Corrigé de l'examen de session 1 2012 d'IF1

Corrigé non-officiel par Baptiste Fontaine pour IP7.

Exercice 1

Question 1

L'évaluation de `speg(123, 456)` provoque une cascade d'appels de fonction décrite ci-dessous :

1. `speg(123, 456)` : `123` est différent de `0`, donc on retourne un appel à `speg(12, 4563)`, car `123/10` vaut `12` (division entière) et `10*456+123%10` vaut `4560+3`, soit `4563`.
2. `speg(12, 4563)` : `12` est différent de `0`, donc on retourne un appel à `speg(1, 45632)`, car `12/10` vaut `1` et `10*4563+12%10` vaut `45630+2`, soit `45632`.
3. `speg(1, 45632)` : `1` est différent de `0`, donc on retourne un appel à `speg(0, 456321)`.
4. `speg(0, 456321)` : `0==0` est vrai, donc on retourne `456321`.

Question 2

On remarque que les appels récursifs à `speg` dans la question précédente permettaient de construire un nombre qui représente la concaténation du second argument avec le premier inversé. Il suffit donc d'appeler la fonction `speg` avec `0` en deuxième argument :

```
static int miroir(int n) {
    return speg(n, 0);
}
```

Question 3

Il suffit de calculer le miroir du nombre donné, et de vérifier s'il est égal à lui-même :

```
static boolean estMiroir(int n) {
    return n == miroir(n);
}
```

Question 4

```
static int persistance(int n) {

    // calcule la somme de n et de son miroir
    int somme = n + miroir(n);

    // teste si la somme est miroir d'elle-même
    if (estMiroir(somme)) {
        // on retourne son miroir, c'est a dire elle-même
        return somme;
    }

    // sinon on recommence
    return persistance(somme);
}
```

L'évaluation de `persistance(78)` provoque une cascade d'appels de fonctions décrite ci-dessous :

1. `persistance(78)` : Le miroir de `78` est `87`, et la somme est `78+87`, c'est-à-dire `165`. `165` n'est pas miroir d'elle-même, donc on retourne `persistance(165)`.
2. `persistance(165)` : Le miroir de `165` est `561`, et la somme est `726`, qui n'est pas miroir d'elle-même, donc on retourne `persistance(726)`.
3. `persistance(726)` : Le miroir de `726` est `627`, et la somme est `1353`, qui n'est pas miroir d'elle-même, donc on retourne `persistance(1353)`.
4. `persistance(1353)` : Le miroir de `1353` est `3531`, et la somme est `4884`, qui est miroir de lui-même, donc on le retourne.

Exercice 2

Question 1

```
static char alea() {
    // on cree un tableau avec chaque figure
    char[] figures = new char[] { 'C', 'F', 'L', 'P', 'S' };

    // on genere un indice entre 0 et 4
    int indice = (int)(Math.random()*5);

    // on utilise cet indice pour recuperer la figure correspondante
    return figures[indice];
}
```

Question 2

```
static int zuper(char f1, char f2) {
    // si les deux figures sont identiques
    if (f1 == f2) return 0;

    // sinon, on teste chaque cas possible
    switch (f1) {
        case 'C': if (f2 == 'F' || f2 == 'L') return 1 else return -1;
        case 'F': if (f2 == 'P' || f2 == 'S') return 1 else return -1;
        case 'L': if (f2 == 'F' || f2 == 'S') return 1 else return -1;
        case 'P': if (f2 == 'C' || f2 == 'L') return 1 else return -1;
        case 'S': if (f2 == 'C' || f2 == 'P') return 1 else return -1;
    }

    // ceci ne devrait pas arriver si les arguments donnes sont corrects
    return -2;

}
```

Question 3

```
static int tour() {
    char c1 = alea();

    Scanner sc = new Scanner(System.in);
    System.out.print("Entrez une figure : ");

    // on suppose que l'utilisateur entre un caractere correct
    char c2 = sc.nextLine().charAt(0);

    System.out.println("Ma figure est " + c1 + ".");

    return zuper(c1, c2);
}
```

Question 4

```
static void match() {

    int pointsMachine     = 0,
        pointsUtilisateur = 0;

    while(pointsMachine < 5 && pointsUtilisateur < 5) {
```

```
        int resultat = tour();

        if (resultat == 1) {
            pointsMachine++;
        } else if (resultat == -1) {
            pointsUtilisateur++;
        }

    }

    if (pointsMachine == 5) {
        System.out.println("L'ordinateur gagne, vous avez perdu. :(");
    } else {
        System.out.println("Vous avez gagne ! :)");
    }

}
```

Question 5

Pour truquer le jeu, on pourrait laisser l'utilisateur choisir sa figure, puis choisir une figure aléatoire parmi celles qui peuvent battre celle de l'utilisateur.

Exercice 3

Question 1

```
static boolean estDureeMaya(int[] duree) {

    // si on a pas 5 quantites
    if (duree.length != 5) return false;

    // si le nombre de kins est mauvais
    if (duree[0] < 0 || duree[0] > 19) return false;

    // si le nombre de winals est mauvais
    if (duree[1] < 0 || duree[1] > 17) return false;

    // si le nombre de tuns est mauvais
    if (duree[2] < 0 || duree[2] > 19) return false;

    // si le nombre de katuns est mauvais
    if (duree[3] < 0 || duree[3] > 19) return false;
```

```
    // si le nombre de baktuns est mauvais
    if (duree[2] < 0) return false;

    // sinon, c'est valide
    return true;
}
```

Question 2

```
static int maya2Jours(int[] duree) {

    // on prend la plus haute unite, le baktun
    int jours = duree[4];

    // on transforme en katuns
    jours = jours*20 + duree[3];

    // on transforme en tuns
    jours = jours*20 + duree[2];

    // on transforme en winals
    jours = jours*18 + duree[1];

    // on transforme en jours
    jours = jours*20 + duree[0];

    return jours;
}

static int jours2Maya(int jours) {

    int[] duree = new int[5];

    // kins (jours)
    duree[0] = jours%20;

    // winals (20 kins)
    jours /= 20;
    duree[1] = jours%18;

    // tuns (18 winals)
    jours /= 18;
    duree[2] = jours%20;
```

```
    // katuns (20 tuns)
    jours /= 20;
    duree[3] = jours%20;

    // baktuns (20 katuns)
    jours /= 20;
    duree[4] = jours;

    return duree;
}
```

```
        // si le numerateur est divisible par le denominateur,
        // le resultat est entier
        if (tmp%t[i+1] == 0) {
            n = tmp / t[i+1];

        // sinon, c'est termine
        } else {
            break;
        }

    }

    return n;

    }
```

L'énoncé n'est pas très clair concernant ce qu'on doit faire quand un produit n'est pas entier. Soit on s'arrête (c'est ce qu'on fait ici), soit on saute la fraction et on continue (on peut le faire en supprimant le `break`).

Exercice 5

Question 1

D'après l'énoncé, une inversion est un couple d'indice pour lesquels les entiers correspondants dans le tableau sont en ordre décroissant, c'est-à-dire que l'entier à gauche est plus grand que celui à droite. Le tableau d'inversions est un tableau de même longueur que le tableau original, avec pour chaque case d'indice `i` le nombre d'inversions en `(i, j)` avec `j` plus grand que `i`, c'est à dire, si le nombre en case `i` est `n`, le nombre de nombres à droite de `n` qui sont plus petits que `n`.

Le tableau `[3|1|2|0]` a donc 4 inversions, sont tableau d'inversions étant `[3|1|1|0]`, car il y a 3 nombres à gauche du `3` qui sont inférieurs à `3` (`1`, `2`, `3`), 1 nombre à droite du `1` qui est inférieur à `1` (`0`), et de même, 1 nombre à droite du `2` qui est inférieur à `2` (`0`). Il n'y a pas de nombre à droite du `0`.

Question 2

La permutation de longueur `n` qui a le plus grand nombre d'inversions est le tableau des entiers `n-1`, `n-2`, ..., `1`, `0`.

```
static int[] maxInversions(int n) {
    int[] permutation = new int[n];
```

```
    for (int i=0; i<n; i++) {

        permutation[n-i-1];

    }

    return permutation;
}
```

Question 3

```
static int nbInversions(int[] tab) {
    // si tab n'est pas une permutation
    if (!estPerm(tab)) return -1;

    int nb = 0;

    // pour chaque entier du tableau...
    for (int i=0; i<tab.length; i++) {

        // ... on regarde chaque entier a droite
        for (int j=i+1; j<tab.length; j++) {

            // si il y en a un plus petit
            // on incremente le nombre d'inversions
            if (tab[j] < tab[i]) nb++;
        }
    }

    return nb;
}
```

Question 4

```
static int[] tableauInversions(int[] tab) {

    int[] inv = new int[tab.length];

    // pour chaque entier du tableau
    for (int i=0; i<tab.length; i++) {

        // on compte le nombre d'inversions entre cet entier et les entiers
        // a sa droite
```

```
        int nb = 0;

        for (int j=i+1; j<tab.length; j++) {
            if (tab[j] < tab[i]) nb++;
        }

        // on stocke ce nombre dans le tableau d'inversions
        inv[i] = nb;

    }

    return inv;
}
```

Question 5

On sait que dans un tableau d'inversions `t`, pour un `i` donné, `t[i]` ne peut pas être supérieur aux nombre de cases à sa droite.

```
static boolean estInv(int[] tab) {
    for (int i=0; i<tab.length; i++) {
        if (t[i] > tab.length-i-1) return false;
    }

    return true;
}
```

Question 6

Pour récupérer la permutation correspondante à un tableau d'inversions `inv` de longueur `n`, il faut d'abord définir un ensemble `S` contenant les entiers de `0` à `n-1`, ainsi qu'un tableau vide `perm` de même longueur que le tableau d'inversions, que l'on va remplir au fur et à mesure. On parcourt ce tableau, et pour chaque `i`, `perm[i]` prend la valeur du plus petit nombre de `S` qui est plus grand que `inv[i]` nombres, et qui n'a pas encore été utilisé.

Exemple avec le tableau d'inversions `[3|2|0|0]` :

1. On a un ensemble `S` qui contient les entiers `0`, `1`, `2`, `3`
2. On crée un tableau vide de longueur `4` que l'on va remplir au fur et à mesure
3. On part avec `i=0`. `inv[0]` vaut `3`, donc `perm[0]` est le plus petit entier de `S` qui est supérieur à 3 nombres, soit `3`. On supprime `3` des entiers disponibles.

4. On en est à `i=1`. `inv[1]` vaut `2`, et le plus petit entier de `S` supérieur à 2 nombres est `2`. On place donc `2` dans `perm[1]` et on le supprime des entiers disponibles.

5. On en est à `i=2`. `inv[2]` vaut `0`, et le plus petit entier de `S` qui n'est supérieur à aucun nombre est `0`. On le place donc dans `perm[2]` et on le supprime des entiers disponibles.

6. On en est finalement à `i=3`. `inv[3]` vaut `0`, et comme `0` n'est plus disponible, le plus petit entier qui n'est supérieur à aucun nombre disponible est le seul restant, `1`. Donc `perm[3]` vaut `1`.

7. La permutation est donc `[3|2|0|1]`.

   static int[] inv2Perm(int[] inv) { // si tab n'est pas un tableau d'inversion
   if (!estInv(inv)) return null;

```
// on represente S avec un tableau d'entiers ; un entier qui n'est plus
// disponible est remplace par -1.
int[] s = new int[inv.length];

// remplissage
for (int i=0; i<s.length; i++) {
    s[i] = i;
}

// la permutation
int[] perm = new int[inv.length];

for (int i=0; i<inv.length; i++) {
    int plusGrandQue = inv[i];

    // on trouve l'entier qui correspond
    for (int j=0; j<s.length; j++) {
        // si un entier est -1, c'est qu'il a deja ete utilise,
        // on passe
        if (s[j] == -1) continue;

        // si 'plusGrandQue' vaut 0, c'est qu'on a trouve notre entier
        if (plusGrandQue == 0) {

            // on recupere l'entier...
            perm[i] = s[j];

            // ...et on le "supprime" du tableau
            s[j] = -1;
```

```
            break;
        }

        // on decremente 'plusGrandQue' et on passe a l'entier suivant
        plusGrandQue--;
    }
}

return perm;

}
```

Exercice 6

Dans cet exercice, on considèrera que la grille est stockée comme un tableau de lignes (`grille[y][x]` représente la position `(x, y)`), que la position du magasinier est donnée par un tableau représentant `x` et `y`, et qu'une direction est donnée par un tableau de deux entiers, le premier indiquant l'abscisse, l'autre l'ordonnée (ce qui donne `(-1, 0)` pour la gauche, `(1, 0)` pour la droite, `(0, -1)` pour le haut et `(0, 1)` pour le bas), l'origine étant en haut à gauche. Ce choix permet de changer la position du magasinier facilement à partir d'une direction, par exemple pour la direction `(a, b)` et la position `(x, y)`, la nouvelle position est obtenue par `(x+a, y+b)`.

Question 1

```
static boolean sontCorrectes(int[][] grille, int[] pos) {

    // entrepôt vide
    if (grille.length == 0) return false;

    // on verifie que le tableau est rectangulaire
    int longueur = grille[0].length;

    for (int i=1; i<grille.length; i++) {
        if (grille[i].length != longueur) return false;
    }

    // on verifie que le magasinier est dans l'entrepôt

    if (   pos[0] < 0 || pos[0] > longueur
        || pos[1] < 0 || pos[1] > grille.length) return false;

    // on verifie qu'il est sur une case libre ou une case cible
```

```
    int caseMagasinier = grille[pos[1]][pos[0]];

    // 2 : case cible contenant une caisse
    // 3 : case libre contenant une caisse
    // 4 : mur
    if (caseMagasinier > 0) return false;

    // on compte les caisses en trop. Si on trouve une case cible non
    // occupee, on decremente ce nombre, tandis que si on trouve une
    // caisse sur une case libre, on l'incremente.
    int caissesEnTrop = 0;

    // on parcourt le tableau
    for (int i=0; i<grille.length; i++) {
        for (int j=0; j<grille[i].length; j++) {

            // on verifie que chaque case contient une valeur
            // valide
            if (grille[i][j] < -1 || grille[i][j] == 1 || grille[i][j] > 4) {
                return false;
            }

            if (grille[i][j] == -1) { // case cible sans caisse
                caissesEnTrop--;
            } else if (grille[i][j] == 3) { // caisse sur une case libre
                caissesEnTrop++;
            }

        }
    }

    if (caissesEnTrop != 0) return false;

    return true;
}
```

Question 2

```
static String string(int[][] grille, int[] pos) {

    String s = "";

    // on parcours toutes les cases
    for (int i=0; i<grille.length; i++) {
        for (int j=0; j<grille[i]; j++) {
```

```
            // Si le magasinier est sur la case courante
            if (pos[0]==j && pos[1]==i) {

                // si elle est libre, on utilise 'A', sinon 'B'
                if (grille[i][j] == 0) { s += 'A'; continue; }
                s += 'B'; continue;
            }

            switch (grille[i][j]) {

                // case cible
                case -1: s += 'u'; continue;

                // case libre
                case  0: s += '.'; continue;

                // case cible avec caisse
                case  2: s += 'o'; continue;

                // case avec caisse
                case  3: s += 'n'; continue;

                // mur
                case  4: s += 'x'; continue;

                // au cas-ou
                default: s += '?'; continue;

            }

        }

        s += '\n';
    }

    return s;

}
```

Question 3

```
static boolean estComplete(int[][] grille) {

    // on parcours toutes les cases
```

```
    for (int i=0; i<grille.length; i++) {
        for (int j=0; j<grille[i]; j++) {
            // si on trouve une case cible sans caisse ou une case libre
            // avec caisse, la grille n'est pas complete
            if (grille[i][j] == -1 || grille[i][j] == 3) return false;
        }
    }

    return true;

}
```

Question 4

```
static int[] direction(char c) {

    // on cree le tableau de base
    int[] dir = new int[] { 0, 0 };

    // on met a jour la bonne case
         if (c == 'e') dir[0] =  1;
    else if (c == 'z') dir[1] = -1;
    else if (c == 'a') dir[0] = -1;
    else if (c == 's') dir[1] =  1;
    else return null;

    return dir;

}
```

Question 5

```
static void coupPossible(int[][] grille, int[] pos, char d) {
    int[] dir = direction(d);

    // si la direction donnee en argument est mauvaise, on ne fait rien
    if (dir == null) return;

    // on calcule la nouvelle position du magasinier (s'il peut jouer)
    int nx = pos[0] + dir[0]; // nouveau x
    int ny = pos[1] + dir[1]; // nouveau y

    // on verifie qu'on ne sort pas de la grille
    if (   nx < 0 || nx > grille[0].length
```

```
    || ny < 0 || ny > grille.length) {

    return;
}

// on regarde ce qu'il y a a cette position
// si il y a un mur, le coup n'est pas possible et on ne fait rien
if (grille[ny][nx] == 4) return;

// si il y a une case libre ou une case cible sans caisse, on peut
// deplacer le magasinier
if (grille[ny][nx] <= 0) { // -1 ou 0
    pos[0] = nx;
    pos[1] = ny;
    return;
}

// si il y a une caisse, il faut verifier derriere celle-ci qu'il y a
// une case libre. Pour avoir les coordonnees nx2 et ny2 de cette case,
// il suffit d'appliquer la direction du joueur aux coordonnees de la
// caisse.
int nx2 = nx + dir[0];
int ny2 = ny + dir[1];

// on verifie que la case derriere n'est pas en dehors de la grille
if (   nx2 < 0 || ny2 < 0
    || ny2 >= grille.length || nx2 >= grille[0].length) {

    return;
}

// s'il y a une caisse ou un mur derriere, on ne peut rien faire
if (grille[ny2][nx2] > 0) return;

// sinon, on pousse la caisse, donc on met a jour sa position
// on remarque qu'une case cible est representee par -1, et une case
// cible avec une caisse par 2 (donc -1+3), et qu'une case libre est
// representee par 0, tandis qu'une case libre avec une caisse est
// representee par 3, donc il suffit d'ajouter 3 a la representation
// d'une case sans caisse pour avoir son equivalent avec caisse.
grille[ny2][nx2] += 3;

// on met a jour l'ancienne position de la caisse
grille[ny][nx] -= 3;

// on met a jour la position du joueur
```

```
    pos[0] = nx;
    pos[1] = ny;

}
```

Question 6

```
static void jouer(int[][] grille, int[] pos) {

    Scanner sc = new Scanner(System.in);

    // tant que le jeu n'est pas complet
    while (!estComplete(grille)) {

        // on affiche la grille courante
        System.out.println(string(grille, pos));

        // on demande une direction a l'utilisateur
        System.out.print("dir? ");
        char dir = direction(sc.nextLine().charAt(0));

        // on tente de deplacer le magasinier. S'il ne peut pas,
        // ce n'est pas grave, on ne change rien et on refait un
        // tour de boucle
        coupPossible(grille, pos, dir);

    }

    System.out.print("Gagne !");

}
```

Question 7

Pour éviter au joueur de se retrouver bloqué, on pourrait :

- Lui permettre d'annuler son coup précédent
- Lui permettre de tirer les caisses

On pourrait aussi ajouter une méthode qui teste si le jeu est bloqué, et si oui affiche quelque chose à l'utilisateur, par exemple lui proposant de recommencer.