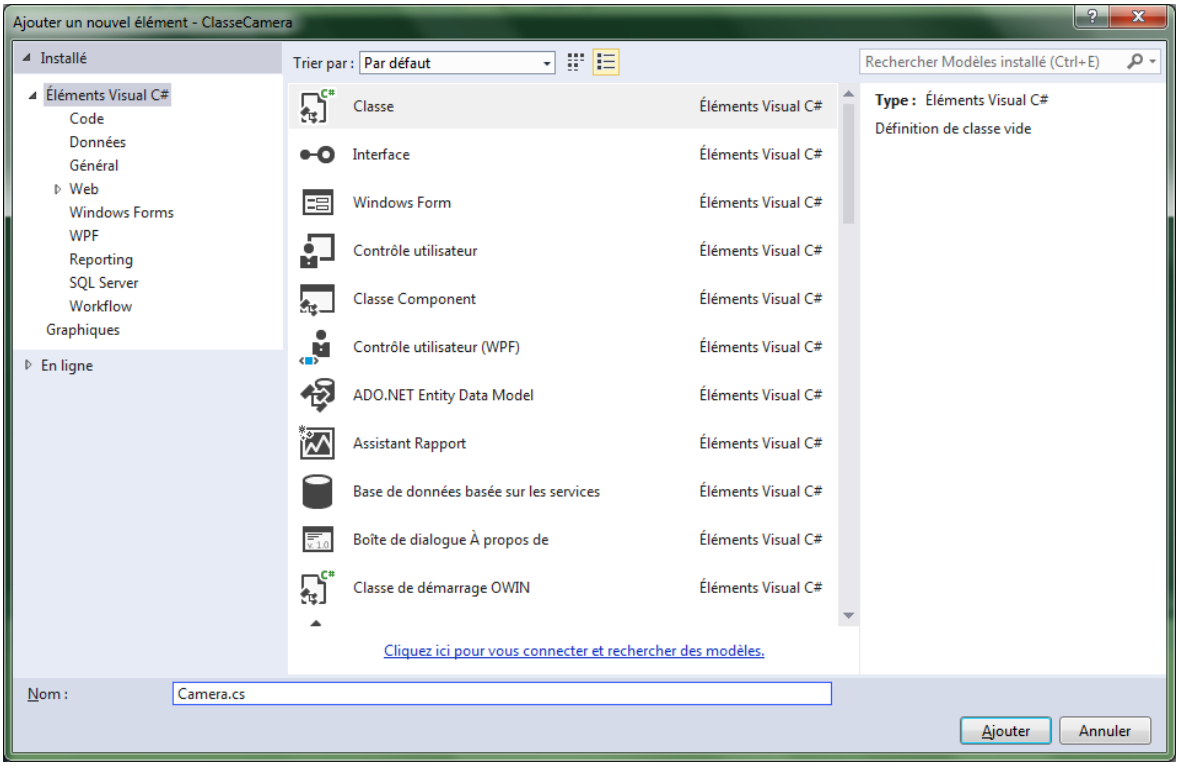
3. Réalisation de la classe Camera.

3.1. Création du projet "ClasseCamera".

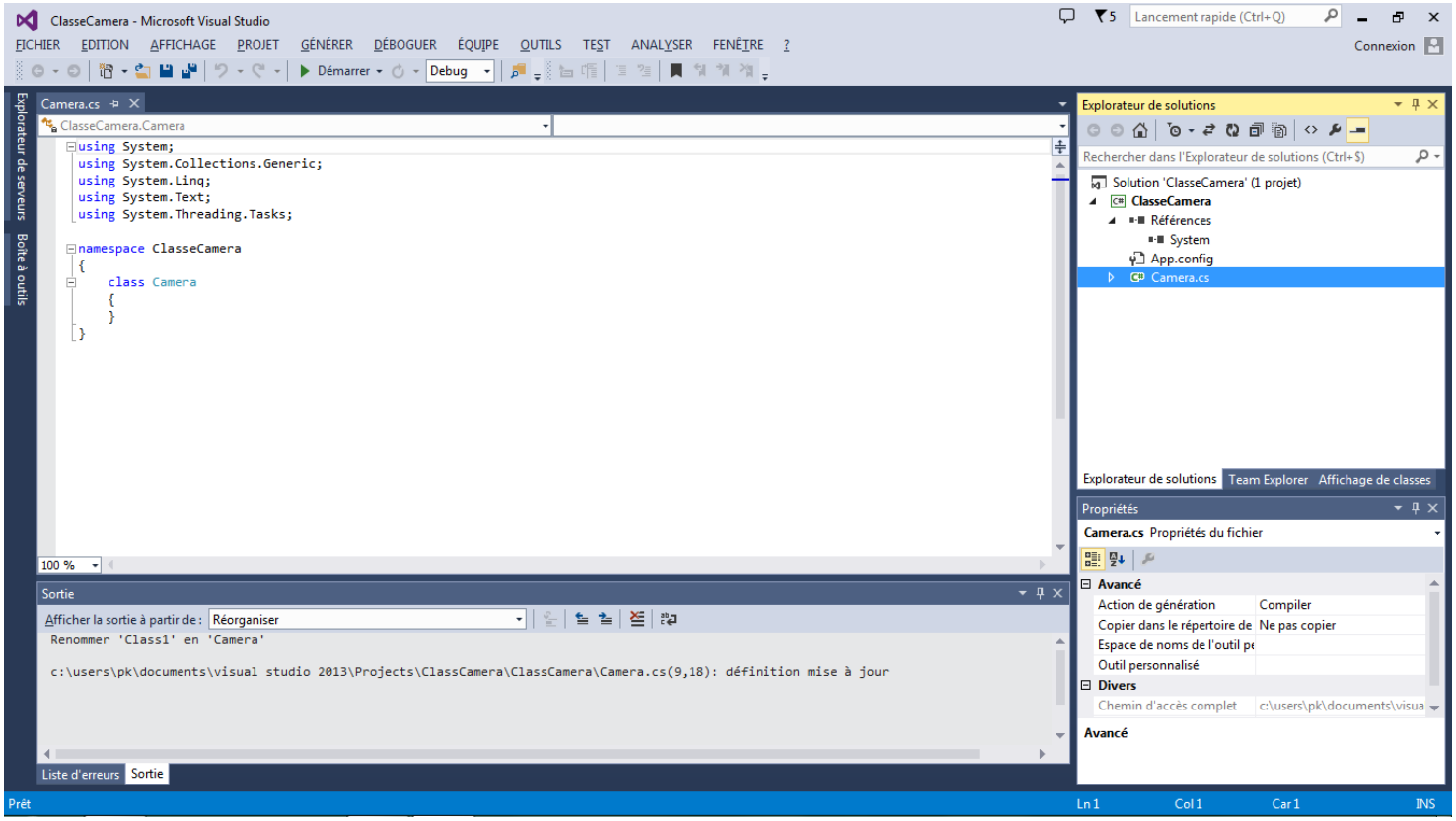
Dans votre EDI Visual studio 2013 créez un nouveau projet vide (**C#**) que vous appellerez ClasseCamera.



Une fois le projet crée, il faut y ajouter un élément de type classe, pour cela il faut **exécuter un clic droit sur ClasseCamera dans l'explorateur de solutions -> Ajouter -> Nouvel élément**, vous arrivez alors sur l'interface suivante:



Vous devez choisir un composant de type "Classe", nommez le Camera. Une fois l'appui sur Ajouter votre projet devient:



Nous allons maintenant pouvoir définir le squelette de la classe Camera conformément au diagramme de classes.

3.2. Définition du squelette de la classe Camera.

En lisant le diagramme de classe on voit que la classe Camera comporte deux attributs privés, Résolution et AdresseIP ainsi que trois méthodes publiques, le constructeur et les méthodes CaptureImage() et Testexistence().

Complétez le fichier Camera.cs afin d'y faire apparaitre les attributs et les méthodes présentées dans le diagramme de classes. Vous pouvez vous aider de l'exemple ci dessous afin de respecter le bon formalisme.

```
namespace ClasseCamera
{
    class Camera
    {
            //attributs

            private int IndiceCouleur;

            // methodes

            public string NomImage()
            {
                string NomImg = "paysage";

                return NomImg;              // on retourne le nom de l'image

            }
    }
}
```

En observant le diagramme de classe de plus près on voit qu'il existe une relation entre la classe Camera et la classe Bitmap, cette relation d'association indique que la classe Caméra a une image (ici de type Bitmap). On va donc la matérialiser sous la forme d'un attribut dans la classe Camera. Sur le diagramme on voit que la relation est nommée "-Image" cela signifie que l'objet Bitmap associé se nomme Image et que sa visibilité est de type privé.

On pourra donc ajouter la ligne suivante dans le squelette de la classe Camera.

```
private Bitmap Image;
```

3.3. Méthodes de la classe Camera.

3.3.1. Le constructeur.

Le constructeur reçoit en théorie deux paramètres qui sont la résolution de la caméra et l'adresse IP de cette dernière. Afin d'éviter les oublis on fixera deux valeurs par défaut pour ces paramètres, on adoptera la valeur "640x480" pour la résolution et "150.200.3.250" pour l'adresse IP. Le constructeur est chargé d'initialiser les attributs **Resolution** et **AdresseIP** avec les deux paramètres qu'il reçoit, il initialisera également l'attribut **Image** à null.

Complétez le code correspondant au constructeur.

3.3.2. La méthode TestExistence().

Cette méthode nous permettra de savoir si une caméra est bien connectée à l'adresse IP indiquée. Pour tester nous allons utiliser le protocole http et analyser la réponse à notre requête. Si nous obtenons un code de retour 200 alors tout va bien, il y a une caméra (du moins un équipement) qui répond. Si on obtient un code de retour 404 c'est qu'il n'y a pas de réponse donc pas de caméra. Pour requêter la caméra nous allons simplement demander une image. Cela se fait en invoquant un script cgi présent sur le serveur web de la caméra.

Testez ce script en saisissant la ligne suivante dans votre navigateur :

htpp://150.200.3.250/axis-cgi/jpg/image.cgi?resolution=640x480

Si tout se passe bien vous obtenez une image dans votre navigateur. Il y donc eu une réponse correcte de la caméra.

Refaites ce test en utilisant l'adresse IP suivante : 150.200.200.200, que constatez vous ?

Dans notre méthode TestExistence() nous allons utiliser les composants **HttpWebRequest, WebRequest et WebResponse.** Saisissez le code suivant dans la méthode TestExistence().

```
        public bool TestExistence()
        {
            bool CamExiste = false;      //booleen qui sera retourné pour indiquer si oui ou non une
                                         // camera est présente
              // on utilise une structure try ... catch afin de ne pas perturber le système
              // d'exploitation en cas d'erreur
            try
            {
                // on construit notre requete http
                HttpWebRequest Reponse = (HttpWebRequest)WebRequest.Create("http://" + AdresseIP +
"/axis-cgi/jpg/image.cgi?resolution=" + Resolution);
                // Si la réponse existe et contient des données
                if(Reponse.ContentLength>0)
                    CamExiste = true;
                //Sinon c'est qu'il n'y a pas de réponse donc pas de camera
```

```
        }
        catch
        {
            CamExiste = false;
        }

        return CamExiste;
    }
```

3.3.4. Méthode CaptureImage.

Cette méthode, comme son nom l'indique, permet de capturer une image sur la caméra. Le principe utilisé est toujours le même, on effectue une requête http en invoquant le script cgi présent sur le serveur web embarqué de la caméra.

Dans la pratique une image est constituée de pixels et chaque pixel est encodé sur un certain nombre d'octets, par exemple dans le format bitmap 24 bits chaque pixel est encodé sur trois octets, chaque octet correspond à une couleur RVB (Rouge, Vert, Bleu). On peut donc dire que matériellement une image est un ensemble d'octets. La taille de cet ensemble dépend de la résolution de l'image. Nous utiliserons donc un tableau d'octets afin de stocker notre image. Reste à en définir la taille.

Prenons par exemple une image en résolution 320x200 pixels soit 64 000 pixels. Si cette image est encodée en bitmap 24 bits il faut alors une capacité de stockage de 192 000 octets. Sachant que la résolution maximale de la caméra est 1920x1200 quelle doit être la taille minimale du tableau afin de pouvoir stocker une image dans cette résolution ?

Si on décrit les différentes étapes à réaliser dans cette méthode nous aurions l'algorithme suivant:

méthode: Capture Image

dépendance: ClasseCamera

Constantes: TAILLEBUFFER **c'est entier** **Valeur à calculer**

Variables:

Nom	Type	Valeur initiale	Commentaire
NbOctetsLus	entier	0	Variable intermédiaire utilisée pour Comptabiliser le nb d'octets lus dans un segment de données (une portion du flux).
Total	entier	0	Nb total d'octets contenus dans le flux de données
Buffer	Byte[]	IND	Buffer destiné à recevoir les données lues dans le flux
Reponse	HttpWebRequest	IND	Objet utilisé pour envoyer une requête http vers le serveur web embarqué de la camera
Resp	WebResponse	IND	Objet utilisé pour récupérer la réponse http de la caméra
FluxDonnees	Stream	IND	Objet utilisé pour récupérer les données contenues dans la réponse

Résultat: Bitmap

debut

Reponse <- Resultat requete hhtp
Resp<- Reponse.GetResponse()
FluxDonnees<-Resp.GetResponseStream()
TantQue((NbOctetsLus<-FluxDonnees.Read(Buffer,Total,0)<>0)
Faire Total= Total + NbOctetsLus

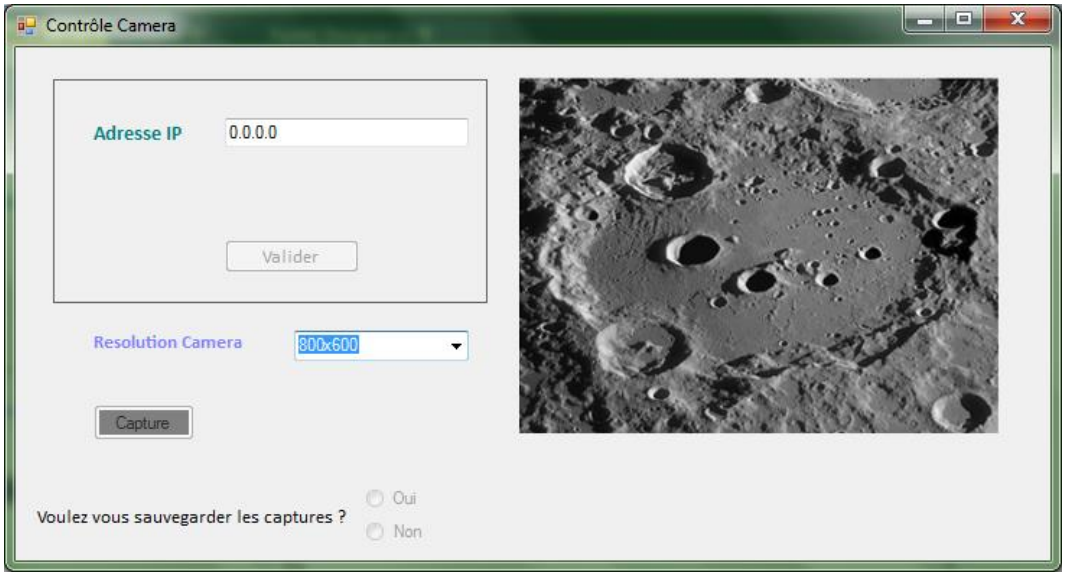
Fin TantQue
retourner (Image)
fin

Codez cette méthode, attention à bien respecter les noms de variables. Il est également important de bien veiller à utiliser un try ...catch.

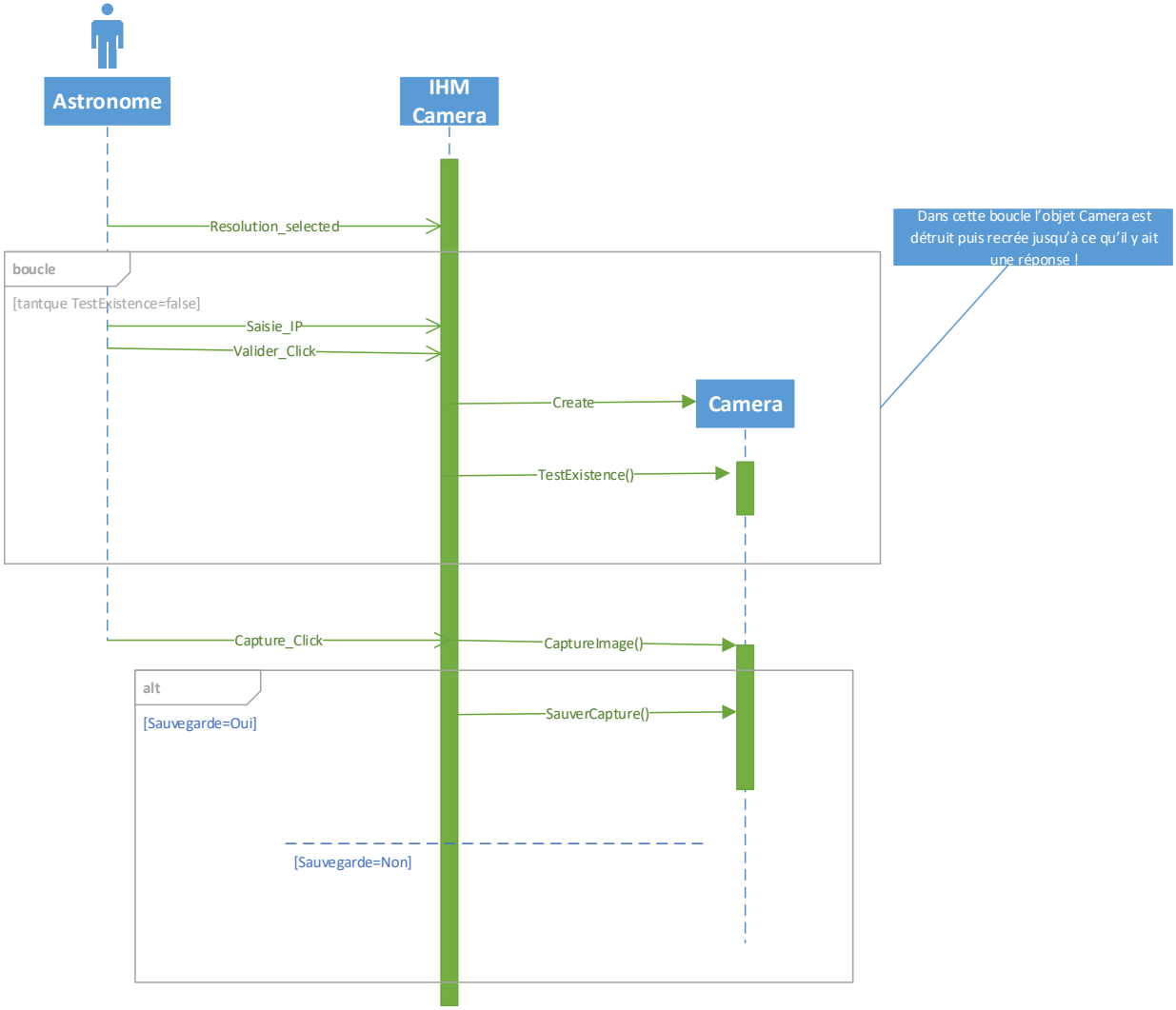
N'oubliez pas de sauvegarder votre projet ClasseCamera.

4. Classe IHMCamera.

La classe IHMCamera correspond à l'interface qui permet de gérer la caméra. Elle se présente de la façon suivante:



Dans votre EDI Visual Studio, créez un nouveau projet **Winform C#** que vous nommerez **IHMCamera**. Le diagramme de séquence correspondant au cas d'utilisation "Visualiser un objet du ciel profond" est donné ci dessous:



4.1. Construction de l'interface.

Construire votre interface "**Contrôle Camera**" conformément à la présentation faite au 4. La liste des composants utilisés pour réaliser cette interface est la suivante:

- 1 Panel (panel1)
- 3 Labels (Lab_ADRIP,Lab_Resolution, Lab_QuestionCapture)
- 2 Boutons (BoutAdrIP, BoutCapture)
- 2 Boutons Radio (RadioBoutonOui, RadioBoutonNon)
- 1 TextBox (TextBoxAdrIP)
- 1 ComboBox (ComboResolution)
- 1 PictureBox (VueTelescope)

4.2. Méthodes de l'interface.

Avant touches choses n'oubliez pas de déclarer les attributs de votre classe IHCamera.

4.2.1. Constructeur.

Il initialise les attributs de la classe, au départ toutes les références sont initialisées à **null**.

Compléter le code du constructeur.

4.2.2. Méthode associées à l'événement Load du formulaire (IHMCamera_load()).

Lorsque le formulaire est chargé, on désactive tous les composants à l'exception de **ComboResolution**, une message box invite l'utilisateur à choisir une résolution pour sa caméra. Lorsqu'une valeur est sélectionnée on met l'attribut **QualiteImage** à jour et on autorise la saisie de l'adresse IP en validant TextBoxAdrIP.

Compléter le code de cette méthode.

4.2.3. Méthode associée à l'événement saisie adresse IP.

Dans cette méthode on attend que l'utilisateur valide sa saisie par l'appui sur la touche entrée. On ne vérifie pas la validité de la saisie. Une fois que l'utilisateur valide l'adresse IP, l'attribut AdresseCamera est mis à jour avec l'adresse IP saisie. Le diagramme de séquence montre que lorsque l'adresse IP est validée on autorise la validation.

Compléter le code de cette méthode.

4.2.4. Méthode associée à l'événement BoutAdrIP_Click.

Dans cette méthode il faut tester la validité de l'adresse IP saisie par l'utilisateur, pour cela nous allons utiliser la méthode TestExistence() de la classe Camera. En observant le diagramme de séquence on voit que lorsque l'utilisateur valide l'adresse IP il y a instanciation d'un objet Camera puis appel de la méthode **TestExisence()** de l'objet instancié. Si la caméra existe, la méthode TestExistence() obtient une réponse et à ce moment on peut autoriser la capture. Sinon c'est que la Caméra Sélectionnée n'est pas disponible, il convient donc de détruire l'objet Camera (appel au garbage collector) et de recommencer l'opération. Une message box invitera à saisir une nouvelle adresse IP.

Compléter le code de cette méthode.

4.2.4. Méthode associée à l'événement BoutCapture_Click.

Dans cette méthode on appelle la méthode CaptureImage() de l'objet caméra, le composant VueTelescope est mis à jour avec l'image capturée. Il conviendra de gérer les formats non supportés (mise en place d'un try .. catch).

Le diagramme de séquence montre une alternative quand à la sauvegarde de l'image capturée.

Compléter le code de cette méthode.

4.2.5. Méthode SauverCature().

Cette méthode permet d'enregistrer les images dans le répertoire image courant de l'utilisateur. On utilisera la méthode save de la propriété image du composant **VueTelescope**.

Compléter le code de cette méthode.

5. Test.

Procédez au test de votre application en utilisant une de deux caméra mise à votre disposition et dont les adresses IP sont: 150.200.5.64 et 150.200.5.65

FIN.