Projet Ascenseur

Logiciel ascenseur
Revue du sprint 2

Dominique Ribouchon – Décembre 2016

Déroulement de la revue

- Rappel des objectifs du sprint 2 et démarche suivie
- Démonstration du logiciel ascenseur (réponse aux exigences)
- Présentation de la conception du logiciel ascenseur

§3

Objectifs du sprint 2 - Rappels

- **Objectif**: Refactoring du code pour améliorer sa maintenabilité et évolutivité (qualité interne) =>
  - Sur le même périmètre fonctionnel (i.e. même exigences)
  - Application des bonnes pratiques de conception fondamentales
  - Exprimer clairement la conception du logiciel: préoccupation des classes et dépendances entre les classes(utilisation d'UML)
- **Début-Fin**: 24/11-01/12
- **Revue de sprint:** 02/12
  - Démo – conformité aux exigences
  - Explication claire de la conception du code et recherche de pistes d'amélioration de la maintenabilté et de l'évolutivité du code

Démarche adoptée pour réaliser le sprint

- Nous avons réalisé les activités d'ingénierie logicielle suivantes:
  - Définition des Exigences/Tests de qualification (vision externe du logiciel ascenseur):
    - Même périmètre fonctionnel qu'au sprint 1
    - Un simple rappel et réappropriation des exigences/tests de qualification du sprint 1 a été suffisant
  - Conception (vision interne du logiciel ascenseur) – application des bonnes pratiques:
    - **Séparation des préoccupations** (qualité interne)
    - **Limitation des dépendances** (qualité interne)
    - **Cohérence** de l'ensemble (qualité externe)
  - Codage du logiciel ascenseur et intégration à la classe SimulateurDriver
  - Exécution des tests de qualification

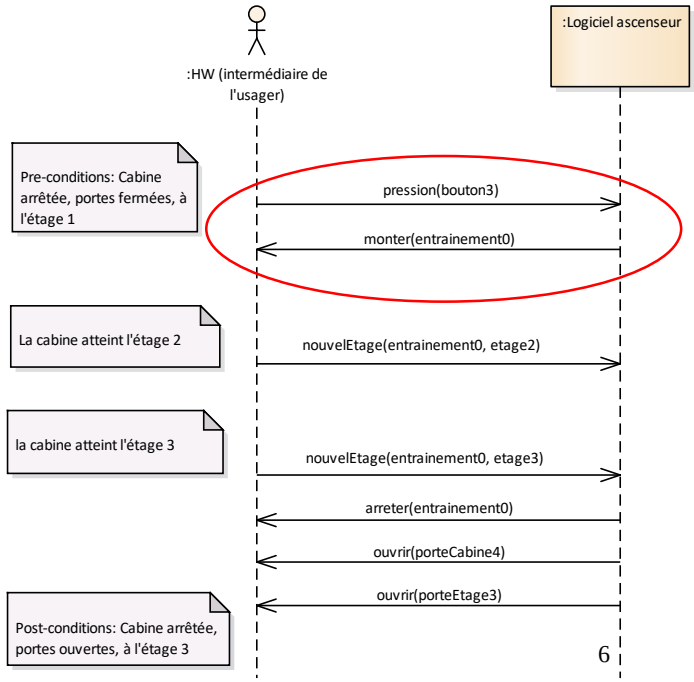
Exigences à réaliser – rappel du périmètre fonctionnel (1/5)

- Le périmètre fonctionnel est le même que celui du sprint 1
- Les exigences à réaliser sont le début du scénario "Normal - 1 appel autre étage" du cas d'utilisation "Appeler l'ascenseur",
- Ce scénario a été adapté aux contraintes matérielles (HW et registres de communication avec le HW non disponibles)
- Ce scénario adapté constitue le scénario de qualification du sprint

§3

Exigences à réaliser – rappel du périmètre fonctionnel (2/5)

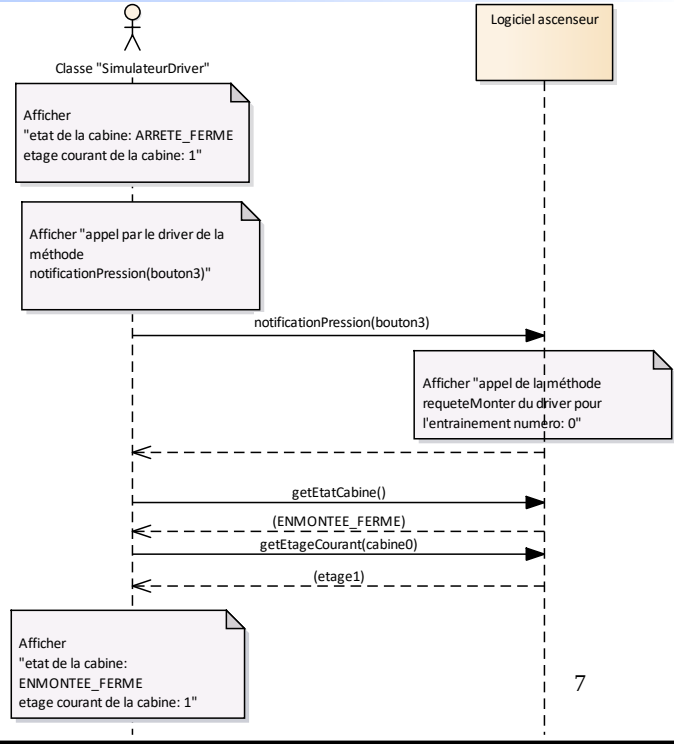
- Scénario de niveau exigences:



§3

Exigences à réaliser – rappel du périmètre fonctionnel (3/5)

- Scénario de niveau tests de qualification:



§3

Exigences à réaliser – rappel du périmètre fonctionnel (4/5)

- Le logiciel à réaliser doit s'intégrer à la classe SimulateurDriver fournie:

```
package test_logiciel_ascenseur;
import logiciel_ascenseur.*;
public class SimulateurDriver {

    public static void main(String[] args) {
        // Initialisation: Ascenseur a 4 etages, dont le RDC
        ???.creerBouton(0,0);
        ...

        //Execution du scenario par simulation des drivers
        System.out.println("...
        ???.notificationPression(3);

        //Affichage état cabine - vérification post-conditions
        System.out.println("etat ...
    }
}
```


§3

Exigences à réaliser – rappel du périmètre fonctionnel (5/5)

- Sortie console:

```
run:
etat de la cabine: ARRETE_FERME
etage courant de la cabine: 1

appel par le driver de la méthode notificationPression(bouton3)
appel de la méthode requeteMonter du driver pour l'entrainement numero: 0

etat de la cabine: ENMONTEE_FERME
etage courant de la cabine: 1
```


§3

Démonstration

- La démonstration consiste:
  - à exécuter le scénario de tests de qualification
  - à l'aide de l'IDE Netbeans

§3

Conception du logiciel ascenseur (1/8)

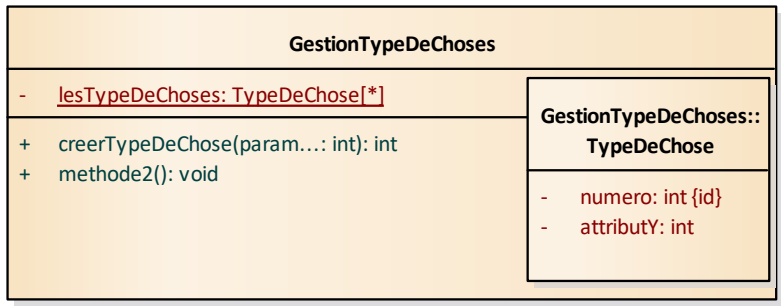
- Le logiciel a s'exécute sur un ordinateur de bureau standard (PC ou Mac)
- Le code est réalisé dans le langage Java
- Afin de garantir sa qualité, nous avons appliqué les bonnes pratiques de conception

Conception du logiciel ascenseur (2/8)

- Pratique 1- Séparation des préoccupations en unités de code:
  - Chaque unité de code possède une responsabilité claire, reflétée par son nom
  - La responsabilité d'une unité de code n'est pas partagée avec une autre unité (i.e. pas de duplication de code)
  - Les données et les traitements associés sont définis dans la même unité, qui est responsable de l'intégrité de ses données: les données sont privées
  - Structurer par "type de choses à gérer"
- Pratique 2- Limitation des dépendances:
  - Eviter la redondance de données
  - Pas de dépendance directe sur les données
  - **Eviter le plat de spaghetti de dépendances entre les unités de code**
- Pratique 3: Cohérence de l'ensemble - réponse aux exigences

Conception du logiciel ascenseur (3/8)

- Conformément aux recommandations de l'architecte logiciel, et afin de garantir une bonne homogénéité du code, nous avons utilisé le pattern de référence dans l'entreprise "Gestionnaire d'objets statique"



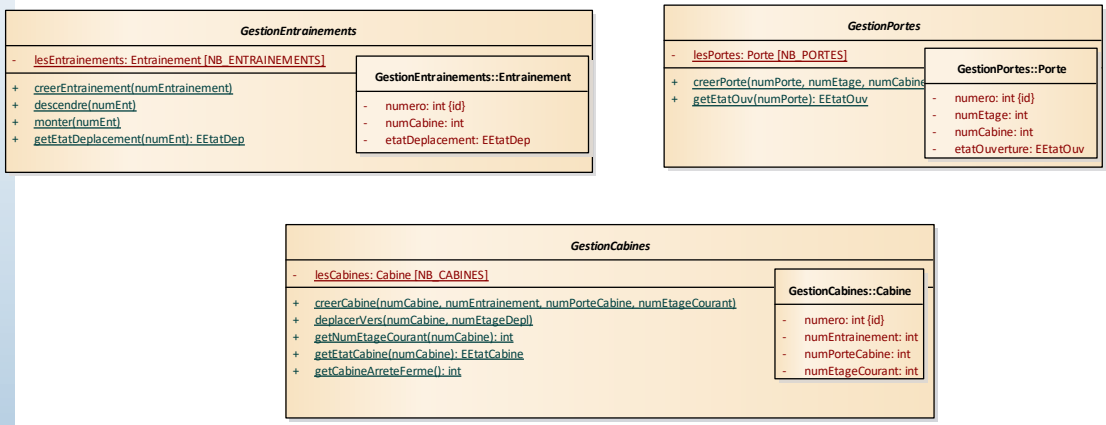
§3

Conception du logiciel ascenseur (4/8)

- Le logiciel ascenseur est divisé en 6 unités de codes, chacune ayant la responsabilité/préoccupation de gérer "un type de choses:
  - GestionPortes,
  - GestionBoutons,
  - GestionEntrainements (évolution plusieurs entrainements anticipée)
  - GestionCabines (évolution plusieurs entrainements anticipée)
  - GestionEtages
  - GestionServices (gestion des appels et des sélections – évolution anticipée)

Conception du logiciel ascenseur (5/8)

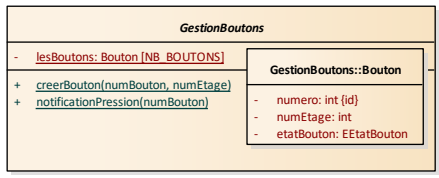
- Séparation en 6 préoccupations (1/2):

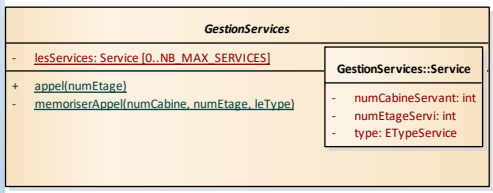


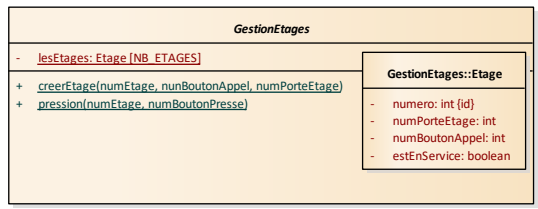
§3

Conception du logiciel ascenseur (6/8)

- Séparation en 6 préoccupations (2/2):



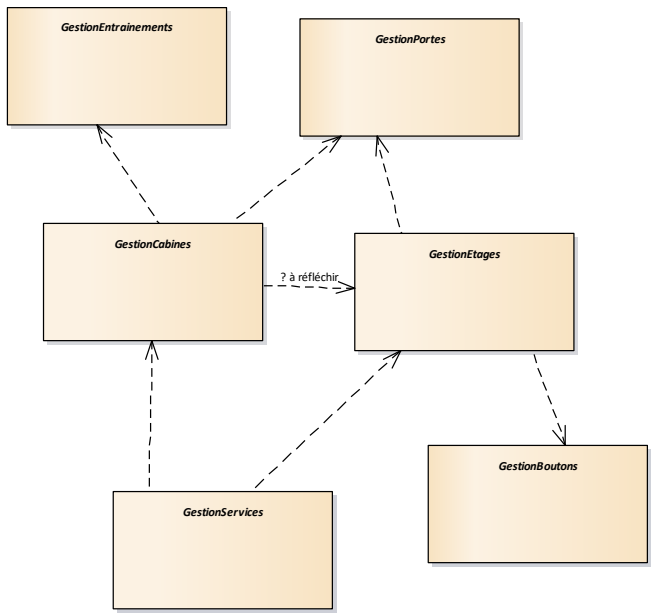




§3

Conception du logiciel ascenseur (7/8)

- Les dépendances "pressenties":



Conception du logiciel ascenseur (8/8)

- Cohérence de l'ensemble (réponse aux exigences fonctionnelles) : avons-nous bien anticiper la cohérence de l'ensemble?

Bonnes pratiques de conception – Synthèse

- Pratique 1- Séparation des préoccupations en unités de code:
  - Chaque unité de code possède une responsabilité claire, reflétée par son nom
  - La responsabilité d'une unité de code n'est pas partagée avec une autre unité (i.e. pas de duplication de code)
  - Les données et les traitements associés sont définis dans la même unité, qui est responsable de l'intégrité de ses données: les données sont privées
  - Structurer par "type de choses à gérer"
- Pratique 2- Limitation des dépendances:
  - Eviter la redondance de données
  - Pas de dépendance directe sur les données
  - Eviter le plat de spaghetti de dépendances entre les unités de code
- Pratique 3: Cohérence de l'ensemble - réponse aux exigences

§3