TE Génie Logiciel - GEN

Nom : Cortès Léo

Date : 9 Avril 2018

1	2	3	4	5	6	7	8	9	10	12	13
1	3	3	2	3	1.5	4	4.5	5	8,5	7	0
1	3	3	2	3	3	4	6	5	12	8	2

9 13 20,5

42,5 / 52 / 48

5,5

EXERCICE 1 – 1 POINT

Mandatée par le client X, notre société de développement doit se déterminer quant au cycle de vie à mettre en œuvre dans le cadre du projet à réaliser.

Ce projet ne présente pas de risque particulier. Nous avons réalisé avec succès un projet très similaire il y a deux années en arrière. Que choisirons nous ? (Cochez une seule réponse)

1/1

- ○ Cycle de vie itératif avec méthode agile
- ○ Cycle de vie en spirale
- ☒ Cycle de vie en V (cascade)

EXERCICE 2 - 3 POINTS

- 1.5 POINT PAR RÉPONSE FAUSSE

Historiquement, la « gestion des risques » a été introduite dans le modèle du cycle de vie dit « en spirale ». Que signifie « gestion des risques » ? Cocher la ou les proposition(s) correcte(s).

3/3

- ☐ Prendre le risque de développer un système pour lequel le groupe n'a aucune expérience de développement
- ☐ Prendre le risque de développer un fragment de code avant que l'analyse ne soit spécifiée globalement et de manière détaillée pour tout le système
- ☒ Donner la priorité aux éléments présentant un certain risque

EXERCICE 3 - 3 POINTS

- 1.5POINT PAR RÉPONSE FAUSSE

Voici quelques problèmes caractérisant le modèle de cycle de vie en V: Cocher la ou les proposition(s) correcte(s).

3/3

- ☒ Le développement est trop linéaire et le feedback de la phase d'implémentation est tardif.
- ☐ Du point de vue des tests, Vérification et Validation ne couvrent que la phase d'implémentation.
- ☐ N'autorise qu'une décomposition fonctionnelle et ne permet pas une modélisation orientée objet.

EXERCICE 4 - 2 POINTS

2/2

La méthode UP s'appuie sur quelle) notation) ? UML

La méthode UP s'appuie sur quel type de programmation ? Orienté Objet

9

Velocité - jour-homme * focal

40 4*2*5

focal = 100%

2

EXERCICE 5 - 3 POINTS

- 1.5 POINT PAR RÉPONSE FAUSSE

Acteurs et cas d'utilisation. Parmi les assertions suivantes, cochez celles qui reflètent correctement le concept d'acteur.

3/3

- ☐ Un rôle peut être associé à plusieurs personnes physiques, mais la même personne physique sera associée à un rôle au maximum.
- ☒ L'héritage entre acteurs implique un héritage des droits
- ☐ Un acteur externe est situé dans un endroit géographiquement éloigné du système à développer

EXERCICE 6 - 3 POINTS

Sachant qu'un point d'histoire correspond à un jour idéal de travail/homme.

Une équipe Scrum de 4 personnes travaille avec des sprints de 2 semaines. Sa vélocité a la valeur 40. Tous les membres de l'équipe travaillent à 100%, à raison de 5 jours par semaine.

1.5/3

Toutes les histoires que l'équipe a planifiées sont toutes estimées (par hasard) à 2 point d'histoire.

Question :

1. Combien d'histoires peuvent-ils espérer terminer dans le courant d'un sprint ? 20 ✓
2. Que vaut le facteur de focalisation de l'équipe (en %)? ~~10%~~ 100%

EXERCICE 7 - 4 POINTS

- 1.5 POINT PAR RÉPONSE FAUSSE

4/4

Parmi les propositions énoncées ci-dessous, cochez la ou les proposition qui vous paraissent correctes.

- ☒ Deux threads Java de la même application peuvent s'exécuter en vrai parallélisme
- ☒ Deux threads Java peuvent se partager des instructions
- ☒ Deux threads Java peuvent se partager les mêmes variables
- ☐ La machine virtuelle de Java est responsable de l'ordonnancement des threads et notamment du time-slicing

EXERCICE 8 - 6 POINTS

- 1.5 POINT PAR* POINTS *FAUSSE

4.5/6

Le vrai et le faux dans les Méthodes agiles
Parmi les assertions suivantes, cochez celles qui reflètent correctement la notion de méthode agile.

- ✓ ☐ On désire obtenir un cahier des charges entièrement spécifié au terme de la phase d'analyse.
- ✓ ☐ Les méthodes agiles souhaitent que l'équipe soit constituée de programmeurs « seniors » (avec expérience), susceptibles d'écrire directement leur code de manière propre et définitive.
- -1.5 ☒ On désire en premier lieu pouvoir s'adapter aux changements concernant les besoins des utilisateurs. En revanche, le processus même de la méthode agile (étapes, gestion, outils, ..) est clairement spécifié.
- ✓ ☒ Contrairement à UP, les méthodes agiles préconisent de développer l'architecture centrale du logiciel non pas au début du développement mais au fur et à mesure des besoins.
- ✓ ☒ La planification des itérations est un processus évolutif.
- ✓ ☐ Avec Scrum, le fait de raccourcir la durée des sprints a le mérite d'augmenter le rythme des feed-backs client et de réduire par la même occasion le nombre annuel de stand-up meetings.

13

EXERCICE 9 - 5 POINTS

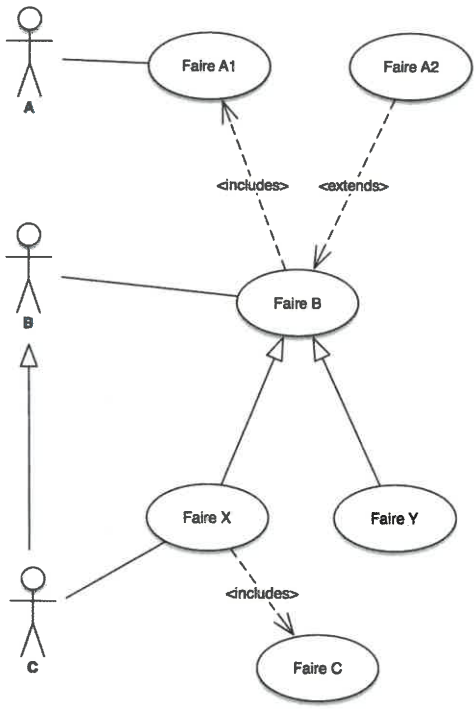
- 1.5 POINT PAR RÉPONSE FAUSSE

Considérez le diagramme de cas d'utilisation présenté ci-dessous. Parmi les propositions écrites ci-dessous, cochez celles qui respectent le diagramme.

Cocher les propositions correctes.

- ☒ L'acteur B peut `Faire X`
- ☒ `Faire Y` nécessite `Faire A1`
- ☐ `Faire B` nécessite `Faire A2`
- ☐ `Faire B` nécessite `Faire C`
- ☒ L'acteur C peut `Faire Y`

5/5



5

EXERCICE 10 – 12 POINTS

Le code de la classe `Chrono` présentée ci-dessous a pour effet de créer un chronomètre constitué de 3 compteurs (centièmes, secondes et minutes), de démarrer ce chronomètre et d'afficher en console son nouvel état, tous les centièmes de secondes, sous la forme d'un string *minutes secondes centièmes*.

```
0 0 1
0 0 2
0 0 3
:
0 0 99
0 1 0
0 1 1
:
```

Le programme s'appuie sur 3 classes :

`Chrono` : Cette classe vous est donnée, ne pas la modifier

`Counter` : A créer entièrement par vos soins

`ChronoView` : Cette classe vous est donnée, la compléter le cas échéant directement sur l'énoncé

Notes sur la classe `Counter` (un modèle) :

- Cette classe est instanciée 3 fois, une fois pour le compteur des centièmes, une fois pour le compteur des secondes et une fois pour le compteur des minutes.
- Le constructeur de la classe `Compteur` accepte 4 paramètres
  - `pulse : int`
    - si `pulse <= 0`
      --> Le compteur attend un signal extérieur pour s'incrémenter
    - si `pulse > 0`
      --> Le compteur est un objet actif, qui s'incrémente tout seul tous les « `pulse` » millièmes de secondes :

```
while(true) {
    :
    Thread.sleep(pulse) ;
    :
    Incrémentation du compteur
    :
}
```

  - `maxValue : int`
    Le compteur recommence à zéro quand cette valeur maximale est atteinte. Au départ, le compteur est initialisé avec la valeur 0.
  - `nextCounter : Counter`
    - si `nextCounter == null`
      --> Le compteur est en bout de chaine : Le fait d'avoir atteint sa valeur maximale n'est notifiée à personne
    - si `nextCounter != null`
      Une notification est envoyée à `nexCompteur` dès que la valeur maximum `maxValue` est atteinte.
  - `view : Observer`
    Cette vue est notifiée dès que le compteur change de valeur

Que faire ??

- Ecrire le code de la classe `Counter`
- Compléter le cas échéant le code de la classe `ChronoView` directement sur l'énoncé

Contraintes à observer impérativement

- Toutes les notifications (entre compteurs, avec leur vue associée) sont opérées par le biais du modèle Observable/Observé
- Par soucis d'efficacité, les objets `Observer` ne doivent pas être notifiés par des événements qui ne les concernent pas.

API Observable/Observé --> **Voir annexe**

```
class Chrono {

      public static void main (String ... args)  {
            ChronoView view = new ChronoView();
            Counter cents, seconds, minutes;
            minutes = new Counter(0, 60, null, view);
            seconds = new Counter(0, 60, minutes, view);
            cents = new Counter(10, 100, seconds, view);
            view.setCounters(minutes, seconds, cents);
            cents.go();
      }
}
class ChronoView implements Observer {

      private Counter[] counters;

      public ChronoView () {}

      public void setCounters(Counter ... counters){
            this.counters= counters;
      }

      public void renderChronoState() {
            String txt = "";
            for (Counter cnt : counters) txt += cnt.getValue() + " ";
            System.out.println(txt);
      }

      public void update (Observable o, Object arg) {
            renderChronoState();
      }
}
```

EXERCICE 11 – 8 POINTS

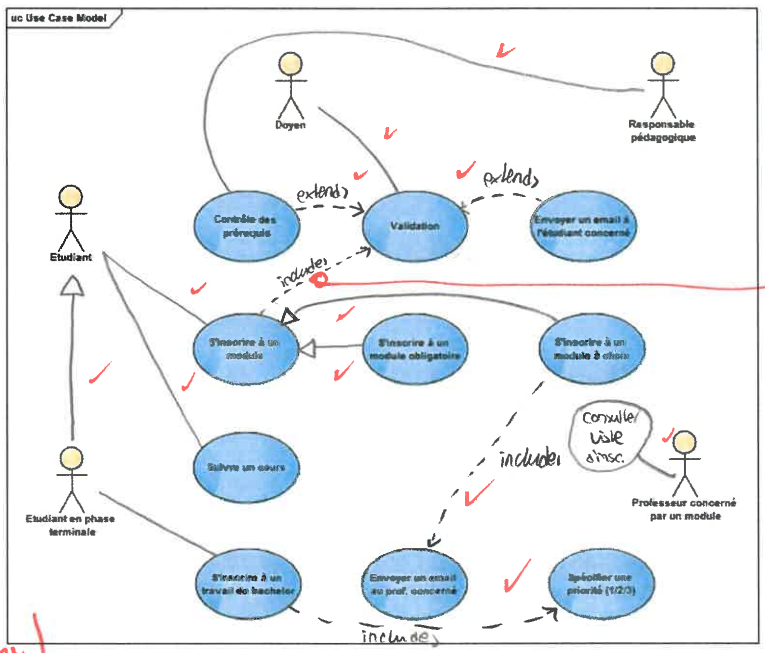
Dessinez le diagramme de cas d'utilisation correspondant à la description suivante.

- Un étudiant peut s'inscrire à un module, il peut s'agir d'un module à choix ou d'un module obligatoire.
- L'inscription à un module sera validée dans les jours qui suivent par le doyen. Suivant le type de module, cette validation peut nécessiter un contrôle des prérequis avec l'aide du responsable pédagogique. En cas de refus, un mail est envoyé à l'étudiant et l'inscription est annulée.
- L'inscription à un module à choix s'accompagne de l'envoi d'un e-mail au professeur concerné. Le professeur concerné peut consulter la liste des inscriptions.
- Un étudiant peut suivre un module. Ce suivi est possible uniquement si son inscription a été validée par le doyen.
- Un étudiant en phase terminale peut faire tout ce que peut faire normalement un étudiant. En plus, il a toutefois la possibilité de s'inscrire à un travail de bachelor en indiquant une priorité entre 1et 3.

7/8

Question a) Complétez le diagramme ci-dessous.

Il faudrait juste ajouter qqch pour savoir si la validation est positive ou non (Sorte de condition pour suivre le cours) (Prérequis) à expliquer dans la description du cas d'utilisation



prérequis ! -1

Question b) Classification des acteurs

Pour chacun des acteurs, dites si cet acteur est un acteur principal, secondaire ou encore externe du point de vue du cas d'utiliation auquel il est relié

L'Etudiant (et Etudiant en phase terminale) est un acteur principal. Il agit directement avec le système. Le Doyen aussi.

Le Responsable pédagogique est un acteur secondaire (il assiste le doyen)

Le Professeur concerné est un acteur externe, il ne fait que consulter le système sans agir directement dessus.

EXERCICE 12 - 2 POINTS

Le père Noël distribue ses cadeaux. Les enfants accourent.

Mais il faut attendre son tour ! Les enfants attendent leur tour en invoquant `attendreMonTour`. Celui qui vient d'obtenir son cadeau invoque la méthode `auSuivant`.

```
class PereNoel {
   private ilEstLibre = false ;
   public synchronized void attendreMonTour() {
      if (!ilEstLibre) try {wait() ;}
      catch (InterruptedException) {}
      ilEstLibre = ! ilEstLibre;
      System.out.println(ilEstLibre) ;
   }
   public synchronized void auSuivant() {
      ilEstLibre = true;
      notifyAll() ;
   }
}
```

Quelle sera la valeur affichée par `attendreMonTour` ? Justifiez votre réponse.

false

Chaque enfant entrera tour à tour dans attendreMonTour (car elle est synchronized).
Pour le premier enfant, admettons que ilEstLibre soit true (sinon il ne pourra jamais passer et les autres non plus*).
Il va ensuite le passer à false puis afficher false.
Il va ensuite le repasser à true avec auSuivant()

0/2

Pas toujours false ! (if → while)

Presque toujours false
mais parfois true (rarement)

*et donc pas d'affichage

/5

ANNEXE – API OBSERVABLE – OBSERVÉ

API Observer

```
interface Observer {
      public void update(Observable o, Object arg);
}
```

Paramètres:

o - l'objet observé (utile si le même observateur observe plusieurs objets différents)

arg - un argument passé par la méthode `notifyObservers`

API Observable

`public void` **`addObserver`**`(Observer o)`

Pour ajouter un nouvel observateur à la liste des observateurs de l'objet

`protected void` **`setChanged`**`()`

Pour enregistrer le fait que l'état de cet objet a été modifié: passage du flag interne «`modified`» à l'état `true`.

`public void` **`notifyObservers`**`()`

Pour notifier tous les observateurs que cet objet a été modifié. L'argument `arg` reçu par la méthode `update` de l'observateur vaudra `null`.

Cette notification a lieu si et seulement si la méthode **`setChanged`** a été invoquée au préalable.

`public void` **`notifyObservers`**`(Object arg)`

Idem que **`notifyObservers`**(). L'argument `arg` sera reçu par la méthode `update` de l'observateur (deuxième paramètre de la méthode `update`).

Léo Cortès

```
class Counter extends Runnable, Observable implements Observer {
    private int pulse;
    private int maxValue;
    private Counter nextCounter;
    private Observer view;
    private int val;
    private Thread activity;

    public Counter ( int pulse, int maxValue, Counter nextCounter, Observer view){
        val = 0;
        if (pulse > 0) {
            activity = new Thread(this);
        }
        this.pulse = pulse;
        this.maxValue = maxValue;
        this.nextCounter = nextCounter;
        this.view = view;
        this.addObserver(view);   // La vue sera notifiée
        if (nextCounter != null) {
            this.addObserver(nextCounter);   // Le compteur suivant aussi!
        }
    }

    public void increment ( ) {
        val ++;
        if (val >= maxValue ) {
            val = 0;
            ~~nextCounter.increment();~~
        }
        setChanged();
        notifyObservers( (Integer) val );   // Notifie la vue et le compteur suivant
    }

    public void update (Observable o, Object arg )
        if (((Integer) arg ) == 0 ) {
            increment();   // Si on atteint 60 secondes, on a une minute de plus
        }
    }

    public void run () {
        while (true) {
            Thread.sleep(pulse);
            increment();
        }
    }
```

-1

et méthode go() ? (création → démarrage du thread)

Dans votre solution, les notifications sont envoyées à tout le monde quel que soit le type d'événement

try .. catch -1

```
public int getValue() {
    return val;
}
}
```

Leo Coits

Evaluation sur 16 points

Compteur : 14 Points

Lien avec la vue

Une classe Observable pour les changements de valeur	1
Un objet correspondant	1
Ajout de la vue en observateur	1
La vue est notifiée si et seulement si « pulse » millièmes écoulés	1

Lien avec le compteur suivant

Une autre classe Observable pour l'événement valeurMax	1
Un objet correspondant	1
Ajout du compteur suivant en observateur	1
Le compteur suivant est notifié i et seulement si valeurMax atteinte	1

Le compteur est un observateur

implements Observer	1
Implémentation de update	1

Création d'un objet actif

Si pulse > 0	1
Implémentation d'une méthode run	1
Création du Thread	1
start du thread dans la méthode go()	1

ChronoView : 2 points

implements Observer	1
Implémentation de update	1

Points obtenus	Note finale
16	12
15	11.5
14	10.5
13	10
12	9
11	8.5
10	7.5
9	7
8	6
7	5.5
6	4.5
5	4
4	3
3	2.5
2	1.5
1	1

13 -2 (Mdl.)

→ 11/16

→ 8,5/12