TD n°7
Programmation orientée objets
MIAS2 *
Correction

Avril 2003

1 Gestion des scores

Nous allons compléter la classe `Joueur` du TD n°2, en lui ajoutant les assertions.

Exercice 1.1 Détermination de l'invariant de la classe `Joueur`.

Les caractéristiques essentielles de la classe `Joueur` sont :

- le nom du joueur
- son score
- le score maximal autorisé

Déterminez les propriétés communes à toutes les instances de la classe `Joueur` et déduisez en les assertions constituant l'invariant de classe.

Correction 1.1 *Première version de la classe `Joueur`.*

Pour toute instance de la classe `Joueur`, les trois caractéristiques essentielles doivent être définies (i.e. avoir une valeur utilisable). Les champs `score` et `scoreMaximal` étant définis à l'aide de types de base (`int`), ils ont une valeur utilisable (dans un premier temps nous ne cherchons pas à savoir si les valeurs de ces champs sont correctes). Par contre, le champ `nom` étant de type classe (`String`), il faut s'assurer qu'il est initialisé avec la référence à un objet d'où l'assertion : `nom != null`.

Nous devons ensuite nous assurer que les valeurs des différents champs sont des valeurs admissibles. Pour `nom`, toute valeur peut être considérée comme admissible. Pour `score` nous devons avoir $0 \leq \text{score} \leq \text{scoreMaximal}$, nous en déduisons l'invariant : `0 <= score && score <= scoreMaximal`.

La première version de la classe `Joueur` peut alors s'écrire :

```
/**
 * Representation d'un joueur pour un jeu avec score.
 *
 * ATTENTION : l'invariant spécifié ici est faux, compte-tenu de la
 * règle interdisant l'utilisation de champs private dans les
 * assertions, une version correcte sera donnée par la suite.
 */
public class Joueur {
    /*@
     * invariant nom != null;
     * invariant 0 <= score && score <= scoreMaximal;
     */

    private String nom;
    private int score;
    private int scoreMaximal;
```

*Marc Champesme – Département d'Informatique – Institut Galilée

```
}
```

Exercice 1.2 Définition d'un constructeur.

En programmation par contrat, le constructeur se voit assigné la tâche de rendre vrai l'invariant de classe. Le premier constructeur défini au TD n°2 respecte cette condition :

```
public Joueur() {
    nom = "inconnu";
    score = 0;
    scoreMaximal = 100;
}
```

Par contre, il n'en est pas de même pour le second constructeur :

```
public Joueur(String nom) {
    this.nom = nom;
    score = 0;
    scoreMaximal = 100;
}
```

Déterminez dans quelles circonstances ce constructeur pourrait enfreindre l'invariant de classe. Comment faire pour s'assurer que ce constructeur respecte le contrat ?

Ecrivez maintenant un nouveau constructeur prenant en paramètres, en plus du nom, le score initial du joueur et le score maximal autorisé.

Correction 1.2 Définition d'un constructeur.

Si le paramètre `nom` du constructeur est une référence `null`, alors il y aura violation du contrat. Pour s'assurer que le contrat sera rempli il faut ajouter une précondition :

```
/**
 * Initialisation d'une instance de Joueur dont le nom sera la chaîne
 * fournie en paramètre.
 * Le score du joueur est initialisé à 0 et le score maximal à 100.
 * @param nom le nom du nouveau joueur
 */
//@ requires nom != null;
public Joueur(String nom) {
    this.nom = nom;
    score = 0;
    scoreMaximal = 100;
}
```

Le troisième constructeur s'écrit :

```
/**
 * Initialisation d'une instance de Joueur dont le nom sera la chaîne
 * fournie en paramètre.
 * Le score du joueur et le score maximal sont initialisés à partir des
 * valeurs fournies en paramètre.
 *
 * @param nom le nom du nouveau joueur
 * @param scoreInitial le score initial du joueur
 * @param scoreMax le score maximal du joueur
 */
/**@
 * @ requires nom != null;
 * @ requires scoreInitial >= 0 && scoreInitial <= scoreMaximal;
 */
public Joueur(String nom, int scoreInitial, int scoreMaximal) {
```

```

    this.nom = nom;
    score = scoreInitial;
    this.scoreMaximal = scoreMaximal;
}

```

Exercice 1.3 Visibilité et méthodes d'accès aux champs.

Les possibilités d'accès et surtout de modification des caractéristiques d'un objet, font aussi partie du contrat de la classe. Pour la classe `Joueur`, les caractéristiques `nom` et `scoreMaximal` ne doivent pas être modifiées après la création de l'instance. Les modifications de la caractéristique `score` doivent être possibles, mais doivent être limitées afin de s'assurer que le contrat ne sera pas enfreint. En conséquence, quelle visibilité doit on associer aux champs et quelles méthodes d'accès / modification doit on définir ?

Correction 1.3 *Visibilité et méthodes d'accès aux champs.*

Aucun des champs ne pouvant être modifié sans restriction, on doit utiliser la visibilité `private` pour les 3 champs. Les trois champs se verront associer une méthode d'accès (à priori aucun raison d'interdire l'accès à un de ces trois champs), mais seul le champ `score` se verra associé une méthode permettant de le modifier.

Exercice 1.4 Un problème de cohérence.

L'invariant de classe que nous avons défini au premier exercice fait référence à des champs `private`, hors ces champs étant `private` l'utilisateur est sensé ignorer leur existence. Trouvez un moyen de résoudre cette contradiction.

Correction 1.4 *Un problème de cohérence.*

Pour lever la contradiction, il suffit d'utiliser les méthodes d'accès à la place des champs :

```

/*@
 @ invariant   getNom() != null;
 @ invariant   0 <= getScore() && getScore() <= getScoreMaximal();
 @*/

```

Exercice 1.5 Définir l'effet des méthodes avec des postconditions.

Utilisez des postconditions pour spécifier l'effet de toutes les méthodes et constructeurs précédemment définis.

Correction 1.5 *Définir l'effet des méthodes avec des postconditions.*

```

/**
 * Représentation d'un joueur pour un jeu avec score. Le score doit
 * toujours être positif et inférieur ou égal à un score maximal fixé
 * à la création de l'instance.
 *
 * @author      Marc Champesme
 * @created     21 avril 2003
 */
public class Joueur {
    /*@
     @ invariant   getNom() != null;
     @ invariant   0 <= getScore() && getScore() <= getScoreMaximal();
     @*/
    private String nom;
    private int score;
    private int scoreMaximal;

    /**
     * Initialisation par défaut d'une instance de Joueur. Le nom du
     * joueur est initialisé à "inconnu", le score est initialisé à 0 et
     * le score maximal à 100.
     */
}

```

```

/*@
  @ ensures getNom().equals("inconnu");
  @ ensures getScore() == 0;
  @ ensures getScoreMaximal() == 100;
  @*/
public Joueur() {
    nom = "inconnu";
    score = 0;
    scoreMaximal = 100;
}

/**
 * Initialisation d'une instance de Joueur dont le nom sera la
 * chaîne fournie en paramètre. Le score du joueur est initialisé a
 * 0 et le score maximal a 100.
 *
 * @param nouveauNom le nom du nouveau joueur
 */
/*@
  @ requires    nouveauNom != null;
  @
  @ ensures getNom().equals(nouveauNom);
  @ ensures getScore() == 0;
  @ ensures getScoreMaximal() == 100;
  @*/
public Joueur(String nouveauNom) {
    this.nom = nouveauNom;
    score = 0;
    scoreMaximal = 100;
}

/**
 * Initialisation d'une instance de Joueur dont le nom sera la
 * chaîne fournie en paramètre. Le score du joueur est initialisé a
 * la valeur du paramètre scoreInitial et le score maximal a la
 * valeur du paramètre scoreMax.
 *
 * @param nouveauNom le nom du nouveau joueur
 * @param scoreInitial le score initial du joueur
 * @param scoreMax le score maximal du joueur
 */
/*@
  @ requires    nouveauNom != null;
  @ requires    scoreInitial >= 0 && scoreInitial <= scoreMax;
  @
  @ ensures getNom().equals(nouveauNom);
  @ ensures getScore() == scoreInitial;
  @ ensures getScoreMaximal() == scoreMax;
  @*/
public Joueur(String nouveauNom, int scoreInitial, int scoreMax) {
    this.nom = nouveauNom;
    score = 0;
    scoreMaximal = 100;
}

```

```

/**
 * Renvoie le nom du joueur. (La seule postcondition pertinente est
 * exprimée dans l'invariant et aucune précondition n'est
 * pertinente, la méthode étant sans effet de bord, on la déclare
 * "pure" pour pouvoir l'utiliser dans les assertions).
 *
 * @return    Le nom du joueur
 */
//@ pure
public String getNom() {
    return nom;
}

/**
 * Renvoie la valeur courante du score du joueur. (La seule
 * postcondition pertinente est exprimée dans l'invariant et aucune
 * précondition n'est pertinente, la méthode étant sans effet de
 * bord, on la déclare "pure" pour pouvoir l'utiliser dans les
 * assertions).
 *
 * @return    La valeur courante du score du joueur
 */
//@ pure
public int getScore() {
    return score;
}

/**
 * Renvoie la valeur du score maximal défini pour ce joueur. (La
 * seule postcondition pertinente est exprimée dans l'invariant et
 * aucune précondition n'est pertinente, la méthode étant sans effet
 * de bord, on la déclare "pure" pour pouvoir l'utiliser dans les
 * assertions).
 *
 * @return    La valeur du score maximal
 */
//@ pure
public int getScoreMaximal() {
    return scoreMaximal;
}

/**
 * Remplace le score actuel par la valeur donnée en argument.
 *
 * @param nouveauScore  nouvelle valeur du score du joueur
 */
/**@
 * @requires    nouveauScore >= 0 && nouveauScore <= getScoreMaximal();
 * @ensures    getScore() == nouveauScore;
 */
public void setScore(int nouveauScore) {
    score = nouveauScore;
}

```

```
/**
 * Ajoute la valeur positive donnée en argument au score courant.
 *
 * @param points nombre de points a ajouter au score
 */
/*@
 @ requires    points >= 0;
 @ requires    (getScore() + points) <= getScoreMaximal();
 @ ensures getScore() == \old(getScore()) + points;
 @*/
public void ajouterAuScore(int points) {
    setScore(score + points);
}
}
```