

# Examen – AO (Approche Objet)

Responsable : Pr. Xavier Blanc – Université de Bordeaux 1 (Xavier.Blanc@labri.fr)

Le 17 décembre 2011, 14h – 17h

4 pages - Tous documents autorisés

## Questions de cours (2 points) :

**Q1** En théorie des graphes, une composante fortement connexe d'un graphe orienté  $G$  est un sous-graphe de  $G$  possédant la propriété suivante, et qui est maximal pour cette propriété : pour tout couple  $(u, v)$  de sommets dans ce sous-graphe, il existe un chemin de  $u$  à  $v$ . Si on applique cela au graphe de classes, (1) donnez les conditions qui font que deux classes sont liées dans le graphe et (2) expliquez la relation entre composante fortement connexe et couplage.

**Q2** Quels sont les trois situations dans lesquelles il est conseillé d'utiliser l'héritage.

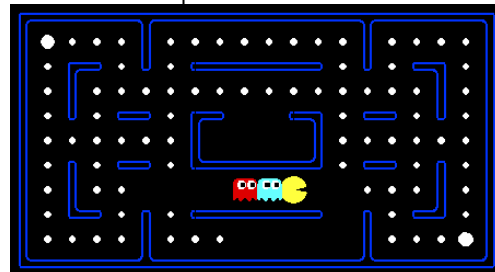
## Questions conception objet (6 points) :

Une équipe d'étudiants a développé PacMan en Java. PacMan est un jeu dont l'objectif est de faire manger à PacMan (le ballon Jaune) tous les points blanc sans se faire manger par les fantômes (voir copie d'écran). Lorsque PacMan mange un gros point blanc il se transforme et peut manger des fantômes pendant un temps limité seulement.

```
package poa.pacman;

public class PacManBigMac {
    int[][] map;
    int playerX,playerY;
    int ghost1X,ghost1Y;
    int ghost2X,ghost2Y;
    int ghost3X,ghost3Y;
    int score;

    public void play() {
        score = 0;
        initMap(); // Build the map
        initLocations(); //give location to player and ghots
        boolean dead = false;
        boolean isEnded = false;
        while (!dead && !isEnded) {
            int pp = obtainPlayerDirection();
            movePlayer(pp);
            score += computeScore();
            end = isEnded();
            moveGhosts();
            dead = checkPacManLife();
        }
    }
}
```



La figure précédente montre la première version de PacMan proposée par les étudiants. Cette version est réalisée en une seule classe. La figure ne fait apparaître que les champs de la classe et la méthode `play()`. Le plan de jeu est implémenté grâce à un tableau d'entier (`int[][]`). Un 0 veut dire que la case est vide, un 1 veut dire qu'elle contient un mur, un 2 un point blanc et un 3 un gros point blanc. La méthode `play()` est la méthode principale du jeu. Elle initialise les données nécessaires (`score`, plan de jeu et localisation de PacMan et des fantômes). Puis,

elle itère tant que PacMan n'est pas tué ou tant qu'il reste des points à manger. Dans chaque itération (1) on obtient la direction demandée par le joueur (haut, bas, gauche, droite), (2) on déplace PacMan, (3) on calcule le nouveau score, (4) on vérifie qu'il reste des points à manger, (5) on déplace les fantômes et enfin (6) on vérifie qu'un des fantômes n'a pas mangé PacMan.

**Q1** Pourquoi cette première version a une faible cohérence ?

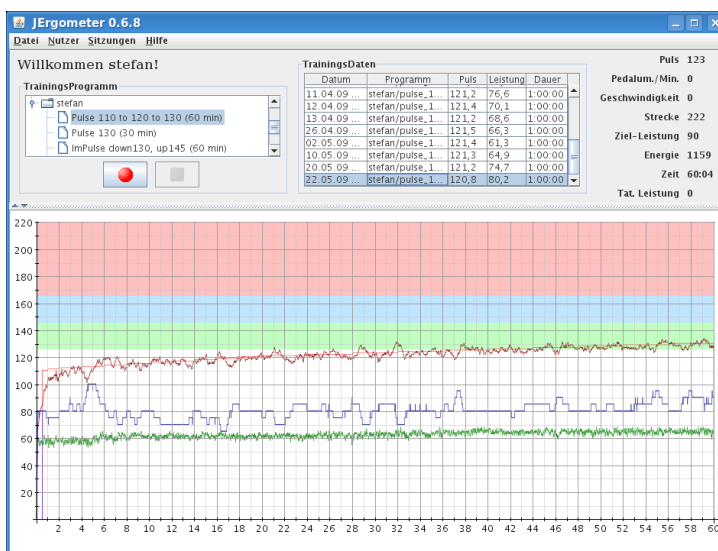
**Q2** Pour améliorer la cohérence de cette classe, proposez la classe PacManPlan qui représente un plan de Jeu (case vide, mur, point blanc et gros point blanc). Vous expliquerez toutes les modifications qu'il faut faire à la classe PacManBigMac.

**Q3** Toujours pour améliorer l'application, proposez les classes représentant PacMan et Ghost. L'objectif étant de faire en sorte que ces classes soient responsables des positions de PacMan et des fantômes. Vous expliquerez les modifications qu'il faut faire à la classe PacManBigMac.

**Q4** On veut maintenant améliorer les fantômes. Pour cela, on ajoute l'opération move() à la classe Ghost. Le code de cette méthode décrit la façon dont les fantômes se déplacent. On veut que le déplacement d'un fantôme dépende de son état. Si le fantôme ne voit pas PacMan (à vous de définir ce que voir veut dire) alors il déambule aléatoirement dans le plan de jeu. Si le fantôme voit PacMan et que celui-ci ne vient pas de manger un gros point blanc alors il le chasse. Si le fantôme voit PacMan et que celui-ci vient de manger un gros point blanc alors il fuit. Développez ce comportement en vous inspirant de patterns bien connus (n'oubliez pas de préciser comment le fantôme voit PacMan et comment il change d'état) !

## Questions Interface Graphique (4 points)

L'application suivante permet de suivre des performances sportives.



**Q1** Décrivez (à l'aide d'un arbre) la structure de cette interface graphique (objets graphiques + layout)

**Q2** Lorsqu'on clique sur le bouton record (le bouton avec le point rouge) l'application démarre l'enregistrement des performances et affiche les résultats. Donner le pseudo-code permettant ce traitement (introduisez toutes les classes que vous jugez nécessaires).

## Tests (4 points)

On souhaite développer une application qui vérifie qu'un document XML est bien constitué (il y a bien une balise fermante pour chaque balise ouvrante et il y a un arbre de composition des balises).

Cette application se résume à la méthode suivante : `boolean checkXML(String st)` qui retourne vrai si le document XML contenu dans la chaîne est bien constitué.

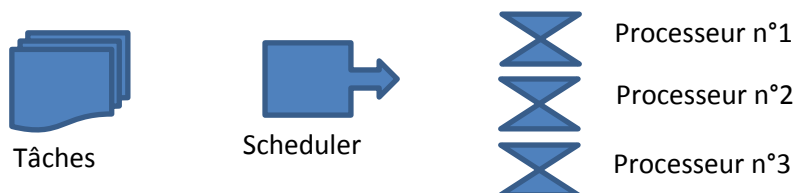
**Q1** Décrivez les tests fonctionnels nécessaires pour tester cette application. Vous préciserez les classes d'équivalences.

**Q2** En vous basant sur le code ci-dessous, proposez des tests structurels.

```
public static boolean isGoodXML(String doc) {
    StringTokenizer st = new StringTokenizer(doc);
    Stack<String> pile = new Stack<>();
    while (st.hasMoreTokens()) {
        String token = st.nextToken();
        if (isTag(token)) {
            if (isOpenTag(token)) {
                pile.push(token);
            }
            else {
                if (!openCloseTags(pile.peek(), token)) return false;
                else pile.pop();
            }
        }
    }
    return true;
}
```

## Simu Scheduler (4 points)

On souhaite simuler un scheduler sur une plateforme disposant de plusieurs processeurs. Le rôle d'un scheduler est d'affecter les tâches aux différents processeurs (voir figure). Le scheduler doit proposer des tâches aux processeurs. Ceux-ci ont la responsabilité d'exécuter des tâches.



**Q1** Décrivez les classes Tache, Scheduler et Processeur

**Q2** Afin de simuler le scheduler, on souhaite ajouter des taches de complexités différentes de façon aléatoire dans le temps. Proposez les classes nécessaires pour réaliser ce besoin.