IUT Paris Descartes – Département Informatique – 2e année

Conception et Programmation à Objets Avancées

Mikal Ziane, Mourad Ouziri, Karim Foughali

TP 1

Thèmes : impact des changements, spécification vs implémentation d'un type, polymorphisme

Dans la suite du texte une **entité** est une méthode, une classe, une interface, ou un paquetage.

Exercice 1 révisions sur l'encapsulation

Récupérez le code de l'exercice 1 sur le serveur. Il s'agit d'une classe Pile qui a été écrite sans encapsuler ses données.

Que peut-on dire sur les problèmes du code client (ici la fonction `main`) qui ne respectent pas l'encapsulation ?

Supposons qu'on remplace les données de la classe Pile par une liste chainée (LinkedList) quelle est la conséquence sur le code client ?

Corrigez la classe Pile et réécrivez le code client tant qu'il est petit. Quels bénéfices en a-t-on tiré ?

Exercice 2 révisions sur le polymorphisme

Récupérez le code de l'exercice 2 sur le serveur. Il s'agit d'un embryon d'application de vente en ligne des différents articles : livres, dvd, etc.

Remarquez que plusieurs tests unitaires ne passent pas (ils sont mis en commentaire). Il faut les faires passer. De plus les méthodes `getPourTous` et `memeArticle` sont très mal écrites.

Quels sont les défauts de ces méthodes ?

Corrigez ces problèmes.

Que constate-t-on des méthodes corrigées ?

Exercice 3 (à finir si nécessaire la semaine suivante)

Le logiciel est censé être facile à modifier (**soft**ware vs. **hard**ware). Pourtant nous allons considérer plusieurs fragments d'applications pour lesquels **certains** changements sont difficiles ou coûteux en termes d'effort de développement.

Considérez le code de l'exercice 3 fourni sur le serveur (disque commun). Le programme saisit dans un premier temps une série de noms d'espèces d'animaux. Dans un second

temps le programme affiche le cri associé à chaque nom de la série. Pour comprendre ce TP il est essentiel de tenir compte du fait que le code source de ce programme ne fait qu'une trentaine de lignes alors que le code source d'un logiciel peut compter plusieurs dizaines voire centaines ou millions de lignes.

Estimez l'impact des changements prévus

Vous allez apporter un certain nombre de changements à ce programme et pour chacun noter l'effort qui a été nécessaire.

Estimez cet effort en nombre de lignes de code et d'entités à lire pour trouver les occurrences du problème et le nombre de lignes de code et d'entités à modifier. Estimez aussi la proportion de code non pertinent à éliminer par rapport au code à modifier ? Plus cette proportion est importante plus le risque d'erreur (oubli, confusion …) et plus la difficulté (pour se concentrer sur ce qui doit être modifié) sont importants.

Généralisez en anticipant l'évolution de l'application en proposant une formule puis calculez là pour CA=10 classes d'animaux de chacune MA=10 méthodes de chacune LA = 5 lignes de code et CC = 100 classes clientes de chacune MC = 15 méthodes de chacune LC = 10 lignes.

Changement A : modifier la capacité d'accueil pour une des espèces

Changement B : remplacer les tableaux d'animaux par des ArrayLists.

Changement C : remplacer le nom des animaux par un numéro de tatouage

Changement D : ajouter une espèce d'animal

Changement E : changer d'interface utilisateur

Proposez des solutions pour réduire l'impact des changements prévus

Pour chacun des changements prévus proposez une solution et estimez à nouveau l'impact de ce changement après application de la solution.

Expliquez pourquoi chaque solution permet de limiter l'impact des changements.

Estimez l'effort pour appliquer la solution : quand la solution est-elle justifiée ?

En fonction du temps disponible : **appliquez les solutions** proposées. En principe on ne remanie (*refactor*) pas du code sans filet de sécurité (sans tests) donc **il faudrait au préalable ajouter des tests unitaires.** Si vous manquez de temps vous vérifierez la correction des modifications en lançant des sessions de contrôle.