Université Constantine 2-Abdelhamid Mehri
Faculté des NTIC
Département TLSI
UE : Programmation Orientée Composants (GL11)
Master : GL

TP 3 : .Net Remoting

Objectif :

L'objectif de ce TP est de réaliser une application à objets distribués en utilisant l'API Remoting et l'exemple de l'objet MailBox qui expose l'interface IMailbox suivante :

```
public interface {
      void sendMsg(Message msg);
      Message[] receiveMsg() ;
}
```

Question 1 : Définition de l'interface

Pour que le client ait connaissance des méthodes offertes par un objet distribué, une interface contenant leur déclaration doit être partagée entre le client et le serveur. Cette interface sera le seul élément commun entre le serveur et le client. L'objet exposé sur le serveur devra implémenter cette interface.

Dans .Net, l'interface est définie par une dll en suivant les étapes suivantes :

- Ouvrir Visual Studio .NET
- Créer une nouvelle solution vierge : "**AppliMailBox**" (aller à nouveau ->Projet-> Autres types de projets-> solution Vierge)
- Ajouter à la solution un nouveau projet C# de type "Librairie de classes", le nommer "**IRemote**" (clic droit sur la solution->Ajouter->Projet->Librairie de classes)
- Renommer le fichier Class1.cs en "**IMailBox.cs**" et remplacer le mot clé **class** par **interface**
- Remplacer le contenu de la classe Class1 par le code définissant les deux services SendMsg() et Receive Msg().
- Compiler le projet (clic droit sur le nom du projet ->build ou générer).

Question 2 : Définition de la classe d'implémentation

- Ajouter à la solution **AppliMailBox** créée précédemment un nouveau projet C# de type "Application Console" (clic droit sur la solution -> Ajouter -> Projet -> Application Console)
- nommer le projet "**Serveur**"
- Définir la dépendance entre le projet Serveur et l'interface en ajoutant dans le projet **Serveur** une référence au projet "**IRemote**"(clic droit sur le projet ->Ajouter->Référence-> Projet) et choisir IRmote.
- Ajouter un fichier de type classe appelé "**MailBox.cs**"
- Compléter le code d'implémentation des méthodes SendMsg et ReceiveMsg dans la classe **MailBox.cs** et définir un attribut privé de type Arraylist contenant la liste des messages déposés.

L'entête de la classe doit être défini comme suit :

```
public class MailBox : MarshalByRefObject, IRemote.IMailBox
```

On doit spécifier que MailBox implémente la classe **MarshalByRefObject** pour indiquer que le domaine d'application client ne contiendra qu'une référence à l'objet et non l'objet lui-même.

Question 3 : Configuration du serveur

- Ajouter une référence à "System.Runtime.Remoting" (clic droit sur le projet ->Ajouter->Référence-> plateforme) et choisir **System.Runtime.Remoting.**
- Renommer le fichier program.cs par "ServerMain.cs"
- Placer-y le code suivant permettant de démarrer le serveur et d'exposer l'objet MailBox en mode Singleton :

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using IRemote;
namespace Serveur {
class ServerMain{
   static void Main( string [] args){
```

```
   try{
      TcpChannel chnl = new TcpChannel(1234);
      ChannelServices.RegisterChannel(chnl, false);
      RemotingConfiguration.RegisterWellKnownServiceType(typeof(MailBox),
      “ObjMailBox", WellKnownObjectMode.Singleton);
      Console.WriteLine( "Serveur démarré..." );
      Console.ReadLine();
   } catch (Exception ex ) {
     Console.WriteLine("Serveur:Erreur d'initialisation !" + ex.Message);
   }   }    }   }
```

- Compiler le projet Serveur

Question 4 : Le client Sender

- Ajouter à la solution un nouveau projet C# de type "Application Console", et le nommer "**ClientSender**"
- Ajouter une référence au projet "**IRemote**"
- Ajoutez au projet une référence à " **System.Runtime.Remoting** "
- Placez-y le code suivant :

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using IRemote;
namespace ClientSender {
class Client  {
static void Main(string[] args) {
   try{
      TcpChannel chl = new TcpChannel();
      ChannelServices.RegisterChannel(chl, false);
      Console.WriteLine( "Client: Canal enregistré" );
      IMailBox obj = (IMailBox)Activator.GetObject(typeof(IMailBox),
      "tcp://localhost:1234/ObjMailBox");
      if (obj == null )
             Console.WriteLine("Problème de SERVEUR... " );
      else {
             Console.WriteLine( "Reference acquise de l'objet Singleton" );
      // Utilisation de l’objet : dépôt d’un message
      Console.ReadLine();
catch (Exception ex ) {
           Console.WriteLine( "ERREUR :" +  ex.Message);
 }} } }
```

- Compiler le projet ClientSender

Question 5 : Le client Receiver

- Répéter les mêmes étapes que celles faites pour le ClientSender, excepté que cette fois-ci on récupère la liste des messages déposés et on l’affiche.
- Compiler le projet ClientReceiver

Question 6 : Exécution de l’application

- Démarrer le Serveur, deux instances de ClientSender et une instance de ClientReceiver.
- Scénario d’exécution : dépôt d’un message par le premier ClientSender, dépôt de deux messages par le deuxième ClientSender et enfin lecture des messages par ClientRecevier.

Question 7 : Activation en mode singlecall

- Modifier le code du serveur pour publier l’objet en mode SingleCall, et refaire le même scénario de la question 6.
- Qu’est-ce que vous remarquez par rapport aux résultats de la question 6.

Question 8 : Activation coté client par une fabrique

- Définir l’interface IFabrique pour la création d’instances d’objets de type MailBox dans le projet IRemote.
- Implémenter cette interface dans le projet Serveur.
- Faire les modifications nécessaires dans les classes Serveur, ClientSender et ClientReceiver.
- Exécuter le même scénario que la question 6.
- Qu’est-ce que vous remarquez par rapport aux résultats de la question 6.