

TP JUnit

Le but de ce TP est de construire des classes Java et de les tester avec JUnit. Dans la première partie on construit une classe qui modélise des sommes d'argent et un premier test vérifiant certaines méthodes de cette classe. Dans la seconde partie on enrichit cette classe et on construit des tests tout en utilisant des notions avancées de JUnit. La troisième et dernière partie décrit la classe modélisant un porte-monnaie. Cette classe utilise la classe précédente et des tests sont construits pour la classe porte-monnaie.

Première partie Le premier test d'une classe

1°) Sous Eclipse, construire un projet Java et écrire la classe `SommeArgent` dans le package `junit.monprojet`:

```
package junit.monprojet;
public class SommeArgent {
    private int quantite;
    private String unite;

    public SommeArgent(int amount, String currency) {
        quantite = amount;
        unite = currency;
    }

    public int getQuantite() {
        return quantite;
    }

    public String getUnite() {
        return unite;
    }

    public SommeArgent add(SommeArgent m) {
        return new SommeArgent(getQuantite()+m.getQuantite(), getUnite());
    }
}
```

Les objets de cette classe sont des quantités d'argent dans une certaine unité (ou monnaie) comme \$US100.65, £45.87, 100,65 CHF, 54,98 €, etc.

2°) On veut écrire une méthode de test qui vérifie le "bon" codage de la méthode d'addition de 2 sommes d'argent. Cette méthode de test va construire deux sommes d'argent, en faire la somme et vérifie que cette somme est correcte.

Pour cela, il faut enrichir la classe `SommeArgent` de la méthode `equals()` qui définit l'égalité (ou l'équivalence) de deux objets et qui redéfinit la méthode `equals()` de la classe `java.lang.Object`

Deux sommes d'argent sont égales si elles sont de même unité et de même quantité. Écrire cette méthode `equals()` de la classe `SommeArgent`. Elle doit avoir pour signature :

```
public boolean equals(Object anObject)
```

(et pas autrement ;-)).

Remarque 1 : pour bien faire, il faudrait alors aussi redéfinir la méthode `hashCode()`. On pourra s'en passer ici.

Remarque 2 :

Il faut bien définir la méthode `equals()` comme indiquée et pas :

```
public boolean equals(SommeArgent anObject)
```

qui ne redéfinit alors pas la méthode

```
public boolean equals(Object anObject)
```

qui, elle, est utilisée dans les méthodes statique de la classe `Assert` (`assertEquals()`, `assertTrue()`, etc.) utilisées dans les tests.

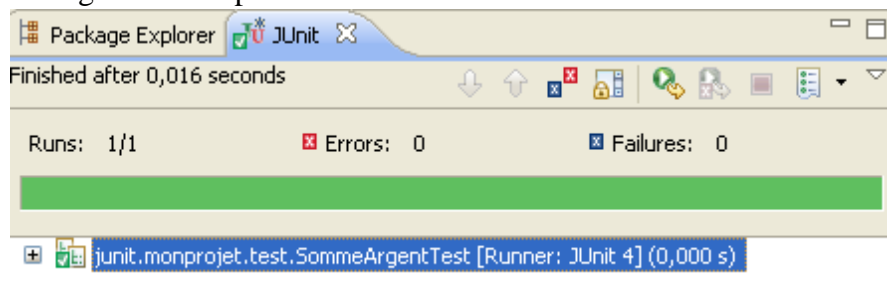
3°) Créer un package `junit.monprojet.test` et dans ce package une classe de tests JUnit 4. Si vous avez une version d'Eclipse récente, cette classe de tests est facilement construite par `New | JUnit Test Case`

5°) Écrire la méthode de test qui construit deux sommes d'argent, en fait la somme et vérifie qu'elle est correcte. Le corps de cette méthode est :

```
SommeArgent m12CHF= new SommeArgent(12, "CHF"); // (1)
SommeArgent m14CHF= new SommeArgent(14, "CHF");
SommeArgent expected = new SommeArgent(26, "CHF");
SommeArgent result = m12CHF.add(m14CHF); // (2)
Assert.assertTrue(expected.equals(result)); // (3)
```

Remarque 1 :

6°) Lancer ce test. Vous devez obtenir :
"Keep the bar green to keep the code clean."



Seconde partie

D'autres tests pour la classe SommeArgent

7°) Euh, en fait, on a oublié de tester la méthode `equals()` de la classe `SommeArgent`. Écrire une méthode de test pour cette méthode `equals()`. Le corps de la méthode de test peut être :

```
SommeArgent m12CHF= new SommeArgent(12, "CHF");
SommeArgent m14CHF= new SommeArgent(14, "CHF");
SommeArgent m14USD= new SommeArgent(14, "USD");

Assert.assertTrue(!m12CHF.equals(null));
Assert.assertEquals(m12CHF, m12CHF);
Assert.assertEquals(m12CHF, new SommeArgent(12, "CHF")); // (1)
Assert.assertTrue(!m12CHF.equals(m14CHF));
Assert.assertTrue(!m14USD.equals(m14CHF));
```

8°) Dans ces deux méthodes de tests il y a des initialisations communes. Écrire un code, dans la classe de test, qui regroupe ces initialisations dans une seule méthode externe. Vérifier que ces initialisations sont bien effectuées avant chaque lancement de méthode de tests.

10°) De manière similaire, si après chaque exécution de méthodes de tests, on veut exécuter un code commun (de désallocation par exemple), comment doit-on procéder ? Vérifier qu'après le lancement de chaque méthode de test, on passe bien dans la méthode que vous venez d'écrire. On voudrait obtenir une exécution des tests qui affichent :

```
1ime passage avant exécution d'une méthode de test
1ime passage APRES exécution d'une méthode de test
2ime passage avant exécution d'une méthode de test
2ime passage APRES exécution d'une méthode de test
```

11°) Le code donné ci-dessus pour la méthode `add()` (question 1°) qui ajoute deux sommes d'argent, n'est pas vraiment correct : que se passe-t-il si ces deux sommes ne sont pas de la même unité ? On se propose dans ce cas de faire lever une exception `UnitéDistincteException` et de modifier alors la méthode en la signant :

```
public SommeArgent add(SommeArgent m) throws UnitéDistincteException
```

Il faut pour cela, ajouter dans le projet `junit.monprojet` cette classe `Exception` :

```
public class UnitéDistincteException extends Exception {
    private SommeArgent somme1, somme2;

    public UnitéDistincteException(SommeArgent sa1, SommeArgent sa2) {
        somme1 = sa1;
        somme2 = sa2;
    }

    public String toString() {
        return "unité distincte : " + somme1.getUnite() + " != " +
somme2.getUnite();
    }
}
```

```
}
```

et modifier la méthode `add()` par :

```
public SommeArgent add(SommeArgent m) throws UniteDistincteException {  
    if (!m.getUnite().equals(this.getUnite())) {  
        throw new UniteDistincteException(this, m);  
    }  
    else return new SommeArgent(getQuantite()+m.getQuantite(), getUnite());  
}
```

11.1°) Comme la méthode `add()` a été modifiée, corriger les éventuelles erreurs qui sont apparues dans votre projet.

11.2°) Ecrire une méthode de test, qui construit deux objets de la classe `SommeArgent` avec des unités distinctes et vérifier que, dans ce cas, une exception `UniteDistincteException` est levée.

Indication : Il faut enrichir l'annotation `@Test`.

11.3°) Vérifier que votre classe `SommeArgent` "passe" tous les tests et, qu'en outre, lorsqu'on essaie d'ajouter deux sommes d'argent d'unité distincte, l'exception `UniteDistincteException` est bien levée. Vous devez obtenir la barre verte de succès de bon passage dans les tests.

Troisième partie : un porte-monnaie

On veut regrouper ces diverses sommes d'argent dans un porte-monnaie et, pour cela construire une nouvelle classe `PorteMonnaie`. Une première version de cette classe peut être :

```
import java.util.HashMap;

public class PorteMonnaie {
    private HashMap<String, Integer> contenu;

    public HashMap<String, Integer> getContenu() {
        return contenu;
    }

    public PorteMonnaie() {
        contenu = new HashMap<String, Integer>();
    }

    public void ajouteSomme(SommeArgent sa) {
        // à définir cf. question suivante
    }
}
```

12°) On veut ajouter dans cette classe, la possibilité d'ajouter des sommes d'argent. Les spécifications du porte-monnaie sont :

"Si un ajoute 12 € et qu'il n'y a pas déjà d'euro dans le porte-monnaie, cette somme est simplement mise. S'il y avait déjà 10 €, il y aurait désormais 22 €."

Coder cette spécification dans la méthode `ajouteSomme()`.

Remarque : vous aurez peut-être besoin des méthodes utilitaires de la classe `HashMap` accessible à l'URL

<http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html>

13°) Ecrire une méthode de `public String toString()` qui affiche le contenu du porte-monnaie. On pourra commencer par écrire la méthode `public String toString()` de la classe `SommeArgent`.

14°) Construire, dans le package de test, une classe de test pour la classe `PorteMonnaie`, contenant une méthode de tests qui vérifie la bonne cohérence de la méthode `ajouteSomme()`. Par exemple un porte-monnaie dans lequel on a mis 5 euro, puis ajouter 5 autre euro est "égal" à un autre porte-monnaie contenant 10 euro.

On pourra par exemple écrire la méthode

```
public boolean equals(Object obj)
```

dans la classe `PorteMonnaie` qui retourne `true` si l'instance et le porte-monnaie passé en paramètres ont les mêmes devises en même quantité.