Composants - Examen Juin 2006

Durée : 2 heures

Documents autorisés : Notes de cours personnelles et énoncés/corrigés de TD

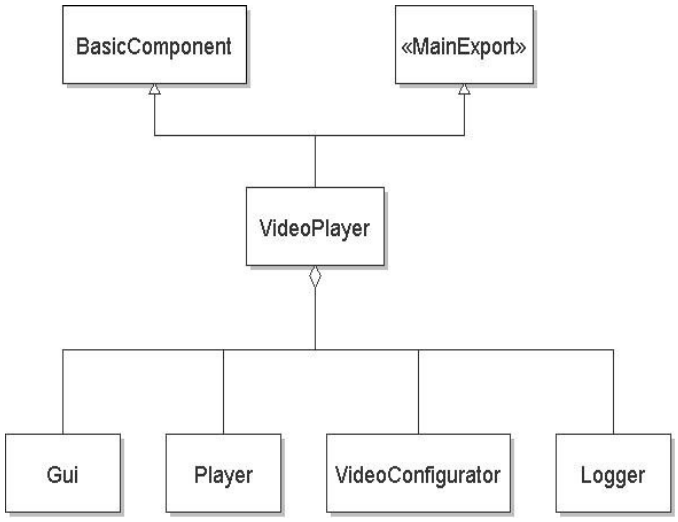
Remarques:

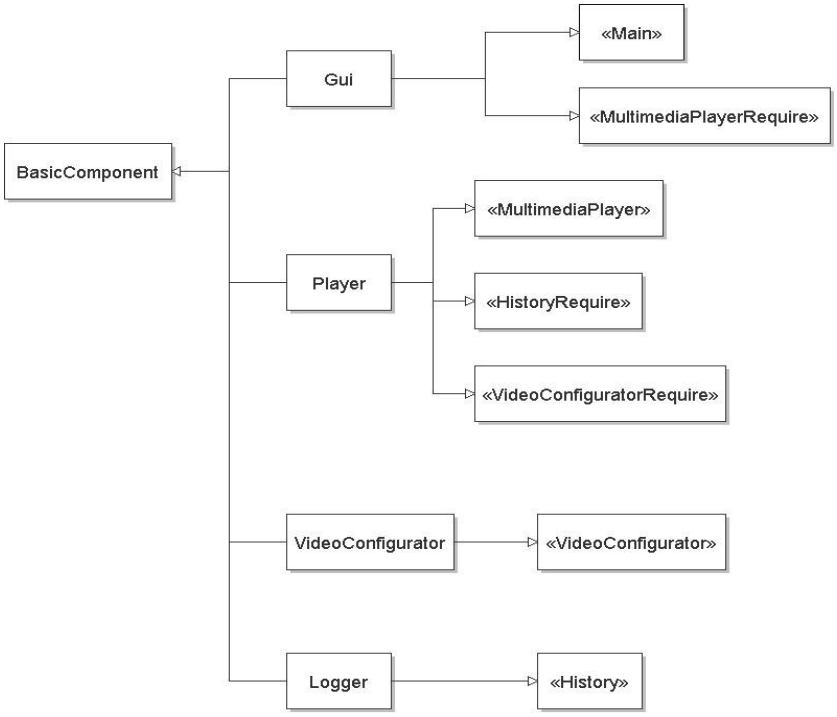
- attention à la gestion du temps, essayez dans la mesure du possible de ne pas dépasser le temps indiqué pour chaque exercice (en particulier l'exercice 1)
- la concision et la clarté des réponses seront appréciées (pas de discours sur 10 pages SVP)

Exercice 1 : Questions de cours (45mn)

Question 1 : Architecture de composants

Nous décrivons dans cette question une application qui lit des fichiers video. La classe `Gui` constitue l'interface graphique tandis que `Player` effectue le décodate proprement dit. La classe `VideoConfigurator` gère la qualité du rendu, elle permet par exemple de limiter le nombre d'images par seconde si le flux de données est insuffisant. Enfin le `Logger` conserve la trace des différentes actions effectuées.
Transformer les schémas UML de classe en un schéma de composition (composants, services requis/fournis) pour une instance de `VideoPlayer` reliée à un composant `MainComponent` dont l'unique service requis est `Main`. Effectuez les connexions qui vous semblent les plus pertinentes.





Question 2 : Architecture EJB: « Pizza - yoyo ! »

Il s'agit de mettre en place un système d'information et un site internet de vente en ligne de pizzas. Le site doit permettre à un client de commander une ou plusieurs pizza(s). Il saisit d'abord le nom des pizzas qu'il souhaite recevoir, puis communique son nom et son adresse au site qui valide alors la commande.
Du point de vue du vendeur de pizza, le site lui permet de maintenir à jour un catalogue de pizzas désignées par leur nom (par ex. : reine, margarita, paysanne). Ce catalogue contient aussi les quantités de chacun des ingrédients qu'elles utilisent (par ex. : pate, fromage, jambon). Enfin le site permet au vendeur de pizza de gérer son stock de matières premières car chaque fois qu'une pizza est commandée celui-ci est mis à jour suivant les quantités d'ingrédients qu'elle requiert.

1. décrire les tables de données utilisées dans cette application : nom de la table et nom/type de chaque colonne
2. décrire les beans nécessaires à la mise en oeuvre de l'application. Pour chaque bean on donnera :
   - son type : session, entity
   - son interface métier
3. décrire l'achat d'une pizza dans un scénario mettant en scène les différents beans qui composent l'application, ainsi que le programme client. On essaiera d'utiliser tous les verbes suivants : « créer », « référencer », « obtenir », « mettre à jour »; ainsi qu'au moins les noms communs « contexte », « client », « serveur », « nom jndi », « factory »

EXERCICE 2 : ADT ET CONTRATS (45MN)

Dans cet exercice, nous voulons modéliser sous la forme de spécifications algébriques (ADT) un ordonnanceur simple pour processus concurrents. Nous considérons deux définitions d'ADT : l'une pour les processus et l'autre pour l'ordonnanceur de type « *round robin* ». Un tel ordonnanceur est une liste circulaire de processus. Une importante contrainte est qu'un seul parmi l'ensemble des processus ne peut être actif à la fois. Lorsque l'on bascule au prochain processus, on choisit le suivant dans la liste et le processus précédemment actif est désactivé, il devient finalement le processus le moins prioritaire (dernier dans la liste circulaire).

Nous donnons la spécification de base pour les processus :

```
ADT Processus
Constructeur :
init : -> [Processus]
Observateur :
isActive : [Processus] -> bool
Opérations :
activate : [Processus] -> [Processus]
      précondition : activate(P) require not isActive(P)
passivate : [Processus] -> [Processus]
      précondition :  passivate(P) require  isActive(P)
Axiomes :
isActive(init()) = false
isActive(activate(P)) = true
isActive(passivate(P)) = false
```

Question 1 : ADT « Round Robin »

Pour l'ADT `RoundRobin`, vous devez donner une spécification complète intégrant, au minimum, le cahier des charges informel suivant. Les fonctionnalités de base de l'ordonnanceur sont:

- `getActiveProcess()` : retourne le processus actif
- `getActiveIndex()` : retourne l'index du processus actif
- `getProcess(index)` : retourne le processus d'index indiqué (>0)
- `addProcess(proc,priority)` : ajoute un processus à ordonnancer, avec trois niveaux de priorités: (1=prioritaire à ordonnancer le plus tôt possible, 2=normal à ordonnancer dans 50% de l'ordonnancement au plus tôt, et >2=à ordonnancer au plus tard).
- `removeProces(pos)` : enlève un processus en cours d'ordonnancement
- `next()` : changement de contexte, activer le processus le plus prioritaire

Vous ajouterez tout ce qui vous semble nécessaire pour modéliser complètement votre `Round Robin`. Parmi les axiomes, on n'oubliera pas d'effectuer un lien avec l'ADT `Processus`. Un bon exemple est l'axiome indiquant le fait qu'un unique processus est actif à un moment donné.

Question 2 : Contrat du processus suivant

Donner la forme générale d'une classe d'implémentation pour l'ADT `RoundRobin` (signature, attributs). On prendra soin de vérifier le ou les invariants de l'ADT. On donnera également le contrat complet de la méthode `next()` (pré et post-conditions, corps de la méthode).

EXERCICE 3 : LOGIQUE DE HOARE (30MN)

Question 1 : Alternative

Chercher la plus faible précondition P dans le programme suivant:

```
{P}  if(x>y) { x=x+y+a; y=2a-2x } else { y=y+2x; x=x-y  } ; x=x+y { x+y=a}
```

On détaillera les différentes éapes employées, sans oublier le nom des règles utilisées dans la logique de Hoare (parmi (aff) (alt) (seq) (mp) (mp-pre) (mp-post) (let) et (while))

Question 2 : Boucle

Prouver la correction ou l'incorrection de la spécification suivante :

```
{n>=0 } x=n; y=0; while(x>0) { y=y+x; x=x-1 }  { y = somme(0,n) }
```

On détaillera encore une fois les étapes suivies pour la preuve. On n'oubliera pas, en particulier, de donner précisément l'invariant le plus adapté pour la boucle while.

Remarque : somme(a,b) =a+(a+1)+...+b (hypothèse : a<=b)