## **RÉPONSES AUX QUESTIONS**

### **Question 2.1 : Identifiez l’intention et les avantages du patron Visiteur.**

Nous avons vu en classe que l’intention du patron Visiteur s’agit de « permettre de définir une nouvelle opération sans modifier aux classes des objets sur lesquels l’opération va agir ». En effet, cette façon de faire est permise puisque le patron Visiteur est capable de séparer l’algorithme des objets sur lesquels ils opèrent dessus. Ce patron offre alors une meilleure flexibilité puisqu’il y a indépendance entre les Visitors et la structure d’objets. Puis, étant donné que les méthodes du Visitor se retrouve tous dans une classe, c’est-à-dire un endroit commun, il y a ce qu’on appelle « fonctionnalité localisée ».

### **Question 2.2 : Tracer un diagramme de classes avec EA pour chacune des deux instances du patron Visiteur.**

### **Question 2.3 : Si en cours de conception, si vous constatiez que vous voudriez ajouter une nouvelle sous-classe dérivée de *AbsTeamComponent*, établissez la liste de toutes les classes qui doivent être modifiées.**

En regardant le code, il est possible de discerner que les sous-classes de *AbsTeamComponent* puisse accepter des visiteurs, grâce à la méthode *accept()*. Pour cela, si jamais la tâche listée dans la question était exécutée, habituellement, il faudrait ajouter une méthode *visit()*, mais dans notre cas, ce serait une *process()* à la classe abstraite de visiteur, ainsi que tous les autres visiteurs afin qu’ils puissent visiter cette nouvelle sous-classe. En gros, il s’agirait d’une nouvelle fonction de traitement, ou bien d’une méthode de visite. La liste de toutes les classes qui devront être modifiées se compose donc de :

* AbsComponentVisitor;
* TeamImageSizeCalculator;
* MemberTextFindReplace.

### **Question 2.4 : Selon vous, la fonction d’ajout d’un rôle pour un membre d’une équipe pourrait-elle être implémentée comme un visiteur? Si oui, discuter des avantages et des inconvénients d’utiliser le patron visiteur pour cette fonction et sinon expliquez pourquoi le patron n’est pas applicable.**

À notre avis, oui, en effet, cette fonction pourrait bel et bien être implémentée comme un visiteur. Ce dernier hériterait des méthodes de visite de la classe abstraite, et se composerait également de d’autres méthodes en lien avec sa fonction principale d’ajout. Encore une fois, le fait d’utiliser le patron Visitor a pour avantage que le code pour cette fonction d’ajout ne soit pas dispersé, mais plutôt dans une seule classe. Puis, comme dicté dans les notes de cours, cela n’affectera pas la hiérarchie des classes. Malheureusement, il y a également des désavantages à cette utilisation; par exemple, il y aura présence de duplication du code des visiteurs.

### **Question 3.1 : Identifiez les points suivants :**

### **L’intention et les avantages du patron Commande.**

Selon les notes de cours, l’intention de ce patron est « d’encapsuler une requête dans un objet de façon à permettre de supporter facilement plusieurs types de requêtes, de définir des queues de requêtes et de permettre des opérations « annuler » ». De cela, on comprend qu’il s’agit d’un patron, où de l’information pour effectuer différentes actions est contenu dans un seul objet. Parmi les avantages de ce patron, il y a le fait que le code soit plus extensible étant donné qu’il est possible de faire des ajouts de nouvelles commandes sans à modifier le code source. Le couplage entre l’objet qui invoque la requête de celui qui sait comment la satisfaire diminue aussi. En d’autres mots, il y a découplage.

### **La structure des classes réelles qui participent au patron Commande ainsi que leurs rôles (faire un diagramme de classes avec Enterprise Architect, ajouter des notes en UML pour indiquer les rôles, et exportez le tout en pdf).**

### **Question 3.2 : Observez attentivement la classe *CommandInvoker* qui permet de gérer la relation entre les commandes et les différents membres des équipes. En plus de participer au patron Commande, cette classe participe à deux autres patrons de conception vus en cours.**

### **Quels sont les noms et les intentions de ces patrons de conception?**

### **Quels sont les éléments de la classe *CommandInvoker* qui sont caractéristiques de es patrons de conception?**

### **Pourquoi avoir utilisé ici ces patrons de conception?**

### **Question 3.3 : Pour compléter la fonctionnalité de *TeamViewer*, il faudrait ajouter de nouvelles sous-classes de la classe *AbsCommand*. Selon vous, est-ce que d’autres classes doivent être modifiées pour ajouter les nouvelles commandes? Justifiez votre réponse.**

### **Question 3.4 : Dans la version proposée de la commande de traduction par la classe *CommandTranslate*, on suppose que la traduction d’une langue vers l’autre peut être effectuée de façon « inversible », c’est-à-dire sans perte d’information durant la traduction. Selon vous, cette supposition est-elle réaliste? Sinon, que faudrait-il faire pour rendre l’opération « undo » robuste si on ne peut pas supposer que l’opération est inversible? On ne vous demande pas d’implémenter une solution au problème, mais simplement de décrire le problème et la façon dont vous vous y prendriez pour le régler.**