Rapport de laboratoire

**Département de génie logiciel et des technologies de l’information**

|  |  |
| --- | --- |
| **No de laboratoire** | 2 |
| **Étudiant(s)** | François Gagné  Patrick Lavallée  Julien Pelletier-Morin  Simon Turcotte |
| **Code(s) permanent(s)** | GAGF20067801  LAVP12048408  PELJ03098909  TURS17118905 |
| **Cours** | GTI525 |
| **Session** | Hiver 2013 |
| **Groupe** | 01 |
| **Professeur(e)** | Éric Boivin |
| **Chargé(e) de laboratoire** | Mohamed Outellou |
| **Date** | CHANGE LA DATE! |

Table des matières

[1 Introduction 3](#_Toc350607843)

[2 Diagramme de classe 4](#_Toc350607844)

[2.1 Le Patron *Transfer Object* 4](#_Toc350607845)

[2.2 Architecture des tests unitaires 5](#_Toc350607846)

[2.3 Injection de dépendance 6](#_Toc350607847)

[3 Discussion 7](#_Toc350607848)

[4 Conclusion 9](#_Toc350607857)

Table des figures

[Figure 1 - Patron Transfer Object 4](#_Toc350607871)

[Figure 2- Architecture des tests unitaires 5](#_Toc350607872)

[Figure 3 - Patron architecturale d'injection de dépendance 6](#_Toc350607873)

# Introduction

Dans le cadre de ce cours, l’équipe doit concevoir et déployer un système de vente de billet en ligne. Il s’agit de la deuxième itération du projet qui vise à dynamiser l’expérience utilisateur notamment en ajoutant une couche applicative qui répondra à des requêtes HTTP.

Lors de la première itération, un prototype statique a été élaboré. Ce prototype est dynamisé en transformant les pages statique au format JSP et en incluant des mécanismes d’obtention de données ce trouvant sur un serveur distant. Ces données sont pour l’instant « *stubbé »*. De plus, la plateforme *Spring* est mise de l’avant et favorise une approche MVC qui est un standard de développement Web.

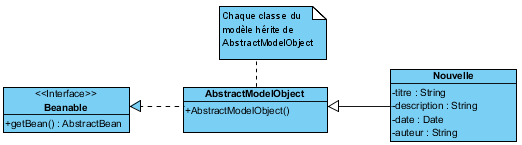
De plus, le diagramme de classe est mis-à-jour et est une évolution des concepts élaborés. Finalement, une discussion est fournie et démontre les difficultés et les bons coups de l’équipe.

# Diagramme de classe

Le diagramme de classe initial fournis pour la première itération du projet a évolué. En effet, plusieurs concepts ont été revisité et bonifié afin d’offrir plus de souplesse au système en développement.

## Le Patron *Transfer Object*

Le patron « *Transfer Object »* est utilisé comme monnaie d’échange entre un *JavaBean* et un objet du modèle. Pour ce faire un *JavaBean* implémente l’interface *Modelable* et en retour, l’entité implémente l’interface *Beanable*. De cette façon, l’information peut être transigée de la vue au modèle pour y subir les transformations propre aux logiques métiers.



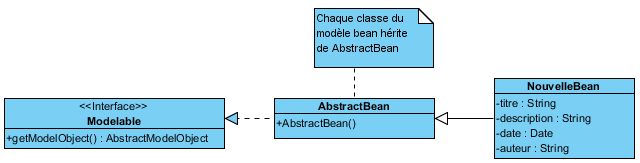


Figure 1 - Patron Transfer Object

## Architecture des tests unitaires

En utilisant les outils offerts par la plateforme de test JUnit4, les fonctionnalités critiques de cette itération ont été identifiées, testées et inclus dans une suite de test automatisée. Une suite de test maîtresse, nommé *AllTests* constitue le point d’entrée. Les suites de tests enfants sont invoquer par cette dernière qui à leurs tours appelent leurs méthode de tests.

La suite de test fournis avec cette deuxième itération couvre l’ajout, la modification et la suppression de données au travers les interfaces gestionnaires. Elle couvre aussi l’implémentation du patron *« Transfer Object »*.

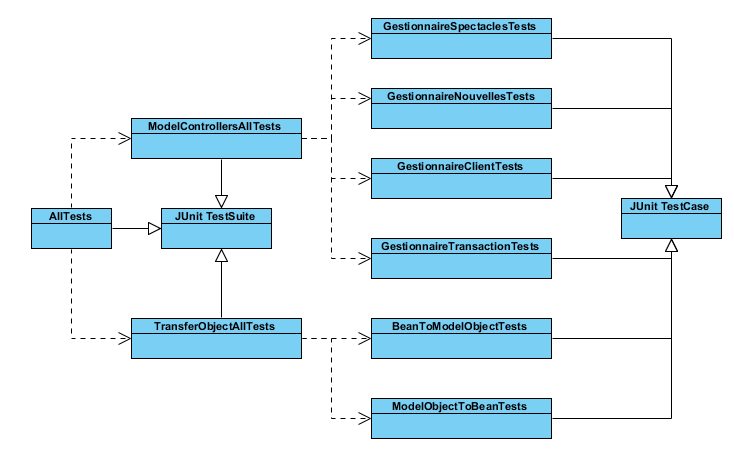
**

Figure 2- Architecture des tests unitaires

## Injection de dépendance

Le patron architectural d’injection de dépendance sert à isoler l’implémentation de la logique d’affaire. Le contrôleur, à l’aide de la méthode *getAccueil()* connais les méthodes proposées par l’interface qui elle est en mesure de faire le pont à l’implémentation concrète désirée.

Dans le cas de l’interface *IPresentationAccueil*, elle sert à isoler la logique de présentation de la page JSP au travers l’appel d’une méthode *presenterNouvelles()* qui s’occuperas de générer le code HTML nécessaire. L’utilisation du patron *View Helper* se fait à ce niveau.

Finalement, l’interface *IGestionnaireNouvelle* isole la logique métier qui accède au modèle et qui transforme les données. Cette logique est invoquée généralement en réponse à un évènement produit par l’utilisateur tel qu’obtenir la liste des derniers spectacles au chargement de la page.

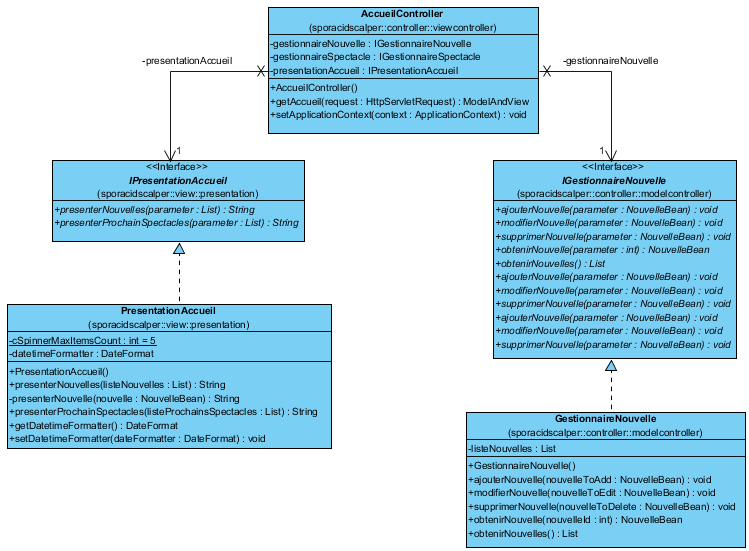


Figure 3 - Patron architecturale d'injection de dépendance

# Discussion

## Quel est la stratégie que vous avez utilisée pour associer un URL au contrôleur associé?

Chacun des contrôleurs est définis dans le fichier Web.xml associant le servlet approprié à l’URL demandée. Les méthodes de traitement sont affublé d’une annotation @RequestMapping qui offre plus de contrôle sur la gestion des URL.

# Quels patrons de conception Web ont été utilisés dans votre application? Si vous avez utilisé des frameworks, quels patrons sont implémentés par celui-ci?

Le patron *Transfer Object* à été mis en place dans une optique d’avoir une monnaie d’échange pour transiger les données de la page Web jusqu’aux classes Java. Cette approche ce veux évolutive afin d’appliquer une persistence aux données, soit l’objectif de la dernière itération du projet.

Le patron *View Helper* permet d’isoler la logique de présentation côté serveur et désengorge le contenu de la page JSP.

De plus, la plateforme Spring favorise une approche MVC et implémente elle-même sa propre version du patron *Front Controller* au travers du servlet *DispatcherServlet.*

# Expliquez à quel endroit vous avez utilisé AJAX dans votre application.

Lors de la sélection d’une représentation, un appel AJAX est fait afin d’obtenir les types de billet associés à cette dernière. Ceci dans l’optique que l’application soit modulaire et puisse éventuellement supporter plusieurs type de billet et non juste le cas d’admission générale.

# Avez-vous utilisé une technologie où un framework additionnel? Si oui, expliquez quel est le rôle de cette technologie dans votre application.

L’application web a été déployée en utilisant la plateforme Spring. Son rôle est principalement de renforcer le patron de conception MVC. Cette plateforme permet aussi d’implémenter le patron *Front Controller* au travers du *DispatcherServlet* qui agit à titre de répartiteur des requêtes HTTP entrantes.

L’utilisation de la plateforme de test JUnit4 c’est révélé fort utile. En effet, les comportements critiques de l’application ont pu être testés et corrigés en cas de besoin. Les suites de tests ont pu être bonifiées par la suite en leurs ajoutant d’avantage de logique de test.

# Suite à l'expérience du laboratoire 2, quelles sont les éléments du diagramme de classe du laboratoire 1 que vous auriez planifié différemment?

Aucun patron de conception n’était présent sur la version initiale du diagramme de classes. Sans toutefois être une lacune, l’équipe a du réviser l’approche en implantant certains patrons de conception tel que le *Transfer Object* ou le *View Helper*. Une couche d’abstraction du modèle a été rajoutée afin d’orchestrer les comportements standard de chacune des entités.

# Quel a été la principale difficulté technique rencontrée dans le laboratoire?

Vu l’ampleur de la tâche à accomplir il a été nécessaire de mettre en place un mécanisme de gestion des sources tel que SVN. L’intégration des sources et la gestion des conflits c’est avéré plus ardu qu’espéré. En effet, des erreurs de compilations ce sont insérées au fil du développement du a une mauvaise mise à jour des sources du projet. Il s’agit d’être vigilant et de ne rien briser lorsque que des nouveaux changements sont poussés au serveur central SVN.

L’utilisation d’une technologie comme GIT permettrais de décentraliser la gestion des conflits en empêchant un utilisateur de poussé des changements en conflit ou qui ne compile tout simplement pas.

# Comment a été faite la séparation des tâches dans votre équipe?

Chacun des membres de l’équipe c’est occupé de la dynamisation d’une section du site. Simon c’est chargé de produire un exemple à suivre pour toute l’équipe afin de standardiser le plus possible le code de l’application.

# De quelle manière avez-vous géré la gestion de la session?

La gestion de la session est découpée par fonctionnalités. Chacune des parties s’occupe de gérer son bout de session. Pour l’instant elle est séparée en deux partie soit :

* Message
* Panier d’achat

# Conclusion

Le but de cette deuxième itération consistait en la dynamisation de la solution logicielle déployé lors du laboratoire précédent. L’utilisation de la technologie JSP a permis à l’équipe d’insérer de la logique Java à l’intérieur d’un document HTML.

Suite à une contrainte imposée due au nombre de membre de l’équipe, la plateforme *Spring* a été la technologie centrale du développement du livrable de cette itération. Qui plus est, l’équipe a su tirer avantage d’inclure des tests automatisées tôt dans le cycle de développement permettant d’effectuer des tests de régression.

Le résultat est une application web permettant de consulter du contenu obtenue dynamiquement; qui envoie des requêtes HTTP vers un serveur qui traite la requête, obtient les données, applique la logique d’affaire et retourne un résultat interprétable par un navigateur.

Finalement, l’utilisation d’un *« Stub »* pour simuler la persistance des données a permis de bien découpler les classes logicielles impliquées pour permettre à l’application de migrer vers l’utilisation de la plateforme *Hibernate* lors de la prochaine itération.