Lycée Léonard de Vinci *Section Technicien Supérieur en Informatique SIO Option SLAM*

MISSION PPE – SLAM Semaine du 8 octobre 2012

Objet : Réfactoring - Maintenance préventive sur un projet

Le problème

L'application GeTAP initiée au cours du printemps 2011 entrera bientôt en phase de déploiement. Nous souhaitons que le code source exprime de façon plus claire, plus communicante, les différentes transitions responsables des changement d'état de l'objet central de cette application, à savoir les demandes de validation de consommation de temps d'accompagnement personnalisé (DVCTAP) représentées par la classe `DemandeValidationConsoTempsAccPers`.

La plateforme de travail

Pour les besoins de cette mission, un morceau de l'application a été extrait de son contexte technique (hors certaines dimensions techniques du web : pas de mvc, spring, tomcat, jsp...)

Le travail préliminaire

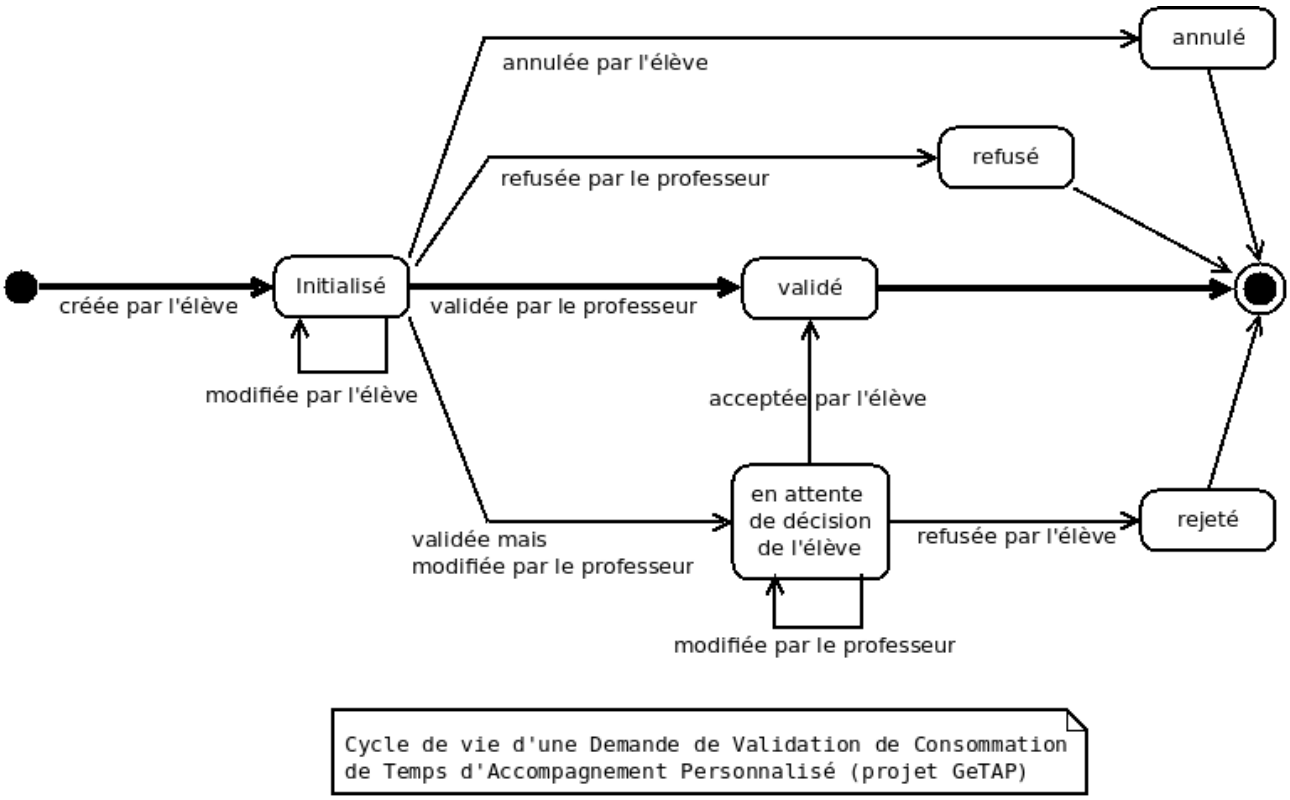
(durée approximative : 20mn à 1h)

1. Vous devez disposer d'un compte GitHub personnel (il est toujours temps de le créer)
2. Puis, à partir de ce dernier, vous cloner en ligne ce projet https://github.com/sio-melun/TPGetapTU
3. Ensuite vous réaliser un import de votre clone à partir d'Eclipse (le plugin Egit est supposé installé)
4. Votre environnement de travail est considéré maintenant comme opérationnel.

Le travail demandé

Phase 1 : prise en main de l'existant

D'un point de vue logique, la demande, par l'élève, d'une validation de consommation de temps d'accompagnement personnalisé peut passer par différents états, représentés par le diagramme d'état-transition ci-dessous :



Le pseudo-état à l'extrême gauche se nomme « état initial » et le pseudo-état à l'extrême droite « état final » (n'est plus source de transitions).

Dans la classe `DemandeValidationConsoTempsAccPers` l'état (initialisé, refusé, annulé, rejeté, en attende de validation élève, validé) est représenté par un attribut de type entier nommé `etat` (voir l'Annexe et le code source ou javadoc pour connaître les valeurs attendues).

1/ Réaliser un diagramme de classes UML de l'existant. Les **getter/setter** ne sont pas exigés.
(durée approximative : 20mn)

2/ Concevoir une classe nommée `Exemple`, qui instancie une DMCTAP, d'après cette déclaration :

```
DemandeValidationConsoTempsAccPers dvctap;
```

Avec les données ci-dessous :

L'élève : `Nizar Ben Ragdel`

La classe : `SIO22`

L'accompagnement personnalisé : `« Salon du libre »` d'une durée de 4heures

Le prof concerné: Olivier Capuozzo

La date : le 07/10/2012 (astuce : utilisez la méthode static *valueOf* de java.sql.Date)

(durée approximative : 20mn)

3/ Dans le *main* de la classe `Exemple`, passer l'objet référencé par `dvctap` par les différents états suivants : (volontairement sans cohérence entre eux) du statut « demande créée » à « demande modifiée par l'élève », puis « demande validée par le professeur » puis à nouveau par « demande modifiée par l'élève » ensuite « durée modifiée par le professeur » et « demande rejetée par l'élève » pour finir par « demande validée par le professeur ».

=> à chaque changement d'état, la (ou une) représentation textuelle de l'objet sera affichée.

(durée approximative : 20mn)

***Phase 2* : Refactoring avec tests unitaires**

(C'est la mission professionnelle proprement dite)

(durée approximative : Une séance de 4H de TP)

L'objectif est d'augmenter la cohérence, la lisibilité d'une partie stratégique du projet. Vous concevrez des tests unitaires pour réaliser ce travail, dans une approche *Test First*.

1. Le projet ne dispose pas d'une branche de test, vous devez créer une branche `test` parallèle à `src`
2. Créer une classe de test unitaire (dans la branche `test`) avec, comme classe de test : `DemandeValidationConsoTempsAccPers.`
3. Initialiser l'instance, objet du test, avec les mêmes données utilisées dans le programme Exemple.
4. Vérifier que l'état de l'instance est correct dès sa création (voir exemple plus bas).
5. Contrôler le fait que les différents chemins du cycle de vie (valeur *etat* de l'instance) sont possibles, et **interdire tout autre chemin**. Vous respecterez les consignes suivantes :

    - Dans le cas d'un chemin non autorisé, une exception sera lancée de type **DVCTAPException** (classe déjà présente dans le projet).

    - Toute transition sera représentée par une méthode (public) dédiée.
      Par exemple, la transition « refusée par le professeur » sera représentée par la méthode `refuseeParLeProfesseur()` de la classe `DemandeValidationConsoTempsAccPers`.

      Attention : les méthodes retournant l'état d'un attribut de type booléen sont préfixées par ***is*** et non *get*.

    Exemple de test unitaire où `dvctap` est instancié dans le *setup* (ou `@Before` selon la version de Junit) :

```
@Test
public void testEtatInitial() {
      assertTrue("Etat initial", dvctap.isEtatInitial());
}
```

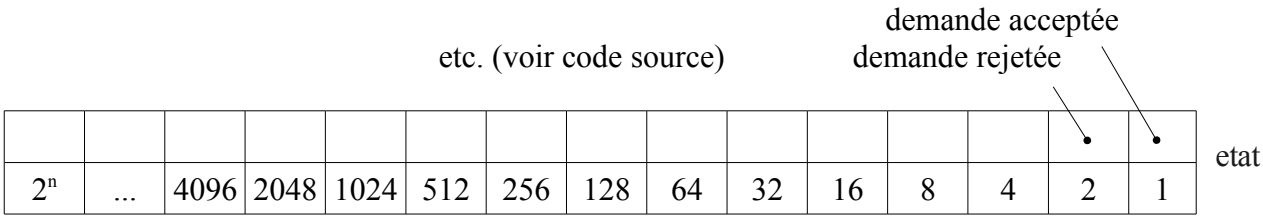
Ce qui est attendu :

Un rapport contenant

- Le diagramme de classe
- Les tests unitaires
- Les nouvelles méthodes de `DemandeValidationConsoTempsAccPers`, commentées à la **javadoc**
- Le javadoc du projet (voir dossier doc/ du projet)

ANNEXE

Exemple de représentation des états (de niveau bit)



2^n	...	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

Pour savoir si `etat` est positionné à « demande rejetée » il suffit d'appliquer le bon masque avec un ET binaire. Exemple :

```
private static final int DVCTAP_REJETEE = 2;


boolean rejetee = (this.etat & DVCTAP _REJETEE) != 0;
```

En effet si on applique un ET avec un masque ne contenant qu'un bit à 1, deux résultats sont possibles :

- Si le bit correspondant de la variable est positionné à 1, il restera à 1 (1 & 1 == 1)
- Si le bit correspondant de la variable est positionné à 0, le restera sera 0 (0 & 1 == 0)

Variante de l'expression booléenne :

```
boolean rejetee = (this.etat & DVCTAP _REJETEE) == DVCTAP _REJETEE;
```

Pour modifier la valeur de la variable `etat` on appliquera à cette dernière un OU binaire, avec un masque composé d'un bit à 1 placé au bon endroit.