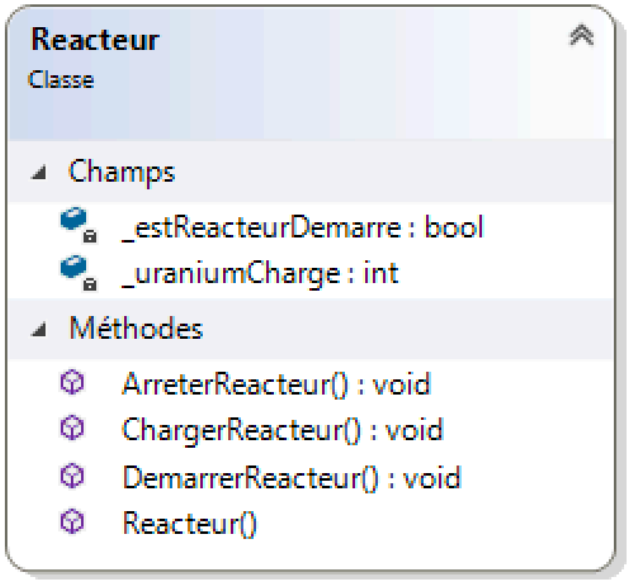
1 Contexte

Une grande entreprise Française nous demande de réaliser un logiciel de contrôle de réacteur nucléaire. Le logiciel devra gérer l'ajout des barres d'uranium, la mise en route du réacteur ainsi que son arrêt. De plus deux salles des commandes partageront le contrôle du réacteur.

Un réacteur ne peut démarrer que s'il est chargé avec 50 kg d'uranium.
Le chargement du réacteur ajoute 50 kg d'uranium lorsque le réacteur n'est pas démarré.
L'arrêt du réacteur ne peut se faire qui s'il est démarré.

2 Préparation

Créez une projet application Console.
Crééz la classe Reacteur :



Codez la classe Reacteur de façon conventionnelle
les méthodes qui renvoient les messages suivant :

```
ChargerReacteur : Le réacteur contient xx kg d'Uranium
DemarrerReacteur : Le réacteur est démarré...
ArreterReacteur : Le réacteur est arrêté...
```

3 Codage du singleton

Nous avons deux postes de commande pour piloter un seul réacteur. Mais il y a soucis car si chacune d'elle crée une instance de réacteur nous allons nous retrouver à charger deux fois le même réacteur…

Il faut s'assurer qu'une et une seule instance de réacteur puisse être créée…c'est le principe du design pattern singleton !

Déjà il ne faut pas que la classe Réacteur puisse être instanciée depuis l'extérieur, le constructeur sera donc privé. Par contre la classe non instanciable ! Pour régler ce souci on crée une variable statique privée de type Reacteur dans la classe Reacteur associée à une méthode statique d'obtention de la variable :

```
public class Reacteur
   {
       // La variable de type Réacteur
       private static Reacteur _instance;

       private int _uraniumCharge;
       private bool _estReacteurDemarre;

       private Reacteur()
       {
           this._uraniumCharge = 0;
           this._estReacteurDemarre = false;
       }

       /// <summary/>
       /// Méthode d'obtention de l'instance de réacteur
       /// <summary/>
       /// <returns> </returns>
       public static Reacteur GetReacteur()
       {
          // Si le réacteur n'est pas instancié on crée une nouvelle instance, le constructeur
          // étant privé seule un appel depuis une méthode interne peut fonctionner.
           if (_instance == null)
               _instance = new Reacteur();
           return _instance;
       }
}
```

Instanciez deux postes de commande à l'aide du singleton, vérifiez le bon fonctionnement du code suivant:

```
// Le poste de commande 1 charge et lance le réacteur
posteCommande1.ChargerReacteur();
posteCommande1.DemarrerReacteur();

// Le poste de commande 2 essaye de charger et d'arrêter le réacteur
// Le chargement ne peut se faire vu que le réacteur est déjà chargé !
posteCommande2.ChargerReacteur();
posteCommande2.ArreterReacteur();
Console.ReadLine();
```

4 Amélioration

Pour éviter l'instanciation simultanée on utilisera une variable statique en readonly :

```
public class Reacteur
   {
       // La variable de type Réacteur
       private static readonly Reacteur _instance = new Reacteur();

       private int _uraniumCharge;
       private bool _estReacteurDemarre;

       private Reacteur()
       {
           this._uraniumCharge = 0;
           this._estReacteurDemarre = false;
       }
       public static Reacteur GetReacteur()
       {
           return _instance;
       }
}
```

5 Application

Dans vos différents TP, mettez en place un singleton pour la connexion à la base de données.