TD-TP : Principes « SOLID »

Exemple de mise en œuvre du Principe de Simple Responsabilité :
le mécanisme pour **Déporter une méthode**

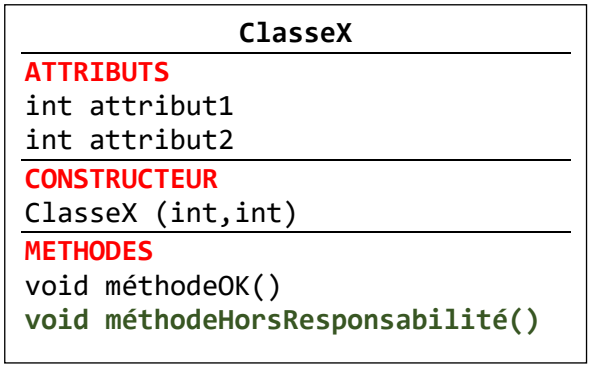
1. SIMPLE RESPONSABILITE : rappel théorique

Le Principe de **S**imple Responsabilité vise à ce qu'une classe n'ait qu'une seule responsabilité. Selon ce principe, lorsqu'une classe a trop de responsabilités, à savoir des méthodes qui ne caractérisent pas l'objet, **on déporte** vers une classe destination, un ensemble de méthodes de la classe d'origine. Ainsi, la classe d'origine est déchargée des méthodes sélectionnées, et c'est la classe destination qui en assume les responsabilités correspondantes.

2. DEPORTER UNE METHODE : le mécanisme

Pour se préparer à une telle pratique (cf. déporter des méthodes) vous allez programmer un cas école qui déporte dans une nouvelle classe, une méthode qui se trouve dans une classe origine donnée.

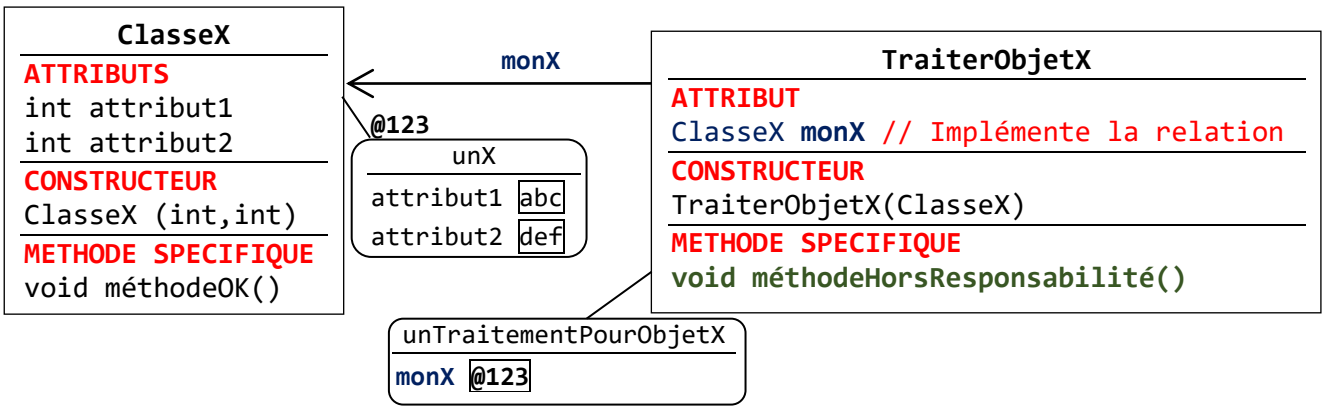
Etant donné le diagramme de classe ci-dessous dans lequel la `ClasseX` comporte entre autres, deux attributs et une `méthodeHorsResponsabilité()` qui n'a pas lieu d'être dans cette classe car elle ne caractérise pas l'objet.



Une fonction `main()` qui utiliserait une telle classe le ferait selon la section de code ci-dessous :

```
1. ClasseX unX = new ClasseX (10, 20);    // Déclare et crée un objet de ClasseX
2.
3. unX.attribut1 = ...;                   // Accède aux membres via l'objet
4. unX.attribut2 = ...;
5. unX.methodeOK();
6. unX.methodeHorsResponsabilite();
```

Pour satisfaire au principe de **S**imple Responsabilité, il s'agit de retirer la `méthodeHorsResponsabilité()` et la déporter dans une classe destination `TraiterObjetX`, conformément au schéma de classe UML ci-dessous.



Travail à faire

1. Sur la base des deux classes ainsi définies, quelle serait la section de code à produire pour satisfaire un comportement équivalent à celui de la section de code du `main()` précédent ?

3. DEPORTER UNE METHODE : exercice

Etant donnée la classe `Date` suivante dont on veut sortir la méthode **`int toInt()`** considérée comme hors responsabilité.

Date
ATTRIBUTS int année int mois int jour
CONSTRUCTEUR Date (int,int,int)
ENCAPSULATION get&set année, mois, jour
METHODE USUELLE String toString()
METHODE SPECIFIQUE **int toInt() {**   return (10000*année + 100*mois + jour) **}**

Travail à faire

Sur la base du schéma de solution vu dans la section **2. DEPORTER UNE METHODE** pour isoler une (des) méthode(s) dans une autre classe, proposer une solution qui soit spécifique à ce cas concret. A savoir :

2. Proposer un schéma de classe UML correspondant à la solution.
3. Quel est le code de la classe **`TraiterDate`** et notamment le code de la méthode **`toInt()`** ?
4. Quel est le code de la méthode **`main()`** de la classe **`TesterMéthodeDéportée`** qui crée une date, crée un objet capable de traiter cette date, puis demande à cet objet d'exécuter le traitement (cf. **`toInt()`** dans notre cas) dont il a la responsabilité ?
5. En TP, pour expérimenter la mécanique de la Méthode Déportée, comme mise en œuvre possible du principe de Simple Responsabilité, créer le projet Eclipse **`8.SOLIDdéporterMéthode`**, avec les 3 classes **`Date`**, **`TraiterDate`** et **`TesterMéthodeDéportée`**, et vérifier le bon fonctionnement.