# LO43 - Examen final du 27 Juin 2019

## Durée 2 heures – aucun document autorisé

**Exercice 1.** Rappeler les mécanismes de passage de paramètres en C++ et en Java respectivement. Expliquer brièvement quelle est la différence entre l’opérateur d’affectation « = » en C++ et l’opérateur « = » du langage Java. Par exemple, comment doit-on s’y prendre en Java lorsque l’on veut modifier la valeur d’un int via un paramètre de fonction.

**Exercice 2.** Considérons le schéma de conception Composite. Il a plusieurs intérêts. Il permet la composition récursive d’objets, de représenter une hiérarchie d'objets, d’ignorer la différence entre un composant simple et un composant en contenant d'autres (interface uniforme).

1) Rappeler un exemple type d’application où ce schéma est utilisé.

2) Donner le modèle de classe UML de ce pattern conception.

**Exercice 3.** Considérons l'organisation d’une compétition internationale de football ayant lieu dans un pays organisateur. Un joueur appartient à une équipe d’un pays. L’équipe s'entraîne dans un stade attitré. Les matchs entre deux équipes compétitrices ont lieu dans différents stades et celui de la finale dans la capitale du pays.

1) Donner un modèle de classes UML permettant le suivi de la compétition sportive. Notamment, on devra pouvoir connaître la composition des équipes, les matchs auxquels une équipe a participé avec la date, le stade, le lieu et le résultat, de même que son lieu d’entraînement et le pays.

2) On veut également pouvoir connaître pour chaque match et chaque joueur le rôle de joueur lors du match (titulaire, remplaçant, avant-centre, milieu offensif, etc…). Rajouter ces informations dans le diagramme de classes.

**Exercice 4.** On considère un téléphone portable avec ses fonctions standards de communication. La touche « verte » du téléphone permet d’accepter la réception d’un appel ou de lancer un appel vers un destinataire, sachant que le clavier alphanumérique comporte les dix chiffres de « 0 » à « 9 » pour la numérotation préalable. En plus de cela, un ensemble de menus sont fournis. On se déplace dans un menu à l’aide des boutons « en bas », « en haut » et « ok ». Une fois entré dans le menu « répertoire », l’utilisateur se sert des mêmes touches pour se déplacer, choisir un nom, un numéro de téléphone, puis valider pour lancer l’appel. Le but est de produire un diagramme Etat/Transition spécifiant les aspects dynamiques des différents modes d’utilisation du téléphone portable en fonction des séquences d’appui-touche. On ne détaillera que les modes précisés ci-dessus (veille, numérotation, appel, réception, saisie dans le répertoire). On attachera une importance particulière à proposer un diagramme bien lisible en procédant par étapes successives (approche descendante) d’imbrication des états.

1. Donner la liste (organisée) des événements d’entrée de l’automate, et préciser les états/modes de fonctionnement retenus. On peut s’aider d’un schéma de l’interface.
2. Donner un diagramme Etat/Transition hiérarchisé présentant le système dans sa globalité sans entrer ici dans le détail des sous-états principaux.
3. Préciser les sous-automates principaux en plusieurs étapes de raffinement.

**Problème.** Problème des philosophes dînant

Dans le problème des "philosophes dînant", il y a cinq philosophes autour d’une table et 5 fourchettes posées sur la table à gauche et à droite de chaque philosophe. Il s’agit d’un problème typique d’allocation de ressource. Chaque philosophe a accès à deux fourchettes situées à sa gauche et sa droite. Une contrainte est qu’un philosophe ne peut manger que s’il a saisi ses deux fourchettes adjacentes, gauche et droite (une dans chaque main). Lorsqu’il a mangé, il repose les 2 fourchettes puis entame une discussion. Lorsqu’il a fini de parler, il décide de manger à nouveau. Et le cycle se répète. On veut réaliser un programme de simulation Java qui simule les philosophes dînant par des threads synchronisés. Chaque philosophe doit donc tenter de s’emparer des 2 fourchettes, s’il réussit, il mange, repose les deux fourchettes, puis discute. Et le cycle manger-discuter reprend, sans fin. Notez que tous les philosophes doivent manger équitablement au bout d’un temps fini.

Afin de garantir le bon fonctionnement, plusieurs solutions sont envisagées pour le choix des objets partagés sur lesquels s’effectue la synchronisation.

1) Une première solution proposée peut consister à se synchroniser sur les fourchettes, chacune d’elles étant partagée entre 2 philosophes. Par exemple, chaque philosophe regarde si sa fourchette gauche est posée et donc libre, il la saisit si c’est le cas, sinon il attend (sur le moniteur de la fourchette), une fois la fourchette gauche dans sa main, il procède de la même façon avec la fourchette droite. En quoi cette solution n’est-elle pas correcte. Que risque-t-il de se passer ? Donner un exemple précis de fonctionnement non désiré.

2) Une deuxième solution proposée consiste à se synchroniser sur un seul et unique objet partagé (prenons la Table) qui gère l’accès aux fourchettes et mémorise l’état de chaque fourchette (posée, prise). La saisie des 2 fourchettes peut-elle être séquentielle (la gauche puis la droite) comme dans le cas précédent ? pourquoi cela pose-t-il problème ? On en déduit que la saisie sera simultanée, les 2 fourchettes sont saisies en même temps si elles sont libres en même temps. Le philosophe mange puis repose les deux fourchettes en même temps.

1. Donner un modèle de classe UML du programme Java et représenter les threads.
2. Modéliser la dynamique du programme via un diagramme Etat/Transition représentant l’activité des philosophes. Les événements considérés sont par exemple les « notifications » qui réveillent les philosophes en attente, et les « fins de repas ». Bien sûr, préciser les conditions et actions.
3. Donner le programme Java de simulation complet (classe Table, classe Philosophe, fonction main()), mettre en évidence la gestion de la synchronisation.