Examen - INF460EX INF460   
POA (Programmation Objet Avancée)

Responsable : Pr. Xavier Blanc – Université de Bordeaux 1 (Xavier.Blanc@labri.fr)  
Le 16 décembre 2010, 14h – 17h  
3 pages - Tous documents autorisés

# Questions de cours (4 points) :

1 Le premier principe des design pattern est de programmer via les interfaces. Pour quelle raison ?

2 Lorsque l’appel à une opération peut lancer plusieurs exceptions, l’ordre des clauses catch est-il important ? Pourquoi ?

3 Expliquer la signification de l’interface Serializable.

4 Peut-on synchroniser deux Thread (à l’aide du mot clé Synchronized) alors qu’ils ne partagent pas d’objet ? Pourquoi ?

Correction :

1. L’objectif est de ne pas dépendre de l’implémentation (de la classe). Les dépendances doivent être vers des définitions de fonctionnalité et non pas vers du code d’implémentation.

Barême : 100% si mention de la séparation de l’implémentation (ou classe)

2. Oui c’est important. Chaque exception a un type. L’ordre des clauses catch doit suivre la hierarchie de type (le plus précis en haut et le moins précis Exception en bas). Sinon, une clause catch ne sera pas executée.

Barême : 100% si mention du type des exceptions et explication de l’ordre à suivre

3. Serializable est un simple marqueur. Il permet de préciser que les objets instances d’une classe qui réalise Serializable pourront être sauvegardé dans un fichier.

Barême : 100% si explication du marqueur pour sauvegarde.

4. Non cela n’est pas possible. Car il faut que les Thread partagent un objet afin que celui-ci serve de verrou.

Barême : 100% si mention du verrour avec un objet.

# Questions conception objet (4 points) :

Une application de commerce électronique est classiquement composée d’une classe Panier (représente le panier électronique) et d’une classe Produit (représente la description du produit qui est à la vente).

|  |  |
| --- | --- |
| **public** **class** Produit {  private **int** prix;  private String description;    **public** Produit(**int** prix) {  **this**.prix = prix;  }    **public** **int** getPrix() {  **return** prix;  }  } | **public** **class** Panier {  private List<Produit> produits;    **public** Panier() {  produits = **new** LinkedList<Produit>();  }    **public** **void** ajouterProduit(Produit p , **int** quantité) {  //**TODO}**    **public** **void** enleverProduit(Produit p , **int** quantité) {  //**TODO**}    **public** **int** calculerPrix() {  //**TODO**}  } |

1 Complétez les //TODO de la classe Panier (vous pouvez faire toutes les modifications que vous jugerez nécessaires).

2 Appliquer un Pattern Factory sur la classe Panier afin qu’un client puisse demander la construction d’un nouveau panier.

Correction :

1. La difficulté principale vient de la « quantité » à gérer. Il faut absolument effectuer les modifications 2 suivantes :

Ajouter la classe LigneDeCommande qui se trouve entre Produit et Panier. Cette classe peut aussi être une Hashtable dans Panier.Il aussi possible (c’est moins élégant mais ca marche) d’ajouter autant de produit qu’il y a de quantité.

LA CLASSE Produit NE PEUT PORTER LA QUANTITE

Ajouter les codes nécessaires afin que l’ajout d’un produit + quantité soit possible et que la somme soit possible

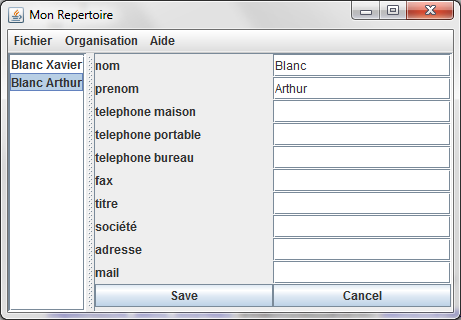
Barême : 100% si la solution est bonne.

2. Cette question n’est pas compliquée. Il suffit d’appliquer le pattern Factory sur Panier. Ajouter une classe PanierFactory et lui mettre la méthode createPanier

Barême : 100% si le pattern est bien codé.

# Questions Interface Graphique (4 points)

L’application suivante permet de gérer des contacts.



1 Lister tous les objets graphiques (dont la classe est un JComponent)

2 Décrivez (à l’aide d’un arbre) la structure des Layout de cette interface graphique

3 Lorsque l’on clique sur le bouton Save, le nom et le prénom du contact se mettent à jours dans la partie gauche de l’interface graphique. Donner le code permettant ce traitement (introduisez toutes les classes que vous jugez nécessaires et soyez exhaustif).

Correction :

1. Il faut absolument identifier les objets suivants :

La fenêtre principale (JFrame), la bare de menu et les menu JMenu, le conteneur en 2 parties (JSplitPane), la partie de gauche avec la liste (JList), la partie de droite (JPanel ou autre qui contient tous les champs), les 10 JLabel et les 10JTextField, les 2 JButton. Il y a d’autres objets graphiques mais de moindre importance.

Barême : 50% pour la structure (la fenêtre, le menu, les conteneurs de gauche et de droite), 50% pour les éléments utiles (bouton, zone de texte, label, etc.).

ATTENTION, plusieurs copies ne font que mentionner les classes. C’est Hors Sujet. Il faut lister les objets !

2. il reprendre les objets qui forment la structure de l’application (la fenêtre, le conteneur de gauche et le conteneur de droite). Il faut identifier que la fenêtre contient le menu et le splitpane dans son rootPane. Puis le splitpane qui contient à gauche la liste et à droite tous les champs dans un gridlayout.

Barême : +50% si la structure est à peu pres présente, +50% les objets sont bien présentés et si les layout sont bon (split pane, flow à gauche, grid à droite).

3. Il n’y a que peut d’information pour répondre à cette question. Il faut donc créer une classe qui réalise ActionListener et coder l’opération actionPerformed. Dans ce code, il faut mettre la modification du champ correspondant (on peut considérer qu’on connait sa référence)

Barême : 50% pour la structuration (Listener + ActionPerformed + Inscription du listener), 50% pour le code.

# Questions Génériques (3 points) :

La classe Viewer<T> hérite de JLabel. Elle permet de construire des JLabel à partir de n’importe quel objet. Le texte affiché dans le JLabel correspond à l’appel de la méthode toString() sur l’objet.

La méthode main de la classe MainViewer manipule une collection de Viewer<T>. L’objectif est de passer cette collection à une JFrame afin d’afficher tous les JLabel. Pour autant, ce code ne compile pas !

1 Expliquer pourquoi ce code ne compile pas.

2 Donner la solution pour faire en sorte que ce code compile.

|  |  |
| --- | --- |
| **public** **class** Viewer<T> **extends** JLabel {  T source;  **public** Viewer(T source) {  **super**();  **this**.source = source;  setText(source.toString());  }  } | **public** **class** MainViewer {  **public** **static** **void** main(String[] args) {  Viewer<String> view1 = **new**  Viewer<String>("nom:");  Viewer<Integer> view2 = **new**  Viewer<Integer>(3);  List<Viewer<Object>> viewers = **new**  LinkedList<Viewer<Object>>();  viewers.add(view1);  viewers.add(view2);  }  } |

Correction :

1. Ce code ne compile pas car il y a un problème de typage. List<Viewer<Object>> viewer = new LinkedList<Viewer<Object>>(). En effet, Viewer<String> n’hérite pas de Viewer<Object> ni Viewer<Integer>

Barême : 100% si mention du problème de sous typage

2. Il suffit de remplacer Object par « ? »

Barême : 100% si utilisation du « ? »

# Questions Fichier et Thread (5 points)

On souhaite développer une application qui permet d’absorber un site web afin d’en faire une cartographie (class Cartographie). Cette application lit une page HTML racine (index.html) et cherche à lire les pages HTML qui y sont liés. Pour ce faire, l’application cherche les lignes qui contiennent la chaine de caractère suivante <A href=’’XXX.html’’> où XXX correspond à une page HTML liée. Ainsi l’application peut absorber récursivement toutes les autres les pages HTML d’un site web, pour peu que celles-ci soient liées directement ou indirectement à la racine.

Nous considérons que cette application s’exécute sur un système de fichier local (nous ne considérons pas les appels http).

|  |  |
| --- | --- |
| **public** **class** Cartographie {  **private** Page root;  **public** **void** setRoot(Page root) {  **this**.root = root;  }  } | **public** **class** Page {  **private** String fileName;  **private** Set<Page> hrefs;  **public** Page(String fileName) {  **this**.fileName = fileName;  hrefs = **new** HashSet<Page>();  }  **public** **void** addHref(Page href) {  hrefs.add(href);  }  } |

1 Coder cette application. Il faut construire une classe qui réalise l’opérations : public Cartographie buildCarto(String rootPath)

2 Quelles sont les exceptions que cette application peut lancer ? Traitez ces exceptions.

3 On souhaite multi-threader cette application pour faire en sorte que les liens d’une page soient lus par plusieurs Thread. Proposez une solution multi-threadée.

Correction :

Cet exercice est volontairement ouvert.

1. En principe, il suffit de coder l’opération buildCarto qui va lire dans le fichier afin de detecter les patterns (<A href>). Dès qu’un pattern est detecté, on créé un objet page et on l’ajoute à la carto. Il faut ensuite relancer le buildCarto sur le fichier correspondant.

Barême : 40% si ouverture du fichier (30%) et recherche du pattern (10%). +40% si l’objet Page est bien construit et bien ajouté à la carto. +20% si on ouvre le fichier suivant et si on fait attention à ne pas boucler ☺

2. Essentiellement des exceptions d’ouverture de fichier (FileNotFound etc.)

Barême : 100% si bonne gestion des exceptions

3. Cette question est très délicate. L’idéal serait de faire un pool de Thread. Néanmoins, on peut accepter le fait de construire un Thread par fichier.

Barême : +50% si les Thread sont construit intelligemment (il faut faire attention à ce que chaque Thread lise son propre fichier. +50% si les objets pages sont bien construit (Synchronized).