Injection de dépendance

1 Gestion de dépendance sans outil

On met en place pour le logiciel de caisse d'un supermarché une classe TicketDeCaisse dont le rôle sera de cumuler les achats lors d'un passage en caisse, puis d'éditer le ticket de caisse. Le prix des articles peut varier en fonction du profil du client(famille nombreuse, sélection de produits à prix réduits), il faut donc le charger.

On prévoit une méthode getParamClient qui se charge de cette tâche, et retourne un ParamClient. Pour l'exemple, ParamClient est une classe qui ne contient qu'une chaîne de caractères. Dans getParamClient, on doit faire appel à la classe ChargementParamètres, qui à partir du numéro de client, récupère le compte du client en se connectant à un système central, récupère les paramètres, et peut les retourner.

Dans ChargementParamètres, on aura une méthode permettant de récupérer un compte client, et une méthode retournant les paramètres du client (on ne fera pas d'accès distant à un serveur central, sauf si vous en avez envie, simulez celà par tout artifice de votre choix).

Question 1. Mettez en place le code adéquate.

On constate que la méthode getParamClient doit créer une instance de ChargementParamètres, et en est donc fortement dépendante. Si le constructeur de ChargementParamètres change, la méthode changera aussi.

Nous allons voir 3 moyens simples pour modifier cette dépendance : en l'injectant par mutateurs/champs, par constructeur, et par utilisation d'interfaces.

Question 2. Pour éviter que la classe TicketDeCaisse ne soit responsable de l'instanciation de la dépendance, une solution naturelle est d'introduire un champs (attribut) et un mutateur (accesseur en écriture) sur ce champs. Ainsi, l'instanciation de la dépendance devient à la charge de l'utilisateur de la classe TicketDeCaisse, qui appelera le mutateur. Essayez cette solution.

Pour bien faire, il faut prendre des précautions car le mutateur peut ne pas avoir été appelé, et il nous faut éviter les appels à de références null ... L'utilisateur doit systématiquement appeler le mutateur avant d'appeler la méthode getParmClient, ce qui est difficile à imposer et pas intuitif.

Question 3. Pour éviter que le client n'omette l'appel au mutateur, une autre façon d'injecter la dépendance est de le faire par le biais d'un constructeur : le constructeur de TicketDeCaisse requiert une instance de ChargementParamètres. Mettez en place cette solution. On remarque que l'utilisateur ne peut plus oublier de fournir l'instance de ChargementParamètres, puisqu'il en a besoin à la construction.

Question 4. La dépendance est toutefois toujours présente, et en cas de changement d'implémentation pour le chargement de paramètres, la classe TicketDeCaisse sera impactée. Pour limite l'impact, on peut alors utiliser une interface, en plus de l'injection soit par mutateur, soit par constructeur. L'interface IChargementParametres déclare les méthodes publiques propres au chargement de paramètres, et ChargementParametres les implémente. Dans TicketDeCaisse, on remplace les références à ChargementParametres par des références à l'interface. Testez. On note que maintenant on a un couplage plus faible, l'implémentation du chargement de paramètres peut être changée sans impact sur le TicketDeCaisse.

La dépendance est maintenant completement reléguée vers l'utilisateur. Toutefois, l'instanciation devra bien avoir lieu, potentiellement plusieurs fois.

2 Injection de dépendance avec conteneur IOC

Pour faciliter l'utilisation de TicketCaisse par le client, on souhaite lui éviter l'instanciation directe. Ainsi on paramètre une fois que tel Client utilisera tel type de ChargementParam, et ensuite le client, quand il en a besoin, demande au conteneur de lui donner une instance. Nous allons utiliser Ninject.

2.1 installer Ninject dans le projet

Ninject se déploie dans un projet particulier, car la version de Ninject dépend de la version de .net utilisée dans le projet.

— Sur la solution : ouvrir le menu "gérer les packages NuGet pour la solution", puis onglet parcourir, choisir Ninject, sélectionner le projet souhaité, puis valider.

2.2 fonctionnement global de Ninject

Ninject travaille à partir d'un objet de type Kernel. Dans un kernel, on peut charger des modules, ce sont les modules qui contiennent les informations de résolution de type.

Créer un module. Un module est une classe, par exemple MonModule, qui implémente StandardModule (ou tout autre module Ninject). StandardModule est dans le namespace Ninject.Modules. Au chargement d'un

module, sa méthode public void Load() est appelée, il faut ici la redéfinir. A l'intérieur, on fait les associations entre une interface et une classe. Ainsi, quand le client réclamera un objet typé par cette interface, Ninject pourra retourner une instance de la classe. L'association se fait ainsi :
`Bind<IBidule>().To<Bidule>();`
où Bidule implémente IBidule.

Créer un Kernel et lui associer un module : `IKernel kernel = new StandardKernel(new MonModule());`
IKernel et StandardKernel sont dans le namespace Ninject.

Demander une instance au kernel : IBidule bidule = kernel.Get<IBidule>() ;

On remarque que Ninject, contrairement à d'autres conteneurs IOC, ne travaille pas via des fichiers de configuration.

2.3 Application aux tickets de caisse

— Reprenez la version avec les interfaces.
— Développez une nouvelle implémentation de IChargementParametres qui simule toujours le chargement dse paramètres.
— Créez un module capable d'associer l'une ou l'autre des implémentations à IChargementParametres, en fonction d'un paramètre (booléen par exemple, ou entier ou autre).
— Créez un kernel avec le module, demandez un IChargementParametres, utilisez-le pour créer un ticket de caisse.
— Vérifier que selon le paramètre choisi pour créer le module, on passe d'une implémentation à l'autre de manière totalement transparente.

3 Exemple de Martin Fowler

Question 5. Reprenez l'exemple de Martin Fowler (https ://martinfowler.com/articles/injection.html) en .net.

Question 6. Comparez les modes d'injection de Ninject (https ://github.com/ninject/Ninject/wiki/Injection-Patterns) avec ceux exposés par Martin Fowler.