EXAMEN TERMINAL – SESSION 1

GÉNIE LOGICIEL ET GESTION DE PROJET

DURÉE : 1H30

SUPPORTS DE COURS ET DE TD AUTORISÉS

le barème est donné à titre indicatif
une mauvaise orthographe, grammaire et présentation entrainent des malus

A. Questions de cours (10 points)

1. Qu'est-ce qu'une fonctionnalité transversale (crosscutting concern) et comment la programmation orienté aspect (AOP) permet-elle d'implanter ces fonctionnalités plus efficacement ?

R : Une fonctionnalité transversale est une fonctionnalité du programme distribuée sur plusieurs classes qui est difficilement décomposable du reste du système auquel elle est associé, que ce soit en terme de conception ou d'implémentation (ex: persistance, logging,…). Le résultat est un code éparpillé (et dupliqué), et emmêlé (avec des dépendances entre parties qui ne devraient pas l'être).

La programmation orienté aspect permet d'extraire la définition de ces fonctionnalités du code métier, et de les regrouper au sein d'un aspect. Tout en permettant leur utilisation via la définition de points de coupe dans le code métier ou pourra être inséré le code.

2. En quoi les annotations Java facilitent-elles la programmation par configuration ?

R : La programmation par configuration vise à isoler données de configuration de l'application (paramètres en entrée, paramètres codés en dur) et code, à abstraire les paramètres du programme et les traitements répétitifs liés aux données, et à standardiser. Avec les annotations java, il est possible définir un processus d'injection de la configuration sous forme d'AnnotationProcessor, qui injectera les bons paramètres aux endroits du code voulus.

3. Lister 3 avantages et 3 inconvénients des méthodes agiles

Avantages

- Concepts intégrés et simples
- Pas trop de management
- Gestion continuelle du risque
- Estimation permanente des efforts à fournir
- Insistance sur les tests : facilite l'évolution et la maintenance

Inconvénients

- Passe difficilement à l'échelle

- Risque d'avoir un code pas assez documenté

- Risques de design peu générique, pas beaucoup d'anticipation des développements futurs

4. Peut on utiliser de la modélisation UML avec les approches Agiles, si oui comment, si non qu'utilise t'on à la place.

- La modélisation vise avant tout à comprendre et à communiquer

- Modéliser pour les parties inhabituelles, difficiles ou délicates de la conception.

- Rester à un niveau de modélisation minimalement suffisant

- Modélisation en groupe

  - Outils simples et adaptés aux groupes

  - Les développeurs créent les modèles de conception qu'ils développeront

5. À quoi correspond la notion de couverture dans les tests ?

R: Aux éléments de l'application testés.
Comment choisir les bons scénarios?

Idéalement, il faudrait qu'ils soient exhaustifs mais cela est généralement impossible.
Ils doivent donc permettre d'exécuter un maximum de comportements différents de l'application

- Couverture des cas dits nominaux : les cas de fonctionnements les plus fréquents
- Couverture des cas limites ou délicats
- Entrées invalides
- Montée en charge
- Sécurité

6. Quelle(s) différence(s) entre tests boite noire ou boite blanche ?

Test fonctionnel ou boîte noire: la sélection se fait en se basant sur la spécification

- On teste ce que doit faire l'application

Test structurel ou boîte blanche/ transparente: la sélection se fait en se basant sur le code

- On teste la manière dont l'application le fait

B. Études de cas (6 points)

Pour les deux cas ci-dessous, répondre aux trois questions:
1. Quel design pattern est le plus approprié, et dans quel but ?
2. Pourquoi ce pattern répond au problème ?
3. Comment ce pattern peut être implémenté ? Illustrer avec un diagramme de classe (pas de code). N'utilisez pas le diagramme de classe générique, mais un spécifique au cas en question. Suivez les conventions des diagrammes de classe UML.

1. POINT

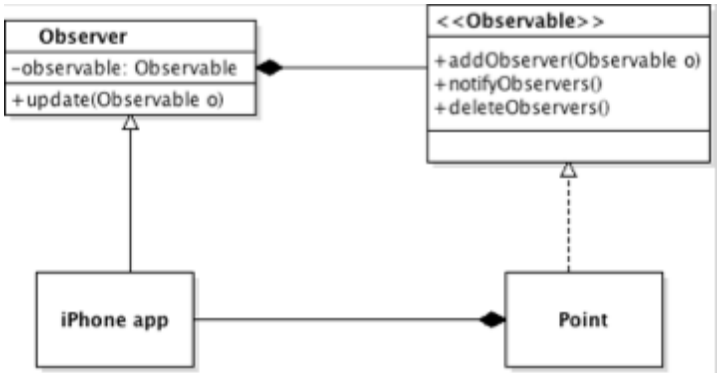
Point est un nouveau dispositif IoT de sécurité dédié à la maison. Equipé de plusieurs capteurs il écoute le bruit ambient d'une maison, analyse l'environnement et notifie ses utilisateurs si quelque chose de surprenant se passe. Par exemple une fenêtre cassée, une alarme qui sonne, la présence de fumée, etc. Le dispositif peut être installé n'importe ou dans la maison et connecté au wifi.

Une application peut être installée sur n'importe quel plateforme mobile (iOS, Android) pour pouvoir être notifié si quelque chose sortant de l'ordinaire se passe. À terme, d'autres dispositifs IoT de la marque Point (serrure connectée, babyphone, alarme incendie) pourront à terme être connectés, et envoyer des messages sur la même application. Si jamais trop de messages arrivent, les utilisateurs garderont la possibilité de pouvoir faire en sorte de ne plus les recevoir.

1. Observeur - définit une dépendance un vers plusieurs (one to many) entre objets pour qu'à chaque changement d'état, tout ceux qui en dépendent soit notifiés et mis à jour automatiquement.

2. Le dispositif Point permet d'avoir plusieurs applications ou objets qui s'écoutent et ce pattern permet aux objets de communiquer entre eux pour maintenir une cohérence globale et faible couplage. Les objets ou application peuvent souscrire à un ou plusieurs objets sans connaitre en détail l'objet observé.

3.Le diagramme.



2. HÉBERGEVERT

L'entreprise HébergeVert offre trois services d'hébergement : perso, pro, et premium. Pour chaque offre, plusieurs services supplémentaires peuvent être ajoutés (SSL, support 24/24, sauvegardes, Google Adwords, CloudFlare, mySQL).

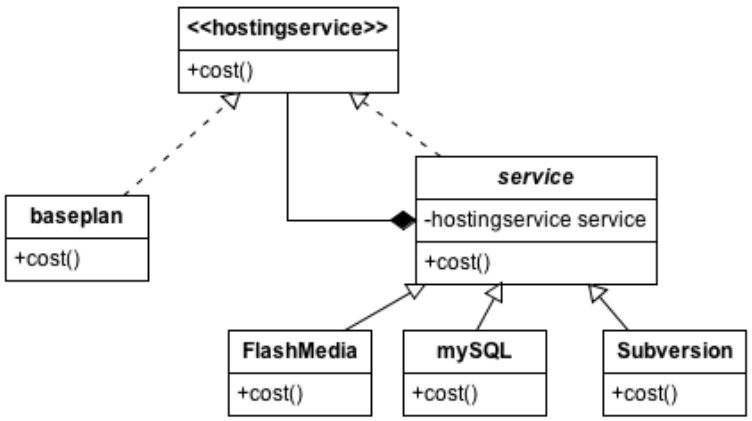
Vous devez concevoir une application qui permette de calculer facilement le coût mensuel du service, pour chaque offre en tenant compte de la combinaison des services supplémentaires. Votre application doit pouvoir permettre d'ajouter facilement de novelles offres quand elles deviennent disponible. Le tout en évitant une explosion de la quantité de classes.

Une combinaison choisie par un client pourrait être :
> Hébergement perso, SSL, Google Adwords, MySQL: 50 euros par mois.

1. Décorateur - on attache dynamique une responsabilité additionnelle à un objet. Le pattern fournit une alternative flexible aux sous-classes pour étendre les fonctionnalités.

2. S'il fallait implémenter toutes les combinaisons possibles, on verrait une explosion du nombre de classes. Ici on ajoute une seule classe qui ajoutera les nouveaux services.

3. Le diagramme



C. Implémentation d'un design pattern (4 points)

Nommer et décrire le design pattern implémenté par cette classe. Expliquer en quoi ce que fait cette classe en vous concentrant sur la méthode invoke

```
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

public class OwnerInvocationHandler implements InvocationHandler {
	PersonBean person;
	public OwnerInvocationHandler(PersonBean person) {
		this.person = person;
	}
	public Object invoke(Object proxy, Method method, Object[] args)
			throws IllegalAccessException {
		try {
			if (method.getName().startsWith("get")) {
				return method.invoke(person, args);
			} else if (method.getName().equals("setHotOrNotRating")) {
				throw new IllegalAccessException();
			} else if (method.getName().startsWith("set")) {
				return method.invoke(person, args);
			}
		 } catch (InvocationTargetException e) {
		    e.printStackTrace();
		 }
			return null;
	}
}
```

On est ici sur le design pattern Proxy.

InvocationHandler est utilisé pour gérer toutes les invocations de méthodes sur les objets de la classe "proxifiée" (ici PersonBean). La méthode invoke reçoit la méthode et les arguments, et n'autorise l'accès qu'aux getters et setters sur l'objet original (person) sauf pour setHotOrNotRating, parce que seul un possesseur de l'objet aurait le droit d'utiliser cette méthode (dans ce contexte donné).