Test unitaire
Les `new` dans le code applicatif

Master info S2
C-TD SVL – 2017-2018

Les exemples sont donnés pour un langage typé statiquement mais valent pour tous les langages..

Quand une méthode d'une classe `ATester` utilise un `new Ximpl()` dans le code :
— l'application dépend de l'implantation `Ximpl` ;
— il n'est pas possible de **tester** l'application indépendamment de `Ximpl`.

Patron d'injection des dépendances

Dans le cas où l'application n'a besoin que d'un nombre prévu d'instances déterminées de `Ximpl` (cas par exemple d'un jeu de dé qui utiliserait 2 instances de dé), on utilise le patron d'injection des dépendances en s'assurant de dépendre d'une interface `X`. Les instances de `Ximpl` sont créées dans la phase de construction du système (par exemple dans le constructeur de la classe mère, ou dans le main). Elles sont ensuite utilisées pendant l'exécution du système. Les tests utilisent des mocks de `X`.

Patron Fabrique

Dans le cas où l'application crée les instances de `X` pendant l'exécution (cas par exemple du gestionnaire de log qui crée des messages, de la bibliothèque qui crée des emprunts) il n'est pas possible de créer ces objets a priori. Pour **contrôler** la création des instances, on peut utiliser le patron Fabrique. Une fabrique de `Ximpl` contient une méthode qui retourne une instance de `Ximpl`. On injecte dans la classe `ATester` une dépendance à une interface `FabriqueX`. Les tests utilisent à la fois des mocks pour `fabriqueX` et `X`.

Dans les tests :

```
X x = mock(X.class);
FabriqueX fabrique = mock(FabriqueX.class);
ATester atester = new ATester(..., fabrique);
when(fabrique.creerX(...)).thenReturn(x);
... atester.methodeATester();
verify(fabrique.creerX(...));
... // autres oracles éventuels portant sur x
```

Dans le code :

```
// quelquepart
FabriqueX fabrique = new FabriqueXImpl();
// ailleurs
ATester atester = new ATester(..., fabrique);
```

Patron Singleton Dans le cas où la fabrique est implantée grâce au patron Singleton, et où la classe `ATester` utilise dans son code :

```
FabriqueX fabrique = FabriqueXImpl.getInstance();
```

alors le code n'est pas testable, la dépendance à la fabrique est cachée et la fabrique n'est pas contrôlable.

Que faire de ce new ?

L'idée est de placer la création des objets dans des classes dont la *responsabilité* de créer ces objets est bien identifiée :
— dans la phase de construction de l'application, ces objets étant ensuite utilisés dans l'application via les dépendances (passage de paramètre dans les constructeurs ou les accesseurs en écriture) ;
— dans une usine ;
— dans une classe dont c'est le rôle, par exemple construction d'un conteneur Java dans une classe `CollectionY`.

et de contrôler les objets créés. Couplée à l'utilisation d'interfaces, cette approche permet :
— de ne pas faire dépendre votre code d'implantations : si vous choisissez de remplacer les `Ximpl` par des `XimplBis`, il suffira de modifier le code qui construit les objets ou de changer de factory (par opposition à chercher dans quelles classes et à quel endroit on a construit un `XImpl`) ;
— de pouvoir tester le code en isolation en remplaçant les objets de l'application par des mocks, ce qui permet aussi de simplifier les objets utilisés pour le test.