# Rapport du Projet tutoré : Gestionnaire de festival

Sommaire

[I – Introduction 2](#_Toc421271568)

[II – Conception générale 3](#_Toc421271569)

[III – Conception détaillée 6](#_Toc421271570)

[IV – Tests 12](#_Toc421271571)

[V – Manuel Utilisateur 15](#_Toc421271572)

[VI – Conclusion 18](#_Toc421271573)

[VII – Annexe 19](#_Toc421271574)

I – Introduction

Dans le cadre de notre projet tutoré, nous allons, Benjamin Sanvoisin et Loïc Bonnet concevoir généralement le programme, puis nous concentrer à un niveau plus bas d’abstraction afin de dégager les problématiques et les enjeux du sujet.  
Le sujet nous demande principalement d’effectuer une interface de création, présentation et de réservation de différents festivals de musiques que nous pourrons proposer via une interface graphique dans un programme Java.  
Les différentes tâches que nous allons effectuer tout au long de ce projet seront la conception générale et détaillée, le développement, les tests, et la rédaction du rapport en parallèle avec toutes ces étapes.  
Nous avons choisi de concevoir globalement le programme et ensuite de commencer les prémices de la programmation avant la conception détaillée, afin de pouvoir évaluer avec précision les objets qui nous seront nécessaires à l’implémentation. A la suite du développement, nous allons revenir sur la conception détaillée afin d’affiner les objets et comportements.  
La conception se fera via le logiciel StarUML2 afin de pouvoir produire des diagrammes de classes, objets, et séquence dans la phase de conception détaillée.  
Nous effectuerons des tests (notamment pour la création d’intervalle de date d’un festival, affichage d’un festival et nombre de places total d’un festival) tout au long du développement afin de réduire les problèmes majeurs en fin de programmation.  
Enfin, nous allons utiliser un gestionnaire de version DCVS afin de faciliter la collaboration et ainsi partager équitablement les différentes phases de l'implémentation grâce au logiciel git sur GitHub.

II – Conception générale

# 

Reformulation du sujet :

Un festival peut être créé par une saisie d’un nom, une série de dates, un genre, une description, une image, un nombre de réservations, un prix de place. La réservation peut se faire via une interface, et les festivals sont représentés via un programme. L’interface de saisie comprendra des éléments éditables afin de créer des festivals. Un fichier permettra la sauvegarde des réservations liées aux festivals.

Tableau objets, états, comportements :

|  |  |  |
| --- | --- | --- |
| **Objets** | **Etats** | **Comportements** |
| date | Repère temporel caractérisé par un jour, un mois, une année | Précise un festival dans le temps |
| festival | Événement qui possède un nombre précis de places, un genre, une image, des dates particulières, un prix | Est réservable pour un nombre limité, se déroule sur plusieurs dates |
| affichage | Possède des éléments graphiques | Affiche les données dans la GUI et permet la saisie de donnée |
| programme | Contient des données concernant les festivals et les réservations | Permet la sauvegarde des réservations et l’ajout de nouveaux festivals |
| saisie | Possède des éléments graphiques éditables | Création et complétion de champs pour un festival |
| réservation | Possède des éléments graphiques éditables | Permet de réserver un festival existant |

Figure 1 : Tableau de conception générale

Classes dégagées :

L’étude de la reformulation du sujet et du tableau Objet/Etat/Comportement nous a permis de dégager plusieurs classes distinctes.

La classe Date (Abstraction de “date”).

La classe Festival (Abstraction de “festival”).

La classe Vue (Abstraction de “affichage”).

La classe Programme (Abstraction de “programme”).

La classe Saisie (Abstraction de “saisie”).

La classe Réservation (Abstraction de “réservation”).

Analyse générale :

On peut aisément, via le tableau, associer des classes. La classe Date est associée à Festival car elle constituera un champ de celui-ci. Un Festival est lié au Programme, et la classe Vue est associée aux classes Réservation, Saisie, et Programme qui seront des interfaces.  
Cependant il faudrait, à un niveau plus proche du code, détailler ces associations et partitionner la Vue en deux classes distinctes (FenetreMere, PanelFils), et le programme en trois classes (Programme, InterfaceProgramme, Fichier) afin de dissocier l’interface de la sauvegarde des données et du stockage des festivals.

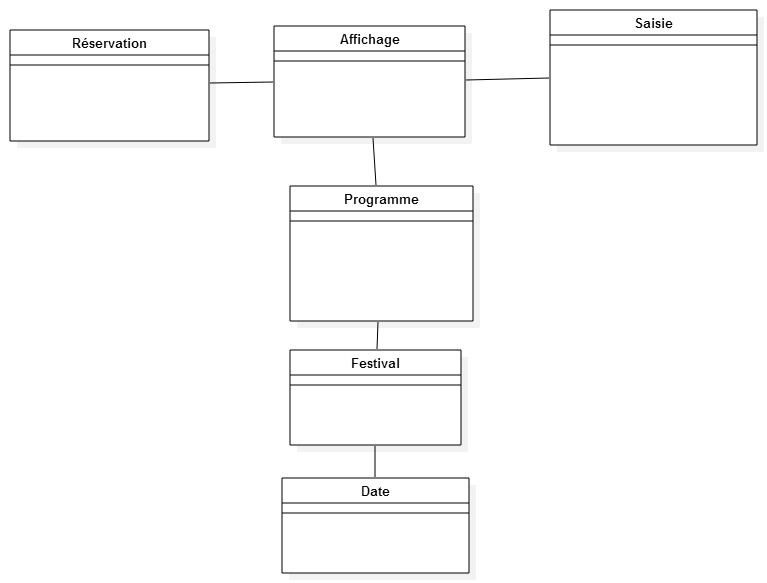


Figure 2 : Diagramme de classe du système à haut niveau d’abstraction

III – Conception détaillée

|  |  |  |
| --- | --- | --- |
| **Objets** | **Etats** | **Comportements** |
| date | Repère temporel caractérisé par un jour, un mois, une année | Précise un festival dans le temps |
| festival | Événement qui possède un nombre précis de places, un genre, une image, des dates particulières | Est réservable pour un nombre limité, se déroule sur plusieurs dates |
| panneau principal | Conteneur principal des interfaces (JFrame) | Instancie le panneau secondaire et crée la fenêtre principale |
| panneau secondaire | Conteneur secondaire des interfaces annexes (JPanel) Avec CardLayout | Instancie les interfaces annexes sous forme de menu (JMenu) |
| programme | Contient des données concernant les festivals et les réservations (HashMap) | Permet le stockage des réservations et l’ajout de nouveaux festivals |
| fichier | Possède un fichier associé que l’on peut lire et éditer | Permet la sauvegarde des réservations et des festivals. |
| interface de programme | Est caractérisé par une DefaultTableModel | Affiche les différents festivals selon les genres |
| interface de saisie | Possède des éléments graphiques éditables | Création et complétion de champs pour un festival |
| interface de réservation | Est caractérisé par une DefaultTableModel et boutons | Permet de réserver un festival existant |
| interface de réservation festival | Pop-up de l’interface de réservation | Permet la réservation d’un festival spécifique |

Figure 3 : Tableau de conception détaillée

L’abstraction à un niveau plus proche du code nous permet de dégager des classes supplémentaires afin de décrire avec précision les classes Vue et Programme décrites lors de la conception générale et précisées dans le tableau de conception générale.

Classes dégagées :

La classe Festival (Abstraction de “festival”),   
la classe Programme (Abstraction de “programme”),  
la classe Date (Abstraction de “date”),   
la classe Fichier (Abstraction de “fichier”),   
la classe FenetreMere (Abstraction de “panneau principal”),   
la classe PanelFils (Abstraction de “panneau secondaire”),  
la classe InterfaceSaisie (Abstraction de “interface de saisie”),  
la classe InterfaceAffichage (Abstraction de “interface de programme”),  
la classe InterfaceReservation (Abstraction de “interface de réservation”),  
la classe InterfaceReservationFestival (Abstraction de “interface de réservation festival”).

Analyse détaillée :

En nous focalisant sur un aspect très proche du code :

* Festival comportera un nom, un tableau de Date, un tableau de places disponibles par date, un genre, un prix, un lieu, un chemin pour l’image.
* BoutonFestival servira à personnaliser un JButton dans le JPanel interfaceRéservation pour stocker un festival et le récupérer facilement lors de la création de l’interfaceReservationFestival.
* FestivalHelper servira à aider la classe Festival. Elle comportera des méthodes « statics » qui renverra une liste de festival d’un genre donné.
* Programme comportera une HashMap. Elle stockera les festivals, avec comme clé, le “genre:String” du festival, et en valeur une liste d’objet “festival:Festival”. Nous allons utiliser cette structure de données car les festivals sont principalement différenciés par leurs genres.
* Date comportera un jour, un mois et une année. Nous allons utiliser notre propre classe Date pour avoir une implémentation personnalisée.
* ExceptionDate servira à verifier si une date est valide et renverra une chaîne de caractères, si ce n’est pas le cas.
* Fichier servira à stocker l’objet Programme dans un fichier pour garder de manière « static » les données. Par conséquent, à chaque ouverture ou écriture dans le Programme les modifications seront sauvegardées dans le Fichier.
* Fenêtre mère servira de main à l’application : elle héritera de JFrame, instanciera un panel fils et configurera la taille ainsi que la position de la fenêtre. Elle instanciera également le PanelFils qui est un contentPane.
* PanelFils héritera de JPanel et instanciera un objet JMenu et un CardLayout dans lequel on ajoutera l’interface de saisie, de réservation et d’affichage, respectivement JPanel, DefaultTableModel, JPanel.
* InterfaceSaisie comportera un formulaire pour créer des festivals et les enregistrer dans le programme et dans le fichier. Les éléments graphiques seront placés avec un GridBagConstraint.
* InterfaceAffichage sera un modèle de table qui héritera de DefaultTableModel. Elle récupérera les festivals dans le programme et les organisera dans des colonnes par rapport à leur genre (leur clé) et affichera le nom, prix, lieu et genre dans les cellules.
* InterfaceRéservation sera un modèle de table qui héritera de DefaultTableModel. Elle récupérera les festivals dans le programme et affichera le nom. Un bouton (BoutonFestival) stockera le nom du festival. Lorsque que l’on clique dessus, un objet de la classe interfaceReservationFestival s’ouvrira et récupérera le nom du festival et les places disponibles.
* InterfaceRéservationFestival héritera de JPanel et s’ouvrira dans un JDialog, qui permettra de choisir la date souhaitée pour effectuer la réservation pour un festival.

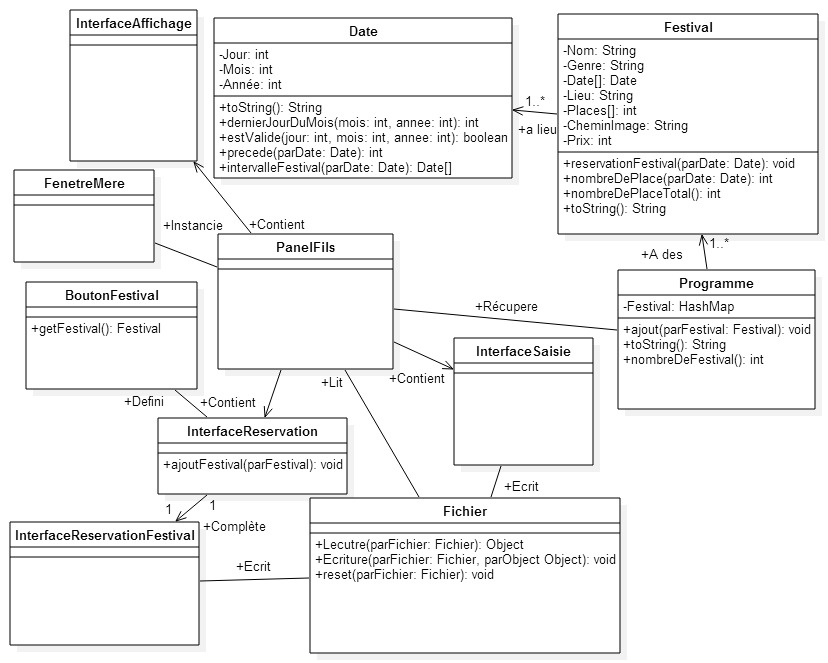


Figure 4 : Diagramme de classe de l’application gestionnaire de Festivals

Nous avons créé ci-dessous le diagramme d’objets correspondant à l’instanciation du diagramme de classe. Ce diagramme nous permet de mettre en évidence la structure MVC de ce système.  
On a d’abord sur la partie correspondante au rectangle rouge, avec Date, Festival, Programme et Fichier, la partie Modèle.  
On a ensuite sur la partie correspondante au rectangle vert les Interfaces, soit la Vue et le contrôleur.   
Et enfin la partie Vue uniquement, avec PanelFils et FenetreMere, correspondant au rectangle bleu.  
Ce diagramme nous permettra, à la suite de l'implémentation, la possibilité de faire du débugging en comparant les diagrammes d’objets.

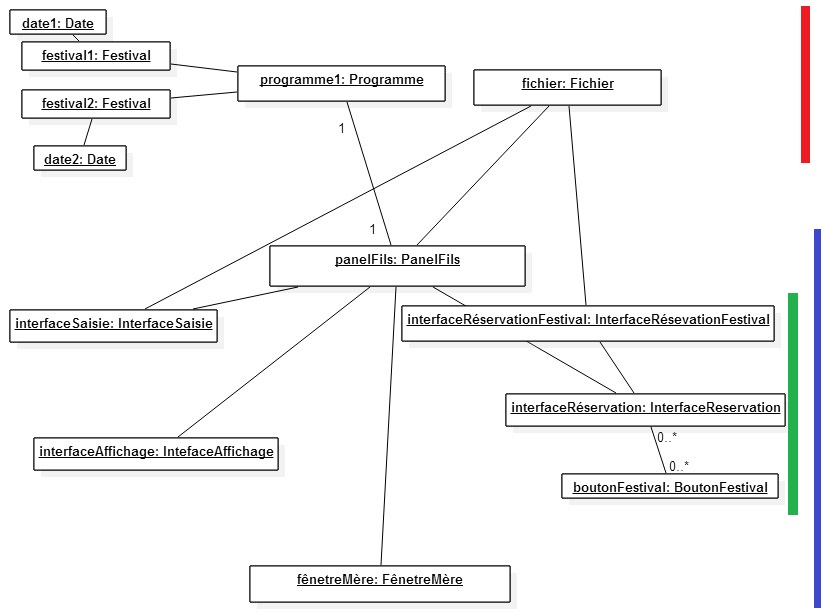


Figure 5: Diagramme objet de l’application gestionnaire de Festivals

L’élaboration du diagramme de séquence nous a permis de déterminer les échanges de messages entre les classes lors du scénario : “Un utilisateur lance le programme, et crée un festival”.  
Nous avons pu mettre en relief les utilisations de chaque classe afin de dégager le fonctionnement au niveau du code de l’interface affichage.

(Figure 6 en annexe)

**Précision de développement :**

Après analyse des diagrammes, nous avons déterminé les classes, et nous avons décidé de restreindre les valeurs des mois à Juillet et Août, au vu du sujet. Cependant, grâce à la conception orientée objet de notre programme, il serait possible d’implémenter les mois de Juin et Septembre.

Nous avons décidé de proposer plusieurs genres que nous avons ajoutés à Festival sous forme « static » et « final ». Les genres sont les suivants : Rock, Pop, Jazz, Electonique, Blues, Classique, Metal, Reggae.

Pour un festival, nous possédons un constructeur qui, par la saisie en paramètre de deux dates, produit un tableau d’intervalle de date, auquel est associé un tableau de place réservable. La fonction toString() s’adapte selon les jours de début et de fin du festival.

Pour la saisie d’un mois : si le mois d’Août est sélectionné en premier, le second ne pourra qu’être Août. Cela supprime des cas d’intervalles invalides. La gestion d’erreur pour des dates inversées est également prise en compte.

IV – Tests

**I) Test de création d’intervalle de Date**

1. **Conception**

Comme nous le précise le sujet, nous allons effectuer des tests boîtes noires pour la création d’un intervalle de date qui seront placés dans un tableau.  
Nous allons lister les cas possibles, et vérifier la création d’un tableau de date correspondant aux entrées. Selon les sorties, nous verrons si les implémentations de création de tableau sont ou non correctes.

1. **Préparation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Entrée 1 | Entrée 2 | Attendu | Résultat |
| 1 | 1 Juillet | 1 Juillet | Date[] | True |
| 2 | 2 Juillet | 1 Juillet | null | True |
| 3 | 31 Juillet | 1 Août | Date[] | True |
| 4 | 1 Août | 31 Juillet | null | True |
| 5 | 1 Août | 2 Août | Date[] | True |
| 6 | 2 Août | 1 Août | null | True |

1. **Analyse**

Les cas où les tests renvoient null dans le tableau sont gérés par l’interface de Saisie afin de restreindre les entrées, et donc ne pas rencontrer cette erreur. Elle n’a donc pas besoin d’être gérée par une exception. Dans le cas où on a null, le festival ne sera pas créé.

**II) Test de toStringReservation() de Festival**

1. **Conception**

Nous allons effectuer des tests boîtes noires pour le rendu en String d’un festival qui change selon les dates.  
Nous allons lister les cas possibles, et vérifier le rendu d’une String correcte correspondant aux entrées. Selon les sorties, nous verrons si les implémentations de création de tableau sont ou non correctes.

1. **Préparation**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Entrée 1 | Attendu | Résultat |
| 1 | tab[1Juillet-1Juillet] | toString() | True |
| 2 | tab[1Aout-1Aout] | toString() | True |
| 3 | tab[12Juillet[..]15Aout]\* | toString() | True |
| 4 | tab[1Juillet[..]19Juillet] | toString() | True |
| 5 | tab[2Aout[..]15Aout] | toString() | True |
| 6 | tab[16Aout[..]12Juillet] | null | False(nullPointerException) |
| 7 | tab[14Aout[..]9Aout] | null | False(nullPointerException) |
| 8 | tab[15Juillet[..]9Juillet] | null | False(nullPointerException) |

**\*** = Tableau des dates du 12 Juillet au 15 Aout

1. **Analyse**

Les tests qui retournent “nullPointerException” ne sont pas des cas présents dans notre programme, car la création d’un intervalle invalide de date pour un festival n’est pas possible (suite au test précédent). Cependant pour nos tests boîte noire, ces cas sont présents.

**III) Test nombre de place total**

1. **Conception**

Nous allons effectuer des tests boîtes noires pour le nombre total de places par festival.  
Nous allons lister les cas possibles, et vérifier l’addition des places selon le nombre de dates et de places par date.   
Nous allons donner, en entrée, des festivals avec des nombres de dates et places disponibles différents.  
Selon les sorties, nous verrons si les implémentations calcul de nombre de places total est correct ou non.

1. **Préparation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Entrée 1 | Entrée 2 | Attendu | Résultat |
| 1 | Festival avec 2 dates | Nombre de place disponible : 10 | 20 | True |
| 2 | Festival avec 2 dates | Nombre de place disponible : 0 | 0 | True |
| 3 | Festival avec 1 date | Nombre de place disponible : 10 | 10 | True |
| 4 | Festival avec 1 date | Nombre de place disponible : 0 | 0 | True |

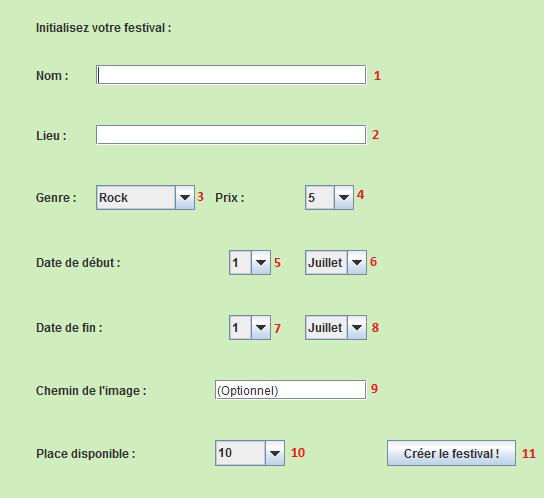
1. **Analyse**

Les tests montrent que les cas de tests se déroulent sans problème et que l’implémentation ne doit pas être modifiée.

V – Manuel Utilisateur

Nous allons ici vous détailler les principales fonctionnalités afin de faciliter l’utilisation de notre programme.

Le menu se caractérise par trois « boutons » : l’Initialisation (raccourcit Ctrl+I), le Programme (raccourcit Ctrl+P), et la Réservation (raccourcit Ctrl+R).



L’interface de saisie (Initialisation) permet de créer un festival.

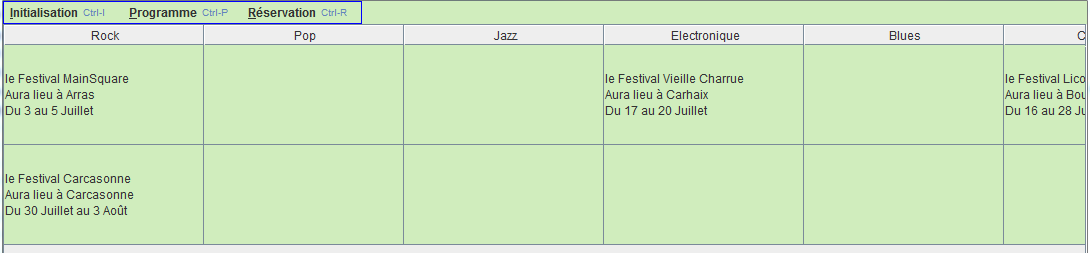
Les champs 1 et 2 sont obligatoires : en cas de non complétion, un message d’erreur apparaîtra.

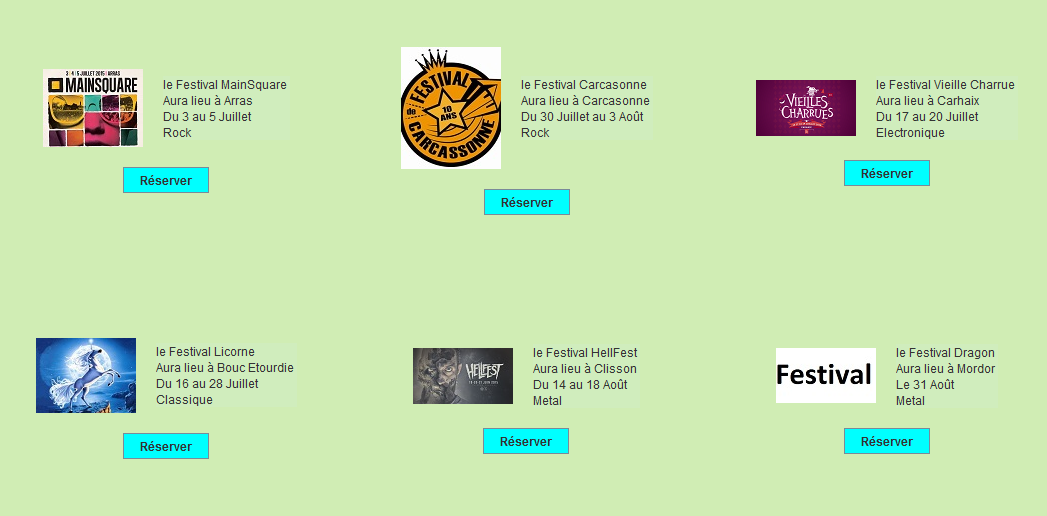
Vous pouvez préciser votre festival à l’aide des comboBox 3,4,5,6,7,8 et 10.

-5 et 6 pour respectivement le jour et le mois de la date de début.

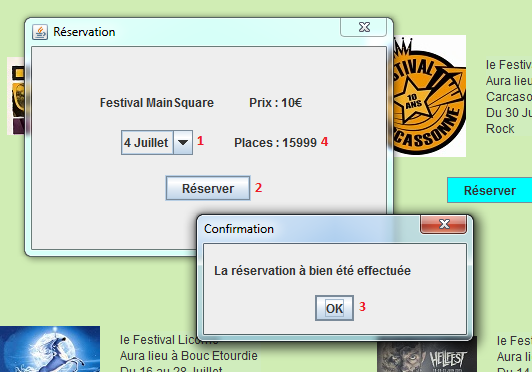
-7 et 8 pour respectivement le jour et le mois de la date de fin.

La création de votre festival se fera grâce au bouton 11.

La création d’un intervalle de date impossible fera apparaître une erreur. La saisie d’Août comme mois de début forcera la saisie du mois de fin à Août.

L’interface Programme permet la visualisation de vos festivals référencés par genre classé sous forme de tableau.

L’interface Réservation permet la visualisation des programmes grâce à leur image et leur brève description.  
Le bouton couleur cyan “Réserver” vous permet d’ouvrir un pop-up pour pouvoir réserver une place pour votre festival.



Le pop-up de réservation se présente sous la forme ci-dessus.  
-En 1, précisez la date de votre réservation puis réservez.   
-En 2, fermez la fenêtre de confirmation.   
-En 3, visualisez les places restantes pour la date sélectionnée en 4.



Si un festival n’a plus de date disposant de places libres, lorsque vous tenterez d’appuyer sur le bouton couleur cyan, un pop-up d’erreur s’affichera.

Nous vous remercions pour votre confiance et l’utilisation de notre logiciel.

Benjamin et Loïc

VI – Conclusion

En conclusion, la conception générale et détaillée de ce projet nous a mobilisé environ 15 heures. L’implémentation et la mise en place des tests a, quant à elle, nécessité 25 heures. La rédaction du rapport s’est faite tout au long du projet, du début de la conception, jusqu’à la fin des tests.

L’utilisation de git nous a amené à utiliser une charte de développement afin d’éviter toute sorte de conflits liés à des manières différentes d’implémenter. Nous avons dû commiter à chaque fois qu’une nouvelle modification majeure a été effectuée dans une branche expérimentale. Lorsque cette branche a été finalisée, nous avons mergé dans la branche master. La branche master regroupe des codes qui sont uniquement fonctionnels.

Ce projet nous a permis de mettre en pratique les méthodes de conception et de programmation vues en cours et en travaux pratiques. L’utilisation de git, qui nous a été fortement conseillée, nous a été bénéfique.

Nous avons appris à avoir une bonne gestion du temps et de la répartition du travail, afin de pouvoir rendre un programme et un compte rendu complet pour la dead-line.

VII – Annexe

1. Charte de développement

**CHARTE DE DEVELOPPEMENT :**

* Les fonctions seront implémentées sous la forme suivante :

**mafonction(){**

**impl;**

**impl;**

**} //mafonction()**

* Une tabullation de 3 espaces pour l’indentation
* Les condition (if else), les boucles (while, for) et autres seront implémentées sous la forme :

**if(x==y)**

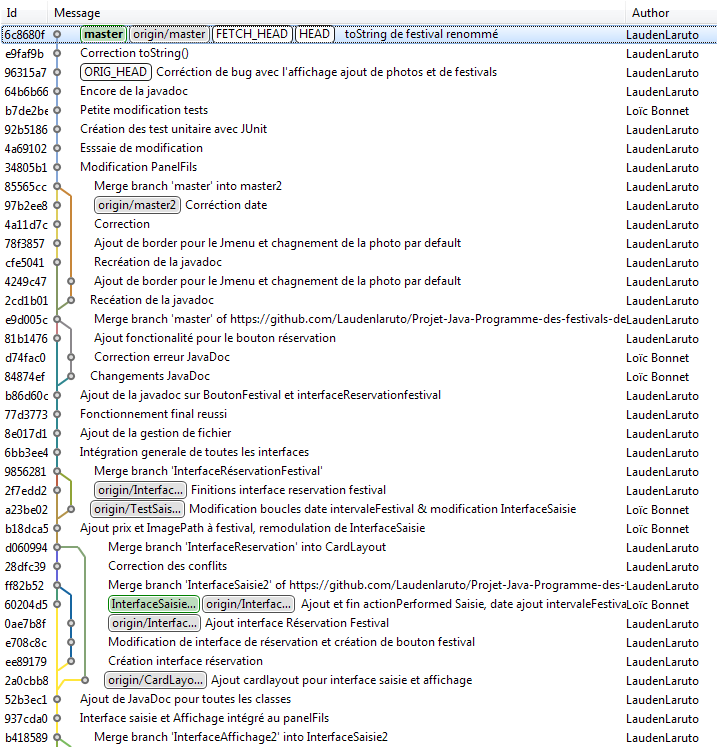
**{**

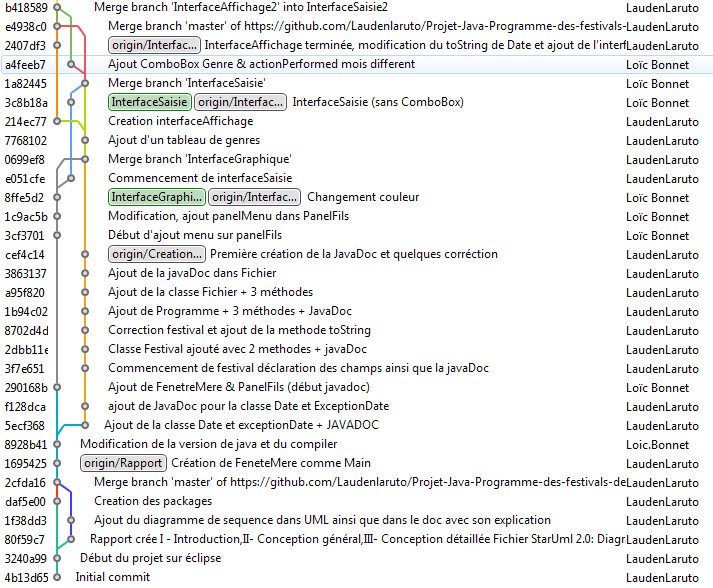
**impl;**

**impl;**

**}**

* Les méthodes get(par) et set seront implémentées systématiquement pour les classes du modèle.
* Chaque classe ou méthode ajoutée devra être décrite précisément selon ses entrées et sorties dans la Javadoc afin de générer une Javadoc complète et détaillée.

2) Arbre git



3) Sources

Image Festival 1 :<http://monster-high-love.centerblog.net/50-licorne>

Image Festival 2 :<http://www.rtl2.fr/actu/regions/des-ligeriens-aux-vieilles-charrues-cette-annee-7778195119>

Image Festival 3 :<http://maze.fr/musique/05/2015/hellfest-2015-les-10-ans-ca-va-etre-denfer/>

Image Festival 4 :<http://france3-regions.francetvinfo.fr/nord-pas-de-calais/2014/11/25/main-square-festival-toutes-les-infos-les-indiscretions-autour-de-l-edition-2015-599836.html>

Image Festival 5 :<http://www.carcassonne.fr/carcassonneville.nsf/vueTitre/docVivreFestival?opendocument&FR&Vivre&Culture%20/%20Animations&40&Les%20temps%20forts>

Image Festival 6 :<http://www.summersound.fi/fi/2-uncategorised>

Image Festival 7 :<http://www.casbah-records.com/radio-webzine/chroniques/une-nuit-%C3%A0-la-route-du-rock-2014>

Image Festival 8 :<http://www.infoconcert.com/festival/reggae-sun-ska-1850/news.html>

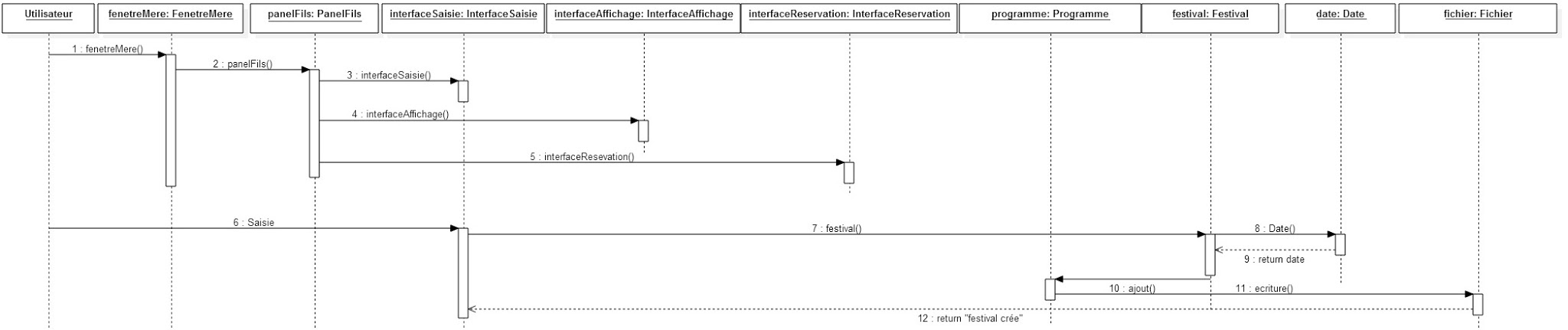


Figure 6 : Diagramme de séquence du scénario : “Un utilisateur saisit un festival”