TP1 Test Logiciel

Travail par : Idris SADDI

https://github.com/idris-saddi/TP1-Test-Logiciel

Première Partie

1. Création de la Classe `SommeArgent` :

- Une classe `SommeArgent` a été créée dans le package `junit.monprojet`.
- Cette classe contient :
  - Deux attributs :
    - `quantite` : un entier représentant la valeur monétaire.
    - `unite` : une chaîne représentant la devise (ex. "CHF").
  - Un constructeur pour initialiser les deux attributs.
  - Une méthode `add(SommeArgent)` pour additionner deux sommes d'argent ayant la même unité.
  - Une méthode `equals(Object)` redéfinie pour comparer deux objets `SommeArgent`.

Code Principal :

```
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) {
        if (!this.unite.equals(m.getUnite())) {
            throw new IllegalArgumentException("Les unités doivent être
identiques pour l'addition.");
        }
        return new SommeArgent(getQuantite() + m.getQuantite(), getUnite());
    }
```

```
    @Override
    public boolean equals(Object anObject) {
        if (this == anObject) return true;
        if (anObject == null || getClass() != anObject.getClass()) return
false;

        SommeArgent that = (SommeArgent) anObject;

        return quantite == that.quantite && unite.equals(that.unite);
    }
}
```

2. Création de la Classe de Test JUnit :

- Un nouveau package `junit.monprojet.test` a été créé.
- Une classe de test `SommeArgentTest` a été ajoutée pour tester la méthode `add`.
- La méthode de test vérifie qu'additionner deux sommes d'argent donne un résultat correct.

Code du Test :

```
package junit.monprojet.test;

import junit.monprojet.SommeArgent;
import org.junit.Assert;
import org.junit.Test;

public class SommeArgentTest {

    @Test
    public void testAddition() {
        SommeArgent m12CHF = new SommeArgent(12, "CHF");
        SommeArgent m14CHF = new SommeArgent(14, "CHF");
        SommeArgent expected = new SommeArgent(26, "CHF");
        SommeArgent result = m12CHF.add(m14CHF);
        Assert.assertTrue(expected.equals(result));
    }
}
```

3. Résultat des Tests :

- Le test a été exécuté avec succès dans Eclipse.
- Le message suivant a été obtenu dans la console JUnit :
  "All tests passed."
  La barre verte indique que la méthode `add` fonctionne correctement.

Deuxième Partie

7°) Écrire une méthode de test pour `equals()`

Une méthode de test a été ajoutée pour tester le comportement de `equals()` :

Code de Test :

```
@Test
public void testEquals() {
    SommeArgent m12CHF = new SommeArgent(12, "CHF");
    SommeArgent m14CHF = new SommeArgent(14, "CHF");
    SommeArgent m14USD = new SommeArgent(14, "USD");

    Assert.assertTrue(!m12CHF.equals(null)); // Test null
    Assert.assertEquals(m12CHF, m12CHF); // Test identité
    Assert.assertEquals(m12CHF, new SommeArgent(12, "CHF")); // Test égalité
    Assert.assertTrue(!m12CHF.equals(m14CHF)); // Quantité différente
    Assert.assertTrue(!m14USD.equals(m14CHF)); // Unité différente
}
```

Interprétation de la Dernière Ligne : La dernière ligne teste que deux objets `SommeArgent` avec des unités différentes (par exemple, "USD" et "CHF") ne sont pas considérés comme égaux.

8°) Regrouper les initialisations communes

Les initialisations communes ont été regroupées dans une méthode annotée avec `@Before` :

Code Modifié :

```
private SommeArgent m12CHF;
private SommeArgent m14CHF;
private SommeArgent m14USD;

@Before
public void setUp() {
    System.out.println("Avant chaque test");
    m12CHF = new SommeArgent(12, "CHF");
    m14CHF = new SommeArgent(14, "CHF");
    m14USD = new SommeArgent(14, "USD");
}
```

Test Vérifié : Avant chaque test, la méthode `setUp()` est bien appelée, initialisant les objets nécessaires.

9°) Nom des objets créés pour chaque méthode de test

Ces ensembles d'objets créés à chaque exécution de méthode de test sont appelés **"Fixtures"**.

10°) Code commun exécuté après chaque test

Pour exécuter du code commun après chaque test, une méthode annotée avec `@After` a été ajoutée :

Code :

```
@After
public void tearDown() {
    System.out.println("Après chaque test");
}
```

Résultat dans la console :

```
Avant chaque test
Après chaque test
Avant chaque test
Après chaque test
```

11°) Gérer les unités distinctes avec `UniteDistincteException`

11.1 Modification de la méthode `add` :

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

Ajout de la Classe `UniteDistincteException` :

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

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

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

11.2 Méthode de test pour l'exception :

```
@Test(expected = UniteDistincteException.class)
public void testAddException() throws UniteDistincteException {
    SommeArgent m12CHF = new SommeArgent(12, "CHF");
    SommeArgent m14USD = new SommeArgent(14, "USD");
    m12CHF.add(m14USD); // Devrait lever une exception
}
```

11.3 Vérification des Tests :

Tous les tests ont été exécutés avec succès. La barre verte dans JUnit confirme que :

- Les tests pour `equals()` sont corrects.
- L'exception `UniteDistincteException` est levée lorsqu'on additionne des unités distinctes.

Troisième Partie

12°) Implémentation de `ajouteSomme()`

Spécifications :

- Si la devise est déjà présente dans le porte-monnaie, la quantité est mise à jour.
- Sinon, la somme est ajoutée comme une nouvelle entrée.

Code de la méthode `ajouteSomme()` :

```
public void ajouteSomme(SommeArgent sa) {
    String unite = sa.getUnite();
    int quantite = sa.getQuantite();
    if (contenu.containsKey(unite)) {
        contenu.put(unite, contenu.get(unite) + quantite); // Mise à jour
    } else {
        contenu.put(unite, quantite); // Nouvelle entrée
    }
}
```

13°) Méthode `toString()`

Méthode `toString()` pour la classe `SommeArgent` :

```
@Override
public String toString() {
    return quantite + " " + unite;
}
```

Méthode `toString()` pour la classe `PorteMonnaie` :

```
@Override
public String toString() {
    StringBuilder contenuStr = new StringBuilder("Contenu du porte-monnaie
:\n");
    for (String unite : contenu.keySet()) {
        contenuStr.append(contenu.get(unite)).append("
").append(unite).append("\n");
    }
    return contenuStr.toString();
}
```

14°) Tests pour la classe `PorteMonnaie`

Ajout de la méthode `equals()` pour `PorteMonnaie` :

```
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    PorteMonnaie other = (PorteMonnaie) obj;
    return contenu.equals(other.contenu);
}
```

Classe de test pour `PorteMonnaie` :

```
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

public class PorteMonnaieTest {
    private PorteMonnaie porteMonnaie1;
    private PorteMonnaie porteMonnaie2;

    @Before
    public void setUp() {
        porteMonnaie1 = new PorteMonnaie();
        porteMonnaie2 = new PorteMonnaie();
    }

    @Test
    public void testAjouteSomme() {
        porteMonnaie1.ajouteSomme(new SommeArgent(5, "EUR"));
        porteMonnaie1.ajouteSomme(new SommeArgent(5, "EUR"));
        porteMonnaie2.ajouteSomme(new SommeArgent(10, "EUR"));

        Assert.assertEquals(porteMonnaie1, porteMonnaie2);
    }

    @Test
    public void testAjouteSommeDifferentDevises() {
        porteMonnaie1.ajouteSomme(new SommeArgent(10, "USD"));
        porteMonnaie1.ajouteSomme(new SommeArgent(5, "EUR"));

        porteMonnaie2.ajouteSomme(new SommeArgent(10, "USD"));
        porteMonnaie2.ajouteSomme(new SommeArgent(5, "EUR"));

        Assert.assertEquals(porteMonnaie1, porteMonnaie2);
    }

    @Test
```

```
    public void testToString() {
        porteMonnaie1.ajouteSomme(new SommeArgent(15, "CHF"));
        String expected = "Contenu du porte-monnaie :\n15 CHF\n";
        Assert.assertEquals(expected, porteMonnaie1.toString());
    }
}
```

Résultats des Tests :

1. Les tests confirment que la méthode `ajouteSomme()` respecte les spécifications :
    - Les devises identiques sont cumulées correctement.
    - Les devises différentes sont ajoutées comme des entrées distinctes.
2. La méthode `toString()` affiche correctement le contenu du porte-monnaie.
3. La méthode `equals()` garantit la cohérence des comparaisons entre deux objets `PorteMonnaie`.