Corrigé TP n°5: Algorithmique – Programmation FIP (ING39)

V. Aponte

Septembre 2015

Les contrats manquant dans l'interface fournie

Nous complétons les contrats manquants dans l'interface fournie `LivretA`. Nous introduisons une nouvelle exception`OperationInterdite` qui sera levée par toute opération invoquée sur un compte fermée autre qu'un accesseur ou l'opération `ouvrir`. Nous réproduisons l'interface complète.

```
public interface LivretA {
 /**
   * Invariant (contrat) d'état interne:
   *     <pre>
   *              this.getPlafond() > 0
   *              &&   si this.estOuvert()     alors (0 <= this.getSolde() <= this.getPlafond())
   *              &&   si !(this.estOuvert())  alors this.getSolde() = 0
   *     </pre>
   */
public interface LivretA {

  /**
   * Contrat pour les constructeurs: tout compte crée respecte l'invariant d'état.
   * Une tentative de création de compte incohérent lève l'exception IllegalArgumentException.
   * Un compte peut être crée fermé, mais avec solde = 0
   * Un compte crée ouvert a forcément un solde positif.
   */

  /** Contrats des méthodes **/

  /**
   * Retourne le nom du titulaire du compte.
   * @return
   */
  public String getNomTitulaire();

  /**
   * Retourne la valeur maximale autorisée pour le solde courant.
   * @return un double strictement positif.
   * postcondition: this.getPlafond() > 0
   */
  public double getPlafond();

  /**
   * Retourne le solde courant du compte.
   * @return un double positif ou nul
   * postcondition: this.getSolde() >=0
   */
```

```
  public double getSolde();

  /**
   * Determine si un compte est dans l'état ouvert.
   * @return  boolean true si le compte est ouvert.
   */
  public boolean estOuvert();

  /**
   * Ajoute le montant m au solde courant.
   * @param m montant du dépôt
   * precondition : m >= 0
   * postcondition : this.getSolde() = oldSolde + m
   * @throws IllegalArgumentException si m < 0
   * ou si m+oldSolde > this.plafondMax()
   * @throws OperationInterdite si !this.estOuvert()
   */
  public void depot(double m);

  /**
   * Fait passer un compte de l'état fermé à l'état ouvert.
   * @return false si le compte était déjà ouvert.
   */
  public boolean ouvrir();

  // Voici les contrats ajoutés

  /**
   * Retire le montant m du solde courant.
   * @param m montant du retrait
   * precondition : m >= 0
   * postcondition : this.getSolde() = oldSolde - m
   * @throws IllegalArgumentException si m < 0 ou oldSolde-m < 0
   * @throws OperationInterdite si !this.estOuvert()
   */
  public void retrait(double m);

  /**
   * Fait passer un compte de l'état ouvert à l'état fermé.
   * precondition: this.getSolde() ==0
   * @return false si le compte était déjà fermé
   * @throws OperationInterdite si this.getSolde() !=0
   */
  public boolean fermer();
}
```

Implantation de la classe CompteLivretA

Variables et constructeurs

Nous déclarons deux constructeurs. Un permet la création d'un compte dans l'état ouvert. Il prend en paramètre tous les arguments nécessaires à l'initialisation du compte, saut l'état d'ouverture qui sera mis à true. Un autre constructeur permet de créer un compte intialisé dans l'état fermé, avec un solde nul (autrement, violation de l'invariant). Il prend en paramètres le reste des valeurs nécessaires à l'initialisation. Ces deux constructeurs échouent si les paramètres passés

ne permettent pas d'initialiser le compte dans un état qui valide l'invariant. Voici le début de la classe, comprenant les variables d'instance et la définition des constructeurs. Notez que les constructeurs sont décrits chacun par un contrat javadoc.

```
public class CompteLivretA implements LivretA {

    private double solde;
    private double plafondMax;
    private boolean compteOuvert = false;
    private String nomTitulaire;

    /**
     * Création d'un compte avec état ouvert.
     * @param nom nom du titulaire
     * @param soldeinit solde initial
     * @param plafondm plafond maximal autorisé par la loi.
     * @throws IllegalArgumentException
     *    si {@code soldeinit < 0 || soldeinit > plafondm ||  plafondm <= 0}
     */
    public CompteLivretA(String nom, double soldeinit, double plafondm){
        if (soldeinit < 0 || plafondm <= 0 || soldeinit > plafondm )
            throw new IllegalArgumentException();
        solde = soldeinit;
        plafondMax = plafondm;
        compteOuvert = true;
        nomTitulaire=nom;
    }

    /**
     * Créeation d'un compte avec solde nul et état fermé.
     * @param nom nom du titulaire
     * @param plafondm plafond maximal autorisé par la loi.
     * @throws IllegalArgumentException si {@code plafondm <= 0}
     */
    public CompteLivretA(String nom, double plafondm){
        if ( plafondm <= 0)
            throw new IllegalArgumentException();
        plafondMax = plafondm;
        compteOuvert = false;
        solde = 0;
        nomTitulaire=nom;
    }
```

Accesseurs

```
public String getNomTitulaire(){
   return nomTitulaire;
}
public double getPlafond(){
    return this.plafondMax;
}
public double getSolde(){
    return solde;
}
public boolean estOuvert(){
```

```
    return this.compteOuvert;
}
```

`depot` et `retrait`

Ces opératons échouent si elles sont invoqués sur un compte fermé.

```
public void depot(double m){
    if (!estOuvert())
        throw new OperationInterdite("Depot␣interdit␣sur␣un␣compte␣fermé.");
    if (m<0 || m + this.getSolde() > this.getPlafond())
          throw new IllegalArgumentException();
    solde = solde + m;
}
public void retrait(double m){
    if (!estOuvert())
        throw new OperationInterdite("Retrait␣interdit␣sur␣un␣compte␣fermé.");
    if (m<0 || m > this.getSolde())
          throw new IllegalArgumentException();
    solde = solde - m;
}
```

`ouvrir` et `fermer`

Si une opération est "redondante" (fermer un compte déjà fermé, etc), alors l'opération renvoie false. Si l'opération a pu s'effectuer elle renvoie true. Si l'ontente de femer un compte dont le solde n'esyt pas nul, l'exception `OperationInterdite` est levée.

```
public boolean ouvrir(){
   if (estOuvert())
           return false;
   else {
       compteOuvert = true;
       return true;
   }
}

public boolean fermer(){
   if (!estOuvert())
       return false;
   else if (this.getSolde() != 0)
       throw new OperationInterdite("Le␣solde␣d'un␣compte␣à␣fermer␣doit␣être␣nul.");
   else {
       compteOuvert = false;
       return true;
   }
}
```

La méthode `invariantOK`: à utiliser pour les tests

Cette méthode permet de tester si l'état courant d'un objet valide l'invariant des comptres. Elle servira à tester **depuis la classe des tests, et uniquement là**, qu'un compte continue à valider l'invariant **après invocation** de la méthode que l'on est en train de tester. Elle doit être définie dans la classe `CompteLivretA` (et non pas dans la classe des tests), car elle doit pouvoir accéder *directement* aux variables de l'objet qui sont cachées (`private`. Cette accès doit

se faire directment, autrement dit, sans passer par d'autres méthodes comme les accesseurs, que l'on souhaite aussi par ailleurs tester. Notez que cette méthode est déclarée sans modificateur d'accès: elle est donc visible depuis la classe des tests, mais n'est pas rendue publique par la classe. Elle ne sera plus accessible une fois les tests terminés.

```
/**
  * Teste si l'état courant valide l'invariant décrit dans l'interface.
  * Doit être invoquée uniquement depuis la classe des tests.
  * @return true si l'état courant valide l'invariant.
  */
 boolean invariantOK(){
     return ((((!estOuvert() && getSolde()==0 )||
               (estOuvert() && getSolde() >= 0 && getSolde()<= getPlafond()))
             && getPlafond() > 0 );
 }
```

Test de `CompteLivretA`

Test des constructeurs

On teste systématiquesment qu'un constructeur qui n'échoue pas, produit un objet qui valide l'invariant. On introduit des tests d'échec pour caque cas qui invalide l'invariant:

```
/**
   * Tests constructeur compte en état ouvert.
   */
  @Test
  public void testConstructeurEtatOuvertOK() {
      System.out.println("constructeurEtatOuvertOK");
      CompteLivretA instance = new CompteLivretA("martin", 10, 20);
      assertTrue(instance.invariantOK());
  }

  @Test
  public void testConstructeurEtatOuvertSoldeAuPlafond() {
      System.out.println("constructeurEtatOuvertSoldeAuPlafond");
      CompteLivretA instance = new CompteLivretA("martin", 20, 20);
      assertTrue(instance.invariantOK());
  }

  @Test
  public void testConstructeurEtatOuvertSoldeZero() {
      System.out.println("constructeurEtatOuvertSoldeZero");
      CompteLivretA instance = new CompteLivretA("martin", 0, 20);
      assertTrue(instance.invariantOK());
  }

  @Test(expected=IllegalArgumentException.class)
  public void testConstructeurEtatOuvertEchecPlafondZero() {
      System.out.println("testConstructeurEtatOuvertEchecPlafondZero");
      CompteLivretA instance = new CompteLivretA("martin", 0, 0);
      assertTrue(instance.invariantOK());
  }

  @Test(expected=IllegalArgumentException.class)
```

```
public void testConstructeurEtatOuvertEchecPlafondNeg() {
    System.out.println("testConstructeurEtatOuvertEchecPlafondNeg");
    CompteLivretA instance = new CompteLivretA("martin", 0, -1);
}

@Test(expected=IllegalArgumentException.class)
public void testConstructeurEtatOuvertEchecSoldeNeg() {
    System.out.println("testConstructeurEtatOuvertEchecSoldeNeg");
    CompteLivretA instance = new CompteLivretA("martin", -1, 10);
}

@Test(expected=IllegalArgumentException.class)
public void testConstructeurEtatOuvertEchecSoldeDepassement() {
    System.out.println("testConstructeurEtatOuvertEchecSoldeDepassement");
    CompteLivretA instance = new CompteLivretA("martin", 20, 10);
}

 /**
 * Tests constructeur compte en état fermé et solde zéro.
 */

@Test
public void testConstructeurEtatFermeOK() {
    System.out.println("constructeurEtatFermeOK");
    CompteLivretA instance = new CompteLivretA("martin", 20);
    assertTrue(instance.invariantOK());
}

@Test(expected=IllegalArgumentException.class)
public void testConstructeurEtatFermeEchecPlafondZero() {
    System.out.println("testConstructeurEtatFermeEchecPlafondZero");
    CompteLivretA instance = new CompteLivretA("martin", 0);
}

@Test(expected=IllegalArgumentException.class)
public void testConstructeurEtatFermeEchecPlafondNeg() {
    System.out.println("testConstructeurEtatFermeEchecPlafondNeg");
    CompteLivretA instance = new CompteLivretA("martin", -1);
}
```

Test des accesseurs

Pour tous les cas nous invoquons le constructeur adapté à notre tests avec des valeurs initiales qui valident l'invariant, et nous invoquons ensuite la méthode à tester. D'après leurs contrats, aucun accesseur ne doit échouer. Nous testons donc uniquement des cas sans échec, à savoir nous testons que la postcondition est validée, et que l'invariant est maintenu.

```
/**
  * Test of getNomTitulaire method, of class CompteLivretA.
  */
@Test
public void testGetNomTitulaireOuvert() {
    System.out.println("getNomTitulaireOuvert");
    CompteLivretA instance = new CompteLivretA("martin", 10, 20);
    String expResult = "martin";
```

```
        String result = instance.getNomTitulaire();
        assertEquals(expResult, result);
        assertTrue(instance.invariantOK());
    }

    @Test
    public void testGetNomTitulaireOuvertFerme() {
        System.out.println("getNomTitulaireFerme");
        CompteLivretA instance = new CompteLivretA("martin", 20);
        String expResult = "martin";
        String result = instance.getNomTitulaire();
        assertEquals(expResult, result);
        assertTrue(instance.invariantOK());
    }

    /**
     * Test of getPlafond method, of class CompteLivretA.
     */
    @Test
    public void testGetPlafondOuvert() {
        System.out.println("getPlafondOuvert");
        CompteLivretA instance = new CompteLivretA("martin", 10, 20);
        double expResult = 20.0;
        double result = instance.getPlafond();
        assertEquals(expResult, result, 0.0);
        assertTrue(instance.invariantOK());
    }

     @Test
    public void testGetPlafondFerme() {
        System.out.println("getPlafondFerme");
        CompteLivretA instance = new CompteLivretA("martin", 20);
        double expResult = 20.0;
        double result = instance.getPlafond();
        assertEquals(expResult, result, 0.0);
        assertTrue(instance.invariantOK());
    }

    /**
     * Test of getSolde method, of class CompteLivretA.
     */
    @Test
    public void testGetSoldeOuvert() {
        System.out.println("getSoldeOuvert");
        CompteLivretA instance = new CompteLivretA("martin", 10, 20);
        double expResult = 10.0;
        double result = instance.getSolde();
        assertEquals(expResult, result, 0.0);
        assertTrue(instance.invariantOK());
    }

    @Test
    public void testGetSoldeFerme() {
        System.out.println("getSoldeFerme");
        CompteLivretA instance = new CompteLivretA("martin", 20);
        double expResult = 0.0;
```

```
        double result = instance.getSolde();
        assertEquals(expResult, result, 0.0);
        assertTrue(instance.invariantOK());
    }

    /**
     * Test of estOuvert method, of class CompteLivretA.
     */
    @Test
    public void testEstOuvertTrue() {
        System.out.println("estOuvertTrue");
        CompteLivretA instance = new CompteLivretA("martin", 10, 20);
        boolean expResult = true;
        boolean result = instance.estOuvert();
        assertEquals(expResult, result);
        assertTrue(instance.invariantOK());
    }

    @Test
    public void testEstOuvertFalse() {
        System.out.println("estOuvertFalse");
        CompteLivretA instance = new CompteLivretA("martin", 20);
        boolean expResult = false;
        boolean result = instance.estOuvert();
        assertEquals(expResult, result);
        assertTrue(instance.invariantOK());
    }
```

Tests pour `depot` et `retrait`

Ces opérations échouent si elles sont invoqués sur un compte fermé. Pour tous les cas (échec attendu ou pas), nous invoquons le constructeur adapté à notre tests avec des valeurs initiales qui valident l'invariant, et nous invoquons ensuite la méthode à tester. *Après cet appel*, pour les appels ne devant pas échouer, nous testons que la postcondition de la méthode testée est validée et que l'invariant est maintenu. Pour les appels devant pas échouer, nous indiquons l'exception attendue. Pas d'assertion, puisque l'appel doit échouer.

```
    /**
     * Test of depot method, of class CompteLivretA.
     */
    @Test
    public void testDepotZero() {
        System.out.println("depotZero");
        double m = 0.0;
        CompteLivretA instance = new CompteLivretA("martin", 100, 2000);
        double old = 100;
        instance.depot(m);
        assertEquals(instance.getSolde(), old+m, 0);
        assertTrue(instance.invariantOK());
    }

    @Test
    public void testDepotOK() {
        System.out.println("depotOK");
        double m = 50.0;
        CompteLivretA instance = new CompteLivretA("martin", 100, 2000);
```

```
        double old = 100;
        instance.depot(m);
        assertEquals(instance.getSolde(), old+m, 0);
        assertTrue(instance.invariantOK());
    }
    /**
    * On peut ajouter jusqu'au plafond compris.
    */
    @Test
    public void testDepotAuPlafond() {
        System.out.println("depotAuPlafond");
        double m = 900.0;
        CompteLivretA instance = new CompteLivretA("martin", 100, 1000);
        double old = 100;
        instance.depot(m);
        assertEquals(instance.getSolde(), old+m, 0);
        assertTrue(instance.invariantOK());
    }
    /**
     * On ne peut pas dépasser le plafond.
     */
    @Test(expected=IllegalArgumentException.class)
    public void testDepotEchecDepassement() {
        System.out.println("testDepotEchecDepassement");
        double m = 901.0;
        CompteLivretA instance = new CompteLivretA("martin", 100, 1000);
        double old = 100;
        instance.depot(m);
    }
    /**
    * Le montant de l'opération ne peut être négatif.
    */
    @Test(expected=IllegalArgumentException.class)
    public void testDepotEchecNeg() {
        System.out.println("depotEchecNeg");
        double m = -1.0;
        CompteLivretA instance = new CompteLivretA("martin", 100, 1000);
        instance.depot(m);
    }
   /**
   * On ne peut pas éffectuer un retrait sur un compte fermé.
   */
    @Test(expected=OperationInterdite.class)
    public void testDepotEchecFerme() {
        System.out.println("depotEchecFerme");
        double m = 0.0;
        CompteLivretA instance = new CompteLivretA("martin", 1000);
        instance.depot(m);
    }

    /**
     * Test of retrait method, of class CompteLivretA.
     */
    @Test
    public void testRetraitOuvertZero() {
        System.out.println("retraitOuvertZero");
```

```
        double m = 0.0;
        CompteLivretA instance = new CompteLivretA("martin", 100, 2000);
        double old = 100;
        instance.retrait(m);
        assertEquals(instance.getSolde(), old-m, 0);
        assertTrue(instance.invariantOK());
    }

    @Test
    public void testRetraitOuvertOK() {
        System.out.println("retraitOuvertOK");
        double m = 50.0;
        CompteLivretA instance = new CompteLivretA("martin", 100, 2000);
        double old = 100;
        instance.retrait(m);
        assertEquals(instance.getSolde(), old-m, 0);
        assertTrue(instance.invariantOK());
    }

    @Test
    public void testRetraitAuPlafond() {
        System.out.println("retraitAuPlafond");
        double m = 0.0;
        CompteLivretA instance = new CompteLivretA("martin", 100, 100);
        double old = 100;
        instance.retrait(m);
        assertEquals(instance.getSolde(), old-m, 0);
        assertTrue(instance.invariantOK());
    }
   /**
    * Le montant de l'operation ne peut être négatif.
    */
    @Test(expected=IllegalArgumentException.class)
    public void testRetraitEchecParamNeg() {
        System.out.println("retraitEchecParamNeg");
        double m = -1.0;
        CompteLivretA instance = new CompteLivretA("martin", 100, 1000);
        instance.retrait(m);
    }
   /**
    * Le compte  ne peut devenir débiteur.
    */
    @Test(expected=IllegalArgumentException.class)
    public void testRetraitEchecSoldeNeg() {
        System.out.println("retraitEchecSoldeNeg");
        double m = 150;
        CompteLivretA instance = new CompteLivretA("martin", 100, 1000);
        instance.retrait(m);
    }
    /**
     * Retrait interdit sur un compte fermé.
     */
    @Test(expected=OperationInterdite.class)
    public void testRetraitEchecFerme() {
        System.out.println("retraitEchecFerme");
        double m = 0.0;
```

```
        CompteLivretA instance = new CompteLivretA("martin", 1000);
        instance.retrait(m);
    }
```

Tests pour `ouvrir` et `fermer`

Nous procédons comme pour les méthodes depot et retrait.

```
    /**
     * Test of ouvrir method, of class CompteLivretA.
     */
    @Test
    public void testOuvrirTrue() {
        System.out.println("ouvrir");
        CompteLivretA instance = new CompteLivretA("martin", 1000);
        boolean expResult = true;
        boolean result = instance.ouvrir();
        assertEquals(expResult, result);
        assertTrue(instance.invariantOK());
    }
     /**
     * Compte déjà ouvert.
     */
    @Test
    public void testOuvrirFalse() {
        System.out.println("ouvrir");
        CompteLivretA instance = new CompteLivretA("martin", 100, 1000);
        boolean expResult = false;
        boolean result = instance.ouvrir();
        assertEquals(expResult, result);
        assertTrue(instance.invariantOK());
    }

    /**
     * Test of fermer method, of class CompteLivretA.
     */
    @Test
    public void testFermerTrue() {
        System.out.println("fermerTrue");
        CompteLivretA instance = new CompteLivretA("martin", 0, 1000);
        boolean expResult = true;
        boolean result = instance.fermer();
        assertEquals(expResult, result);
        assertTrue(instance.invariantOK());
    }
    /**
     * Compte deja ferme.
     */
    @Test
    public void testFermerFalse() {
        System.out.println("fermerFalse");
        CompteLivretA instance = new CompteLivretA("martin", 1000);
        boolean expResult = false;
        boolean result = instance.fermer();
        assertEquals(expResult, result);
```

```
}
/**
 * Le solde devrait etre nul.
*/
@Test(expected=OperationInterdite.class)
public void testFermerEchec() {
    System.out.println("fermerEchec");
    CompteLivretA instance = new CompteLivretA("martin", 10, 1000);
    boolean result = instance.fermer();
}
```