TP Preuves de programmes

Ing2, ESIPE

A faire en OpenJML (http://cse-212294.cse.chalmers.se/courses/sefm/openjml/ en interface online ou http://jmlspecs.sourceforge.net/)

Ou avec KeY (https://www.key-project.org/download/)

JML (http://www.eecs.ucf.edu/~leavens/JML//index.shtml)

Exercise 1

Considérons un tableau « `b` » de « `n` » entiers (integer). Soient « `j` » et « `k` », deux indices tels que `0<=j<k<n`. Dans les questions suivantes nous notons `b[j..k]` pour désigner le segment du tableau `b` (nous ne pouvais pas utiliser un tel sucre syntaxique dans JML).

Ecrivez les expressions JML qui décrivre précisement :

a) Tout les élements de `b[j..k]` sont des zéros

b) Tout les zéros de `b[0..n-1]` sont dans `b[j..k]`

c) Ce n'est pas le cas que tout les zéros de `b[0..n-1]` sont dans `b[j..k].` Ne pas utliser de négation (ce serait trop simple)

d) `b[0..n-1]` contient deux zeros (ni plus, ni moins)

e) `b[0..n-1]` contient au moins deux zeros

f) `b[0..n-1]` contient au plus deux zeros. Attention, un tableau qui a au plus 2 elements est par exemple un tableau ne comportant qu'un unique elements. Vérifier votre solution pour un tel tableau.

Exercise 2

Spécifier public `static void reverse(int[] b)` qui renverse l'ordre des elements de b.

Exercise 3

Considerons la classe suivante :

```
/* A simple linked list of entries. */
public class EntryList {
  Object first;
  EntryList rest;
```

```
  EntryList( Object first, EntryList rest ) {
    this.first = first;
    this.rest = rest;
  }

  // some methods, among them:
  int size() {
    // ...
  }
}
```

Spécifier `size()` en JML (attention, c'est plus compliqué qu'il n'y paraît).

Exercise 4

Nous cherchons a implementer une spécification des files (`Queue`). Nous avons actuellement ce prototype de classe :

```
public class Queue {
  Object[] arr;
  int size;
  int first;
  int next;

  Queue( int max ) {
    // ...
  }

  public int size() {
    // ...
  }

  public void enqueue(Object x) {
    // ...
  }

  public Object dequeue() {
    // ...
  }
}
```

Donnez la spéfication JML complète qui consiste en des invariants de classes et des pre- et post-conditions pour chaque méthode.

Exercise 5

Nous cherchons maintenant a spécifier une classe implémentant un jeu de labyrinthe. Celle-ci est décrit par un tableau a 2 dimensions comportant autant de colonnes par ligne (une matrice). La classe `Maze` déclarer :

- Des constantes `MOVE_X` (X=up, down, left, right) encodant les différent mouvement.
- Des constantes `EXIT`, `FREE` et `WALL` encodant les différents types de cellules, i.e, une sortie, libre ou occupé par un mur.

- Deux entiers `playerRow` et `playerCol` qui détermine la position actuel du joueur (dans la matrice)
- Une matrice d'entiers représentant le labyrinthe lui-même.

Le prototype de la classe est le suivant :

```
public class Maze {
  // CONSTANTS -- MOVE
  public final static int MOVE_UP    = 0;
  public final static int MOVE_DOWN  = 1;
  public final static int MOVE_LEFT  = 2;
  public final static int MOVE_RIGHT = 3;

  // CONSTANTS -- FIELDS
  public final static int FREE = 0;
  public final static int WALL = 1;
  public final static int EXIT = 2;

  /**
   * The playfield is given as a rectangle where
   *
   *   - Walls are represented by entries of value Maze.WALL ('1')
   *   - The exit is represented as an entry of value Maze.EXIT ('2')
   *   - All other entries are MAZE.FREE ('0')
   *   - A playfield has exactly one exit.
   *   - The first number determines the column, the second determines
   *     the row. Row and column numbers start at 0.
   *   - Each row has the same number of columns.
   */
  private int[][] maze;

  /**
   * Player position:
   *
   *   - The position of a player must always denote a field inside the
   *     maze which is not a wall
   */
  private int playerRow, playerCol;

  public Maze(int[][] maze, int startRow, int startCol) {
    this.maze = maze;
    // set player on start position
    this.playerRow = startRow;
    this.playerCol = startCol;
  }

  /**
   * Returns true if the player has reached the exit field;
   * the method does not affect the state.
   */
  public boolean won() {
    // TO BE IMPLEMENTED
    throw new RuntimeException();
  }

  /**
   * A move to position (newRow,newCol) is possible iff the
   * field is inside the maze and free (i.e. not a wall);
   * the method does not affect the state.
   */
  public boolean isPossible(int newRow, int newCol) {
    // TO BE IMPLEMENTED
```

```
    throw new RuntimeException();
  }

  /**
   * Takes a direction and performs the move if possible, otherwise
   * the player stays at the current position; the direction must be
   * one of the defined move directions (see constants
   * MOVE_xyz). The return value indicates if the move was successful.
   */
  public boolean move(int direction) {
    // TO BE IMPLEMENTED
    throw new RuntimeException();
  }
}
```

Votre travail consiste à fournir :

a) Les invariants JML spécifiant que le labyrinthe n'est pas vide ainsi que tout les éléments nécessaires.

b) Chaque ligne a bien le même nombre de colonnes (le labyrinthe est bien une matrice)

c) La position du joueur dans le labyrinthe n'est pas occupé par un mur et qu'il existe bien une sortie

d) Les spécification `normal_behavior` pour toutes les spécifications des méthodes (actuellement écrites en langage naturelle)

Exercise 6

Qu'est-ce que la méthode ci-dessous calcul si on suppose un entier x non négatif ?

```
public int x;

public int method() {
  int y = x;
  int z = 0;

  while (y > 0) {
    z = z + x;
    y = y - 1;
  }

 return z;
}
```

Ecrire la spécification JML d'une telle méthode ainsi les invariants de classe nécessaires et ceux la boucles. Prouver la terminaison d'une telle méthode. Que ce passe t'il dans le cas d'un entier négatif ou d'un dépassement d'entier ?

Exercise 7

En supposant que x est non négtif et y positif, que fait la méthode ?

```
public int x;
public int y;
```

```
public int method() {
  int x1 = x, q = 0;

  while (x1 >= y) {
    x1 = x1 - y;
    q = q + 1;
  }

 return q;
}
```

De même écrire les pre-post conditions ainsi que que les invariants en JML afin de prouver la correction partielle. Et pour la correction total ? Justifier !

Exercise 8

Idem

```
public int a[];

public void method() {
  for (int i = 0; i < a.length; i++) {
    if (a[i] < 0)
      a[i] *= -1;
  }
}
```

Exercise 9

```
public class ArrayHelper {
    /*@ public normal_behavior
      @ requires newE > array[at];
      @ ensures array[at] == newE;
      @
      @ also
      @
      @ public normal_behavior
      @ requires newE <= array[at];
      @ ensures true;
      @ assignable \nothing;
      @*/
    public static void replaceIfGreater(int newE, int at, int[] array) {
      if (newE > array[at]) {
          array[at] = newE;
      }
    }


}
```

Vérifier que la post-condition est correcte. Comment comprennez vous l'affichage eds outils ? Fixer les bugs si nécessaires et recommencez.

Exercise 10

Vous aurez 3 classes. L'un est la classe `Personn`. Imaginez la classe appartenant a une base de données d'une autorité de délivrance de permis de conduire. Chaquper personne dans la base a réussi les tests. Donc chaque personne a déjà un permis mais certains sont obsolètes.Vous devez avoir au moins 18 ans pour passer les tests. L'autorisation de conduire est donnée quand le permis est activée.

a) Spécifier les comportement normaux et exceptionels de la méthode `activateLicense` de la classe `Person`

b) Vérifier que les contracts sont bien respectés dans les post-conditions

c) Vérifier que la méthode `activateLicense` préserves bien les invariants de classe de `Person`.

Si une preuve ne peut être prouvée, essayez de comprendre pourquoi. Quelle situation n'est pas traitée ?

Exercise 11

Considerons la classe `Hashtable` que nous allons fournir. Elle représente un espage d'adressage avec un accès linéaire aux données dans le cas d'une collision. Dans cette table, les objets sont enregistrés dans un tableau « h ». D'ailleurs, afin de savoir facilement quand capacité maximal de « h » est atteinte (le tableau est plein), un champs `size` conserve le nombre d'objets enregistrés et enfin un champ `capacity` représente le nombre total d'objets pouvant encore être enregistrés dans la table.

La méthode `add`, qui est utilisé pour enregistrer un objet dans la table, tente d'abord de mettre l'objet à la position correspondant à son hashage. Si la position est déjà occupée, on recherche dans les cellules consécutives (modulo la taille du tableau) une cellule libre. Une position est libre si et seulement si elle contient un objet `null`.

Complétez en JLM la classe `Hashtable` en commencant par :

- Les champs des tailles ne sont jamais négatifs et toujours plus petits ou égals à la `capacity`
- La `capacity` doit etre la même valeur que `h.length`
- Le tableau h ne peut être `null`
- Il doit y avoir assez de place pour au moins une elements dans la table
- Le nombre d'éléments enregistrés dans le tableau h (i.e., le nombre de cellules qui contiennent autre chose que `null`) est `size`
- Si `size` est strictement plus petit que `capacity`, alors toutes ces propriétés doivent être vrais : add termine normalement, add incrémente que de 1 et après un add(obj,key), l'object obj est enregistré dans h à un index i.

Ecrire les clauses d'assignation et les invariants de classes et e boucles, les variants et les modificateurs JML quand nécessaires.

Exercise 11

Dans cette exercice, nous allons analyser la spécification et l'implantation de la classe `Coordinate` qui vous sera fournis et qui réprésente les coordinonnées cartesiennes d'un système. Vous trouverez dans ce fichier, la présence de situations d'erreurs c'est-à-dire des méthodes qui ne respectent pas complètement la spécification (i.e. `moveRight`), ou des spécifications erronées, etc.

Testez. Maintenant que vous avez bien relu le code, essayez de corriger au mieux les spéfications JML. Pensez que nous pouvons prouvés entièrement le code ? Justifiez !

Exercise 12

Dans cette classe, se trouve une méthode static permettant de trouver l'**indice** du plus petit element d'un tableau supposé initialisé ; l'indice est -1 si le tableau est vide ;

Rajouter les annotations nécessaire pour prouver ce programme.

```
public class Smallest {
   static public int Smallest(int[] a) {
          if (a.length == 0) return -1;
          int index = 0;
          int smallest = 0;
          while (a.length - index > 0) {
             if (a[index] < a[smallest]) {
                smallest = index;
            }
            index = index + 1;
        }
       return smallest;
  }
}
```