Génie Logiciel Avancé

Année 2022-2023

Prof. Burkhart Wolff
wolff@lri.fr

Adrien Durier
adrien.durier@universite-paris-saclay.fr
Jeremy Marrez
jeremy.marrez@universite-paris-saclay.fr

TD 3 - Test boîte noire

Semaine du 10 octobre 2022

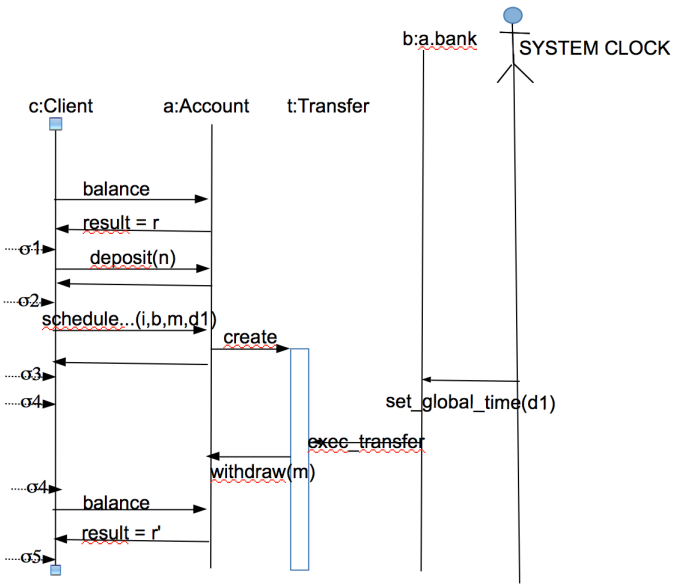
Exercice 1 (Test fonctionnel de séquence.)
On reprend le cahier des charges de la banque du TD1, les formalisations des invariants ainsi que les contrats des opérations.

Questions

1. Construire un diagramme de séquence d'un client effectuant un virement. Ce scenario doit utiliser les opérations `deposit`, `balance`, `schedule_transfer` et `exec_transfer`.
2. Construire un diagramme de séquence de deux clients accédant à leurs comptes chèque. Ce scenario doit utiliser les opérations `deposit`, `balance` et `withdraw`.
3. Similairement, construire un diagramme de séquence de deux clients accédant à leurs comptes épargne.

Solutions

Question 1 *Test-scenario abstrait*



σ_4 should observe : $r' = r + n - m \wedge n \leq 0 \wedge m \leq 0$ for all r n m.
Consider stronger conditions if $r \leq 0$: Under which conditions is this still a valid scenario ?

Question 2 *Analogue. Formuler les contraintes pour une serie de transactions balance of c1 , balance c2 , withdraw c2 , deposit c2 , withdraw(c1) balance of c1 and balance of c2, pour exemple. Les comptes sont des comptes de cheque.*

Si besoin, jouer une variante avec des comptes d'epargne pour les deux clients.

Exercice 2 (Test fonctionnel)

Une société vend deux produits A et B au prix unitaire de 5 € pour A et de 10 € pour B. Une commande comprend une certaine quantité du produit A et une certaine quantité du produit B. Le coût d'une commande est la somme totale des prix unitaires des produits commandés, à laquelle on applique une réduction selon les règles suivantes :

- Si la somme totale est supérieure ou égale à 200 €, on applique une réduction de 5%, si elle est supérieure ou égale à 1000 €, la réduction est de 20%. Ces deux réductions ne sont pas cumulables et portent sur la somme totale.
- La société souhaitant encourager la vente de A, on applique, sur le prix obtenu grâce à la règle précédente, une réduction supplémentaire de 10% si la commande comprend au moins 45 produits A.

Questions

1. Donnez un ensemble de tests pour le calcul du coût total d'une commande. Pour chaque test :
   - expliquez le cas particulier visé par le test (objectif de test) ;
   - donnez la formule du résultat attendu ;
   - donnez des valeurs concrètes et le résultat correspondant attendu.
2. Complétez votre jeu de tests par une analyse aux limites.

Solutions

	Objectif de test	*Formule du résultat attendu*	*Données d'entrée* A	B	*Résultat attendu*
1	*Prix sans réduction :* $5A + 10B < 200$ *et* $A < 45$	$5A + 10B$	*3*	*5*	*65*
2	*Réduction de 5% :* $200 \leq 5A + 10B < 1000$ *et* $A < 45$	$(5A + 10B) \times 0,95$	*20*	*20*	*285*
3	*Réduction de 20% :* $5A + 10B \geq 1000$ *et* $A < 45$	$(5A + 10B) \times 0,8$	*20*	*100*	*880*
4	*Réduction de 5% puis 10% :* $200 \leq 5A + 10B < 1000$ *et* $A \geq 45$	$(5A + 10B) \times 0,95 \times 0,9$	*60*	*10*	*342*
5	*Réduction de 20% puis 10% :* $5A + 10B \geq 1000$ *et* $A \geq 45$	$(5A + 10B) \times 0,8 \times 0,9$	*60*	*100*	*936*

On peut compléter ce jeu de tests par des tests aux limites en choisissant soigneusement des cas de part et d'autre d'une limite entre deux objectifs.

Par exemple entre l'objectif 1 et l'objectif 2, la limite se situe entre les valeurs 195 (maximum possible pour être dans le cas 1) et 200 (minimum possible pour être dans le cas 2), avec moins de 45A. On va construire deux tests ne variant que sur un seul paramètre d'entrée, l'un correspondant au cas 1 et l'autre au cas 2, de manière à vérifier que le calcul prend en compte

correctement la limite spécifiée. Si dans l'implantation, le cas 1 comprend la valeur 200, alors le 2e test échouera.

Petite remarque : ça n'a d'intérêt que si le résultat du 2e test est effectivement différent s'il est calculé selon le cas 1 ou selon le cas 2. Si les calculs mènent au même résultat, il n'y a en fait pas de limite à tester.

Dans le tableau, la valeur en gras est la valeur limite, déterminant le test. On essaie de ne pas choisir 0 comme valeur pour A ou B, même les valeurs 1 et 2 pour A ne sont pas forcément les meilleurs choix. Évidemment, les deux tests avec $A = 0$ (et $B > 0$) et $B = 0$ (et $A > 0$) sont également des tests aux limites.

Limite	*Objectif*	$5A + 10B$	A	B	*Résultat attendu*
Entre 1 et 2	$5A + 10B < 200$ *et* $A < 45$	***195***	*1*	*19*	*195*
	$200 \leq 5A + 10B < 1000$ *et* $A < 45$	***200***	*2*	*19*	*190*
Entre 2 et 3	$200 \leq 5A + 10B < 1000$ *et* $A < 45$	***995***	*1*	*99*	*945,25*
	$5A + 10B \geq 1000$ *et* $A < 45$	***1000***	*2*	*99*	*800*
Entre 4 et 5	$200 \leq 5A + 10B < 1000$ *et* $A \geq 45$	***995***	*99*	*50*	*850,725*
	$5A + 10B \geq 1000$ *et* $A \geq 45$	***1000***	*100*	*50*	*720*
Entre 2 et 4	$5A + 10B \geq 1000$ *et* $A \geq 45$	*300*	***44***	*8*	*285*
	$200 \leq 5A + 10B < 1000$ *et* $A \geq 45$	*305*	***45***	*8*	*260,775*
Entre 3 et 5	$5A + 10B \geq 1000$ *et* $A < 45$	*1220*	***44***	*100*	*976*
	$5A + 10B \geq 1000$ *et* $A \geq 45$	*1225*	***45***	*100*	*882*

Exercice 3 (Option)

On considère une classe `Tableau` permettant de stocker des entiers dans un tableau trié. Le nombre d'éléments du tableau est stocké dans un attribut `taille` et la longueur du tableau dans un attribut `capacite`. La taille est toujours supérieure ou égale à 0, la capacité doit toujours être strictement positive, et la taille inférieure ou égale à la capacité. En particulier, on ne peut pas appeler le constructeur de la classe `Tableau(int capacite)` avec un argument négatif ou nul.

```
public class Tableau {
    private int tab[];
    private int taille;
    private int capacite;

    public Tableau(int capacite) { }

    public boolean inserer(int val) { }

    public boolean supprimer(int val) {
        int i = 0;
        while(i < taille && tab[i] != val) {
            i++;
        }
        if(i == taille) {
            return false;
```

```
        }
        for(int j = i; j < taille-1; j++) {
            tab[j] = tab[j+1];
        }
        tab[taille-1] = 0;
        taille = taille-1;
        return true;
    }
}
```

On dispose de deux méthodes `inserer` et `supprimer`. Si le tableau n'est pas plein, la méthode `inserer` ajoute l'élément passé en argument au tableau en conservant l'ordre et renvoie *true*. Si le tableau est plein, elle renvoie *false*. La méthode `supprimer` enlève une occurrence de l'élément passé en argument et renvoie *true*. Elle renvoie *false* si l'élément n'était pas présent.

On cherche à tester la méthode `supprimer` de cette classe, de manière « interne », c'est-à-dire en ayant accès à tous les attributs.

1. Proposer un ensemble de tests pour la fonction `supprimer`.
2. Pour les modifications suivantes du programme, dire si la modification introduit une faute et lorsque c'est possible, donner un cas de test permettant de révéler la faute.
   (a) On modifie la condition du *if* par `i == taille-1`.
   (b) On modifie la condition d'arrêt de la boucle *for* par `j < taille`.
   (c) On modifie l'initialisation de la boucle *for* par `j=i+1`.
   (d) On change `taille` par `capacite` dans la condition du *while* et du *if*.
   (e) On supprime la ligne `tab[taille-1] = 0`.
   (f) On fait les modifications (d) et (e) ensemble.

On teste maintenant cette méthode depuis l'extérieur de la classe.

3. De quels observateurs aurait-on besoin idéalement pour pouvoir détecter autant de fautes qu'avec les tests internes ?
4. On ajoute les méthodes suivantes à la classe `Tableau` :

   ```
   int taille()
   int capacite()
   boolean present(int val)
   ```

   Quels aspects de l'implémentation ne peut-on plus observer ? Peut-on les tester ? Comment ?
5. Pour deux des tests proposés en question 1, construire la suite d'appels de méthodes (le scénario de test) permettant d'exécuter ce test, depuis l'initialisation des objets jusqu'à la vérification du résultat obtenu.

Solutions

1. *Tests pour supprimer.*

Objectif de test	*Données d'entrée*				*Résultat attendu*			
	tab	*taille*	*capacite*	*val*	*tab*	*taille*	*capacite*	*result*
tableau vide	*[]*	*0*	*10*	*2*	*[]*	*0*	*10*	*false*
tableau non vide et elt absent	*[1,4,6,8]*	*4*	*6*	*5*	*[1,4,6,8]*	*4*	*6*	*false*
tableau non vide et elt présent une fois	*[3,5,6]*	*3*	*10*	*5*	*[3,6]*	*2*	*10*	*true*
tableau non vide et elt présent plusieurs fois	*[1,4,4,6,7]*	*5*	*8*	*4*	*[1,4,6,7]*	*4*	*8*	*true*

2. (a) *La modification introduit une faute : si l'élément qu'on cherche est le dernier élément du tableau, la méthode renvoie false et ne modifie pas le tableau.*
   Données d'entrée : tab = [2,4,6,7], taille = 4, capacite = 5, val = 7
   Résultat attendu : tab = [2,4,6], taille = 3, capacite = 5, result = true
   Avec le code modifié, résultat observé : tab = [2,4,6,7], taille = 4, capacite = 5, result = false.

   Autre test révélant une erreur : si on supprime un élément absent, le dernier élément est retiré, la taille est décrémentée et la méthode renvoie true :
   Données d'entrée : tab = [2,4,6,7], taille = 4, capacite = 5, val = 1
   Résultat attendu : tab = [2,4,6,7], taille = 4, capacite = 5, result = false
   Avec le code modifié, résultat observé : tab = [2,4,6], taille = 3, capacite = 5, result = true.

   (b) *La modification introduit une faute car* `tab[j+1]` *fera référence à une case en dehors du tableau si taille = capacite. Il suffit de tester la suppression d'un élément dans un tableau plein pour révéler la faute.*
   Données d'entrée : tab = [1,3,4,5], taille = 4, capacite = 4, val = 3
   Résultat attendu : tab = [1,3,5], taille = 3, capacite = 4, result = true
   Avec le code modifié, résultat observé : Levée d'exception : indice hors des bornes du tableau.

   (c) *La modification introduit une faute puisque l'élément supprimé n'est pas écrasé, même si tous les suivants sont bien décalés. Il suffit de tester la suppression d'un élément présent, mais attention, cet élément ne doit pas être le dernier (sinon il est écrasé par l'instruction* `tab[taille-1] = 0`*) et il ne doit être présent qu'une seule fois (sinon c'est la deuxième occurrence qui est supprimée et ça revient au même).*
   Données d'entrée : tab = [2,3,5,8,9], taille = 5, capacite = 8, val = 3
   Résultat attendu : tab = [2,5,8,9], taille = 4, capacite = 8, result = true
   Avec le code modifié, résultat observé : tab = [2,3,8,9], taille = 4, capacite = 8, result = true

   (d) *La modification introduit une faute s'il est possible de trouver la valeur après taille. En Java les éléments non définis d'un tableau d'entiers sont à 0, donc ici, même en ayant construit le tableau de manière à ce que tous les éléments après taille ne soient pas définis, il est possible de trouver 0 après taille. Donc si on supprime 0 d'un tableau qui ne contient pas 0 (entre 0 et taille-1), on va quand même le trouver*

à l'indice taille et donc continuer l'exécution de la méthode, c'est-à-dire supprimer le dernier élément, décrémenter la taille et renvoyer true.
Données d'entrée : tab = [1,3,6], taille = 3, capacite = 8, val = 0
Résultat attendu : tab = [1,3,6], taille = 3, capacite = 8, result = false
Avec le code modifié, résultat observé : tab = [1,3], taille = 2, capacite = 8, result = true

(e) *La modification n'introduit pas vraiment une faute. En effet, si on enlève cette ligne, on oublie d'écraser le dernier élément du tableau après la suppression. Or la taille ayant été décrémentée, on ne va a priori pas essayer d'atteindre cette case. On observera l'erreur seulement si on affiche le contenu du tableau jusqu'à capacite par exemple ou si elle entre en jeu avec une autre erreur (d'où la question suivante).*

(f) *La combinaison des deux erreurs introduit une faute plus visible que la modification (d) seule. En effet, le fait de ne pas écraser le dernier élément fait qu'après l'indice taille, il n'y aura pas que des 0 mais aussi d'autres éléments qui n'auront pas été effacés. Il va donc être possible de trouver des éléments depuis longtemps effacés, et donc de supprimer le dernier élément du tableau lorsqu'on cherche à supprimer un élément censé être absent. Par exemple, supprimer deux fois de suite le dernier élément du tableau (alors qu'il n'était présent qu'une seule fois) révèle la faute.*
Données d'entrée : tab = [4,6,8], taille = 3, capacite = 6, val = 8
supprimer(8)
Résultat attendu : tab = [4,6], taille = 2, capacite = 6, result = true
supprimer(8)
Résultat attendu : tab = [4,6], taille = 2, capacite = 6, result = false
Avec le code modifié, résultat observé : tab = [4], taille = 1, capacite = 6, result = true.

3. *Tests depuis l'extérieur.*

   On aurait besoin d'afficher le tableau de 0 à taille, comme on pouvait le faire de manière interne. Plus de connaître les valeurs de taille et de capacite.

4. *Avec les méthodes taille(), capacite() et present(val), on ne peut plus observer l'ordre des éléments dans le tableau, ni la multiplicité des éléments. On ne peut pas tester l'ordre, c'est-à-dire qu'on a aucun moyen de savoir si les éléments sont effectivement stockés dans l'ordre dans la structure. Par contre on peut tester la multiplicité, c'est-à-dire qu'on peut observer que lorsqu'on insère un élément plusieurs fois, il est vraiment présent plusieurs fois. Pour ça, il suffit de le supprimer et de vérifier qu'il est toujours présent : inserer(2) ; inserer(2) ; supprimer(2) ; present(2) = true, puis supprimer(2) ; present(2) = false, montre bien que l'élément était bien stocké en deux exemplaires.*

5. *Scénarios de test.*

   Test pour supprimer dans le tableau vide.

```
// preambule
Tableau test1 = new Tableau(10);

// corps du test
boolean res = test1.supprimer(2);

// identification
```

```
assert(!res);
assert(test1.taille() == 0);
assert(test1.capacite() == 10);
```

Test pour supprimer un élément absent dans un tableau non vide.

```
// preambule
Tableau test2 = new Tableau(6);
test2.inserer(1);
test2.inserer(4);
test2.inserer(6);
test2.inserer(8);

// corps du test
boolean res = test2.supprimer(5);

// identification
assert(!res);
assert(test2.taille() == 4);
assert(test2.capacite() == 6);
assert(test2.present(1));
assert(test2.present(4));
assert(test2.present(6));
assert(test2.present(8));
```

Test pour supprimer un élément présent une fois.

```
// preambule
Tableau test3 = new Tableau(10);
test3.inserer(3);
test3.inserer(5);
test3.inserer(6);

// corps du test
boolean res = test3.supprimer(5);

// identification
assert(res);
assert(test3.taille() == 2);
assert(test3.capacite() == 10);
assert(!test3.present(5));
assert(test3.present(3));
assert(test3.present(6));
```

Test pour supprimer un élément présent plusieurs fois.

```
// preambule
Tableau test4 = new Tableau(8);
test4.inserer(1);
test4.inserer(4);
test4.inserer(4);
```

```
test4.inserer(6);
test4.inserer(7);

// corps du test
boolean res = test4.supprimer(4);

// identification
assert(res);
assert(test4.taille() == 4);
assert(test4.capacite() == 8);
assert(test4.present(1));
assert(test4.present(4));
assert(test4.present(6));
assert(test4.present(7));
```