# LO43 - Examen final du 24 Juin 2015

## 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. Comment doit-on s’y prendre en Java lorsque l’on veut modifier la valeur d’un entier (int) via une méthode quelconque. Donner une méthode Inc qui permet d’incrémenter un entier (int) de 1 unité et un exemple d’utilisation, respectivement en C++ et puis en Java.

**Exercice 2.** Considérons que l'on veuille réaliser une classe Etudiant sous-classe de la classe abstraite Personne (ci-dessous) et que l'on dispose d'autre part d'une classe PersonneImp (ci-dessous) contenant une implantation de la classe Personne. L'héritage multiple permet en C++ d'hériter de plusieurs classes simultanément. Ainsi on peut écrire:

Class Etudiant : public Personne, private PersonneImp { … }

pour réaliser une classe étudiant sous-classe de Personne et réutilisant le code de PersonneImp.

1) Proposer un contenu pour la classe Etudiant avec son constructeur et l'écriture des méthodes abstraites.

class Personne {

public:

virtual ~Personne();

virtual const char\* nom() const = 0;

virtual const char\* dateNaissance() const = 0;

virtual const char\* adresse() const = 0;

virtual const char\* nationalite() const = 0;

};

class PersonneImp {

private:

TChaine Nom;

TDate DateNaissance;

TAdresse Adresse;

TPays Pays;

protected:

 PersonneImp( const TChaine& nom,

 const TDate& date,

 const TAdresse& adr,

 const TPays& pays );

virtual ~PersonneImp();

virtual const char\* nom() const;

virtual const char\* dateNaissance() const;

virtual const char\* adresse() const;

virtual const char\* nationalite() const;

};

2) Si on veut utiliser le "layering" c'est à dire réutiliser la classe PersonneImp en tant que champ private de Etudiant que faut-il changer dans cette classe PersonneImp. Donner la classe Etudiant en utilisant le layering.

**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.** Un télécopieur permettant d’envoyer et recevoir des fax comporte un clavier numérique, avec une touche de prise de ligne, un scanner pour lire les documents, un codeur/décodeur pour compresser/décompresser les documents envoyés/reçus, un module de gestion du protocole de communication automatisée avec le télécopieur distant, et une imprimante pour imprimer les documents reçus. D’un point de vue du logiciel, l’interface utilisateur, l’imprimante, le scanner, le codeur/décodeur, ainsi que la gestion ligne sont des threads synchronisés. Lors d’un envoi, la chaîne de transmission met en action les trois threads scanner, codeur et gestion ligne qui fonctionnent sur le mode producteur/consommateur. En réception, la chaine d’activation est inversée et met en jeu l’imprimante à la place du scanner. Nous nous intéressons davantage ici à la phase d’émission. Le scanner produit des pages brutes, une à une, et les transmet vers la file d’attente du codeur qui à son tour code les pages, une à une, et les transmet au fur et à mesure vers la file d’attente du thread gestion ligne, qui à son tour les émet sur la ligne de communication, une à une. A tout moment, l’utilisateur peut interrompre la communication par un appui sur la touche « prise de ligne ».

1) Donner un diagramme de classes du télécopieur tel que décrit.

2) Donner un diagramme Etat-Transition décrivant l’interaction homme/machine lors de la séquence d’envoi d’un document multi-pages. L’utilisateur positionne le document sur le scanner, saisit le numéro du distant, lance l’émission, éventuellement il peut l’interrompre ou non. Donner la liste des événements, la liste des états, puis un diagramme Etat-Transition. Utiliser l’imbrication d’état pour aider à la lisibilité. Ajouter le mode réception en tant qu’état supplémentaire, sans nécessairement détailler son contenu, seulement ses conditions d’activation.

3) On veut maintenant réaliser un programme C++/QT simulant la phase d’envoi d’un document multi-pages du point de vue de la synchronisation et implémentation des 3 threads Scanner/Codeur/GestionLigne. On s’intéresse principalement au contenu des boucles principales des 3 threads et des mécanismes de communication entre eux. On suppose qu’une communication entre deux threads s’effectue via un objet partagé entre les deux threads selon un schéma producteur/consommateur classique. Cet objet partagé que nous appelons Partage encapsule les mécanismes QT de synchronisation (QMutex, QWaitCondition), il encapsule également un buffer circulaire, et propose deux méthodes d’accès qui sont éventuellement bloquantes : ajouter(data) met en attente si le buffer est plein, retirer(data) met en attente si le buffer est vide. Les temps de copie des buffers vers les files d’attente sont supposés très courts. Les procédures globales scanner(data), coder(data), ou envoiLigne(data) sont supposées données.

1. Combien d’instances de la classe Partage sont-elles nécessaires. Quels sont les threads producteurs et consommateurs. Donner une description de la situation à l’aide d’un schéma, par exemple un schéma d’instance présentant les instances d’objet et leurs relations.
2. Donner les codes C++/QT des classes/threads Scanner/Codeur/GestionLigne et donner leur boucle d’exécution principale.
3. Donner le détail de la classe Partage, dont notamment ses méthodes ajouter() et retirer().