

- Programmation Orientée Objet -

Héritage et Persistance des données

Reservation Manager

TD3

3^e année ESIEA - INF3034

A. Gademer

L. Beaudoin

L. Avanthey

2016 - 2017

Avant propos

*Dans ce TP nous allons mettre en œuvre deux concepts : la notion d'**héritage** que nous avons jusqu'ici abordé de manière théorique dans le cours ainsi que la notion de **persistance des données**. En effet, jusqu'ici, les choix utilisateurs réalisés au cours de l'exécution du programme sont perdus lorsque celui-ci se termine (imaginez de devoir reconfigurer vos préférences système à chaque démarrage !). Dans ce TD nous nous chercherons à rendre nos données importantes de nos programmes entre les exécutions. De manière générale, on sauvegarde ces données dans un système d'information (fichiers, base de données, etc.).*

Comme dans le TD précédent, nous commencerons par une application en mode Console, puis nous réaliserons une interface graphique. Nous allons mettre en œuvre différentes nouvelles notions :

- | | | |
|--------------------------|-----------------------------|-------------------------------|
| — Héritage de classe et | — Lecture/Écriture dans des | — Gestion d'événements souris |
| redéfinition de méthodes | fichiers | — Listes déroulantes |
| — Polymorphisme | — Sérialisation d'objets | — Fenêtres de messages |

Contexte

Nous allons nous placer dans le contexte d'un logiciel de réservation de place de théâtre. Une salle de théâtre est généralement composée d'un ensemble de sièges placés selon un plan précis allant de l'orchestre au paradis (les places les plus hautes et les plus lointaines). Certains emplacements sont occupés par des piliers ou du matériel dédiés aux ingénieurs sons et lumières. Bien choisir sa place est donc important ! Évidemment, il est essentiel de ne pas perdre l'information des réservations entre deux utilisations du programme et c'est pourquoi nous allons utiliser deux moyens de persistance à l'aide de fichiers : la sauvegarde de données dans un format de sauvegarde (ici CSV) et la sérialisation (encodage direct des objets).



Pochacco20
CC BY-NC-SA



Nous allons fonctionner par étape et rajouter des fonctionnalités au fur et à mesure.

Etape 0 : Classe principale : ReservationManagerConsole

EXERCICE 1



Ouvrez un éditeur de texte (gedit ou vim).

Déclarez une nouvelle classe publique

public class ReservationManagerConsole dans le fichier ReservationManagerConsole.java qui contient un constructeur (en s'inspirant du SpaceTravel), qui crée une boucle de question. Pour le moment les actions possibles sont d'afficher l'aide ("h") ou de quitter ("q").

La méthode **main** se contentera de créer un nouvel objet ReservationManagerConsole.

Compilation Pour compiler, on utilise la commande (dans le Terminal) :



javac ReservationManagerConsole.java

Pour exécuter, on utilise la commande :

java ReservationManagerConsole.

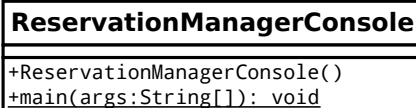
Vous devriez voir :

```
Welcome to the Reservation Manager
What do you want to do (h for help)
h
h: Print this help
q: Quit
What do you want to do (h for help)
q
Bye Bye
```



Classe principale Cette classe, en interface console, vous servira à tester votre travail au fur et à mesure du TP.

Voici son diagramme UML :



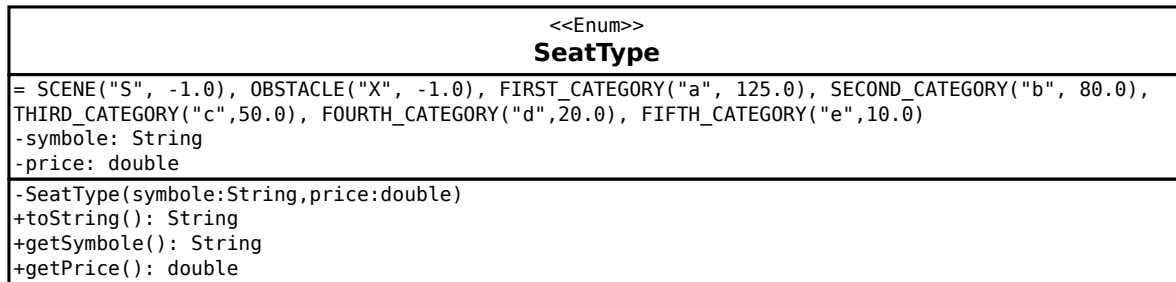
Etape 1 : Plan de salle

Nous allons gérer votre plan de salle sous la forme d'un tableau bidimensionnel de places (en faisant le choix raisonnable que la scène et les obstacles sont considérés comme des places qui seront inaccessibles à la réservation). En fonction de la distance à la scène et de la visibilité, les places sont réparties en 5 catégories de prix décroissantes (les premiers sièges pouvant atteindre des prix mirobolants).

EXERCICE 2



Créez l'énumération **public enum** SeatType dans un fichier SeatType.java, en suivant l'UML ci-dessous. *Remarque* : La méthode toString se contentera de retourner le symbole associé.

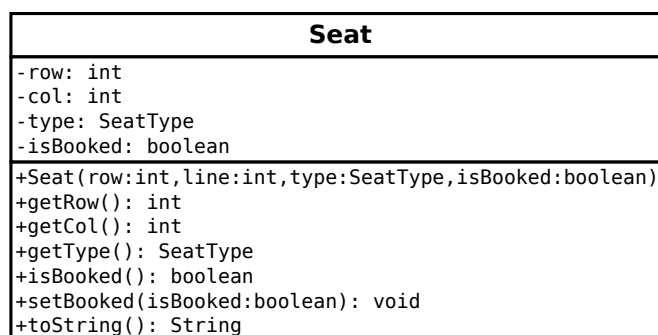


Les places (seat en anglais) sont au cœur de notre programme.

EXERCICE 3



Créez la classe **public class** Seat dans un fichier Seat.java en suivant l'UML ci-dessous.



Affichage des numéro de sièges

Bien que nous stockions le numéro de ligne du siège sous forme de nombre, on utilise généralement un système Lettre+chiffre dans les théâtres (par ex : D4 pour rang D, place n°4). On écrira donc une méthode `toString` en conséquence :

```
/*
 * On utilise l'arithmétique des caractères ('A'+1 => 'B')
 * pour convertir les nombres 0, 1, 2, ... en caractères A, B, C, ...
 */
public String toString() {
    char rowLetter = (char) ('A' + row);
    return ""+rowLetter+col;
}
```

EXERCICE 4



Créez la classe **public class** Theater dans un fichier Theater.java qui contient le tableau bidimensionnel de sièges.

- Le constructeur **public** Theater(int nbRow, int nbCol) prends deux paramètres nbRow et nbCol correspondant aux dimensions du nouveau tableau et le remplit (avec une double boucle) de sièges non réservés de première catégorie,
- les méthodes **public int** getNbRow() et **public int** getNbCol() retournent les dimensions du tableau.



Dimension d'un tableau 2D Rappelez-vous qu'en Java un tableau connaît sa taille ! Et aussi qu'en Java un tableau 2D est un tableau de tableaux 1D.

Exemple :

```
int[][] tab = {{1,2,3},{4,5,6}};
System.out.println("NbRow : "+tab.length); // Size of the main array
System.out.println("NbCol : "+tab[0].length); //Size of the sub-array
```

- la méthode **public** String toString() renvoie une chaîne de caractères représentant la disposition des sièges du théâtre, ainsi que la numérotation des lignes (en lettre) et des colonnes (en chiffre). *Remarque : on distinguera les sièges réservés des sièges libres en les écrivant leur symbole en MAJUSCULE* (méthode toUpperCase() de la classe String). On trouve un exemple d'affichage de cette méthode ci-dessous.

Exemple d'affichage de la méthode toString (les places D2 et G4 sont réservées) :

```
A a a a a a a a
B a a a a a a a
C a a a a a a a
D a a A a a a a
E a a a a a a a
F a a a a a a a
G a a a A a a a
H a a a a a a a
0 1 2 3 4 5 6 7
```

- ✓ **Pour créer une grande chaîne de caractères** On utilise généralement la classe StringBuffer <http://docs.oracle.com/javase/7/docs/api/index.html?java/lang/StringBuffer.html> à laquelle on peut ajouter petit à petit des éléments avant de produire la chaîne de caractères finale. L'utilisation de StringBuffer permet de gagner en rapidité et en place mémoire !

Exemple :

```
StringBuffer buf = new StringBuffer();
buf.append("Puissances de 2 : ");
int pow=1;
for(int i = 0; i <= 10; i++) {
    buf.append("2^").append(i).append(" = ").append(pow);
    if(i!=10) {
        buf.append(", ");
    }
    pow*=2;
}
System.out.println(buf.toString());
```

```
Puissances de 2 : 2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 8, 2^4 = 16, 2^5 = 32, 2^6 = 64, 2^7 = 128,
2^8 = 256, 2^9 = 512, 2^10 = 1024
```

- ✓ **Afficher des lettres à partir de numéros** Rappelez vous que les caractères sont stockés sous la forme de chiffre. On peut donc leur appliquer des opérateurs mathématiques

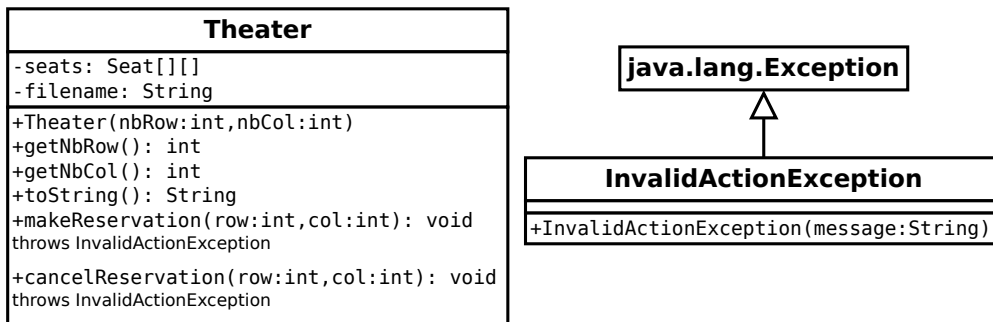
Exemple :

```
for(char c = 'A' ; c <= 'Z' ; c++) {
    System.out.print(c+" ");
}
System.out.println();
for(int i = 0 ; i <= 26 ; i++) {
    System.out.print((char)('A'+i)+" "); // Need to make an explicite cast
}
System.out.println();
```

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

- la méthode **public void** makeReservation(int row, int col) throws InvalidActionException cherche à marquer le siège de la case d'indice (row, col) comme réservé (setBooked) si le siège était déjà réservé elle renvoie une exception de type InvalidActionException.

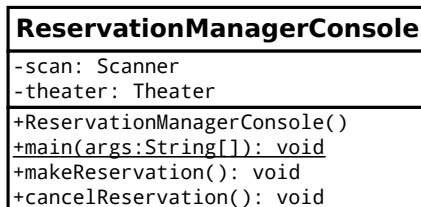
- la méthode `public void cancelReservation(int row, int col) throws InvalidActionException` fait l'action inverse.



EXERCICE 5



Modifiez la classe `ReservationManagerConsole` pour ajouter deux nouvelles actions : afficher la disposition du théâtre "st", faire une réservation "mr" (utilisez une fonction par soucis de lisibilité.) et annuler une réservation "cr".



Exemple d'affichage :

```

Welcome to the Reservation Manager
What do you want to do (h for help)
h
h: Print this help
st: Show Theater
mr: Make a Reservation
cr: Cancel a Reservation
q: Quit
What do you want to do (h for help)
mr
A a a a a a a a
B a a a a a a a
C a a a a a a a
D a a a a a a a
E a a a a a a a
F a a a a a a a
G a a a a a a a
H a a a a a a a
0 1 2 3 4 5 6 7

Please enter row letter:
F
Please enter line number:
4
A a a a a a a a
B a a a a a a a
C a a a a a a a
D a a a a a a a
E a a a a a a a
F a a a a a a a
G a a a a a a a
H a a a a a a a
0 1 2 3 4 5 6 7
    
```

```
What do you want to do (h for help)
mr
A a a a a a a a
B a a a a a a a
C a a a a a a a
D a a a a a a a
E a a a a a a a
F a a a a A a a a
G a a a a a a a
H a a a a a a a
  0 1 2 3 4 5 6 7

Please enter row letter:
F
Please enter line number:
4
/!\This space is not valid for reservation! /\
What do you want to do (h for help)
cr
A a a a a a a a
B a a a a a a a
C a a a a a a a
D a a a a a a a
E a a a a a a a
F a a a a A a a a
G a a a a a a a
H a a a a a a a
  0 1 2 3 4 5 6 7

Please enter row letter:
F
Please enter line number:
4
A a a a a a a a
B a a a a a a a
C a a a a a a a
D a a a a a a a
E a a a a a a a
F a a a a a a a
G a a a a a a a
H a a a a a a a
  0 1 2 3 4 5 6 7

What do you want to do (h for help)
cr
A a a a a a a a
B a a a a a a a
C a a a a a a a
D a a a a a a a
E a a a a a a a
F a a a a a a a
G a a a a a a a
H a a a a a a a
  0 1 2 3 4 5 6 7

Please enter row letter:
F
Please enter line number:
4
/!\This space is not valid for cancelation! /\
What do you want to do (h for help)
q
Bye Bye
```

Chargement à partir d'un fichier Nous aimerions pouvoir charger la disposition du théâtre à partir d'un fichier au format CSV (valeurs séparées par des points virgules, modifiable par des éditeurs de textes ou par des tableurs).

```
8;7;;;;;
X;X;X;X;X;X;X
X;e;e;e;e;e;X
X;d;e;c;e;d;X
X;c;X;c;X;c;X
X;c;b;b;b;c;X
X;b;a;a;a;b;X
X;b;a;a;a;b;X
X;X;S;S;S;X;X
```

- La première ligne contient le nombre de lignes, suivi du nombre de colonnes.
- Chaque ligne suivante contient les symboles des places correspondantes ("X" : obstacle, "S" : scène, "a", "b", "c", "d", "e", : 1^{re}, 2^e, 3^e, 4^e, 5^e catégorie disponible, "A", "B", "C", "D", "E", : 1^{re}, 2^e, 3^e, 4^e, 5^e catégorie déjà réservée).



Pour interpréter ce fichier

il nous faut tout d'abord être capable de faire le lien entre les symboles du fichier et les types de sièges existant.

EXERCICE 6



Rajoutez la méthode **public static** `SeatType getSeatTypeFromSymbole(String symbole)` dans l'énumération `SeatType` qui renvoie l'instance associée au symbole passée en paramètre (**en ignorant la casse c'est-à-dire si le symbole est en majuscule ou minuscule**) ou **null** si le symbole ne correspond à aucune instance.

<<Enum>> SeatType	
= SCENE("S", -1.0), OBSTACLE("X", -1.0), FIRST_CATEGORY("a", 125.0), SECOND_CATEGORY("b", 80.0), THIRD_CATEGORY("c", 50.0), FOURTH_CATEGORY("d", 20.0), FIFTH_CATEGORY("e", 10.0)	
-symbole: String	
-price: double	
-SeatType(symbole:String,price:double)	
+toString(): String	
+getSymbole(): String	
+getPrice(): double	
+getSeatTypeFromSymbole(symbole:String): SeatType	



Parcourir les instances d'une énumération La méthode de classe `SeatType.values()` renvoie la collection d'instance de l'énumération, ce qui permet de parcourir facilement toutes les instances d'une énumération.

Exemple :

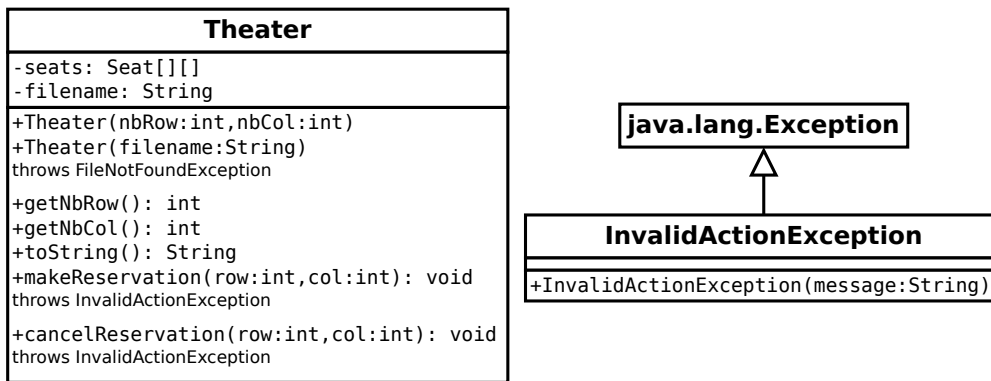
```
for(SeatType s:SeatType.values()) {
    System.out.print(s+" ");
}
```

S X a b c d e

EXERCICE 7



Rajoutez le constructeur **public** `Theater(String filename)` qui prend en paramètre le chemin vers un fichier CSV respectant le format ci-dessus et initialise le tableau `seats` à partir du contenu du fichier grâce à un objet `Scanner`.



- ✓ **Délimiteur point virgule** On peut spécifier à Scanner quels sont les caractères qui marquent la séparation entre les mots. Par exemple, dans notre cas, nous pouvons spécifier le caractère ; comme étant un délimiteur entre les mots.

Si l'on ne précise rien, les séparateur sont les caractères "blancs" (espace, tabulation, retour à la ligne).

```

Scanner scan = new Scanner("mouton;vache coq;zebre");
System.out.println("Délimiter : "+scan.delimiter());
while(scan.hasNext()) {
    System.out.println(scan.next());
}

```

"mouton;vache" est considéré comme un mot unique.

```

Delimiter : \p{javaWhitespace}+
mouton;vache
coq;zebre

```

Si on ajoute le nouveau délimiteur (attention à la syntaxe !) :

```

Scanner scan = new Scanner("mouton;vache coq;zebre");
scan.useDelimiter(scan.delimiter()+"|;"); // Add ; as a valid delimiter
System.out.println("Délimiter : "+scan.delimiter());
while(scan.hasNext()) {
    System.out.println(scan.next());
}

```

Chaque mot est, ici, pris séparément (notez que les caractères "blancs" restent aussi des délimiteurs) :

```

Delimiter : \p{javaWhitespace}+|;+
mouton
vache
coq
zebre

```

- ✓ **Passer les cases vides** Attention ! Un certain nombre de "cases" sont vides après le nombre de ligne et de colonne. La méthode `getSeatTypeFromSymbole` peut donc renvoyer **null** à certains moment. Vous prendrez soin de traiter ces cas là avec une boucle "Faire...Tant que(la fonction renvoie **null**)".

- ! **Sièges réservés ?** Pour savoir si un siège est réservé on regarde si le premier caractère du symbole est en MAJUSCULE. On utilise pour cela la méthode statique `isUpperCase()` de la classe `Character`

```

String symbole = scan.next();
seats[row][col] = new Seat(row, col, SeatType.getSeatTypeFromSymbole(symbole),
    Character.isUpperCase(symbole.charAt(0)));

```

Exemple d'affichage de votre programme (utilisant le fichier CSV décrit ci-avant) :

```

Welcome to the Reservation Manager
File theater1.csv loaded with success.
What do you want to do (h for help)
h

```



```

h: Print this help
st: Show Theater
mr: Make a Reservation
cr: Cancel a Reservation
q: Quit
What do you want to do (h for help)
st
A X X X X X X X
B X e e e e e X
C X d e c e d X
D X c X c X c X
E X c b b b c X
F X b a a a b X
G X b a a a b X
H X X S S S X X
  0 1 2 3 4 5 6

What do you want to do (h for help)
mr
A X X X X X X X
B X e e e e e X
C X d e c e d X
D X c X c X c X
E X c b b b c X
F X b a a a b X
G X b a a a b X
H X X S S S X X
  0 1 2 3 4 5 6

Please enter row letter:
F
Please enter line number:
4
A X X X X X X X
B X e e e e e X
C X d e c e d X
D X c X c X c X
E X c b b b c X
F X b a a A b X
G X b a a a b X
H X X S S S X X
  0 1 2 3 4 5 6

What do you want to do (h for help)
q
Bye Bye

```

Etape 2 : Persistance - Sauvegarde des données au format CSV

Notre principal problème vient du fait que toutes nos réservations disparaissent dès que l'on quitte le programme ! Il nous faut donc rajouter une méthode save pour sauvegarder l'état de notre tableau dans un fichier.

Comme nous avons déjà défini le format de fichier CSV pour charger notre tableau, il semble raisonnable d'utiliser le même format pour notre sauvegarde. Pas question pour autant d'écraser notre fichier de départ ! Nous devons pouvoir revenir à la situation originelle lorsque nous le désirons.

EXERCICE 8



Rajoutez à la classe Theater la méthode **public void** save() qui écrit dans un fichier d'extension **".bak"** (theatre1.csv.bak si le fichier de départ était theatre1.csv ou theatre.bak si filename est **null** car on a utilisé le premier constructeur). La fonction écrira le nombre de lignes et le nombre de colonnes, séparés par un point virgule. Puis, elle écrira sur les lignes suivantes le contenu du tableau bidimensionnel sous forme de symboles séparés par des points-virgules.

Exemple de fichier .bak

```
8;7
X;X;X;X;X;X;X
X;e;e;e;e;e;X
X;d;e;c;e;d;X
X;c;X;c;X;c;X
X;c;b;b;b;c;X
X;b;a;a;A;b;X
X;b;a;a;a;b;X
X;X;S;S;S;X;X
```



Sièges réservés en MAJUSCULE Rappelez-vous d'écrire les symboles des sièges réservés en MAJUSCULE dans le fichier pour conserver cette information.



Écrire dans un fichier Pour écrire dans un fichier on utilise la classe `FileWriter`

<http://docs.oracle.com/javase/7/docs/api/index.html?java/io/FileWriter.html>

Exemple :

```
FileWriter fw = new FileWriter("myFile.txt");
try {
    fw.write("MyText");
} catch (IOException ex) {
    System.err.println(ex.getMessage());
    System.exit(-1);
}
fw.close();
```

EXERCICE 9



Modifiez vos constructeur pour charger les fichiers **".bak"** en priorité **si ils existent**. En cas d'absence de fichier **".bak"** le constructeur garde son mode de fonctionnement précédent (lecture dans un CSV ou initialisation "à la main").

Première exécution :

```
Welcome to the Reservation Manager
What do you want to do (h for help)
mr
A X X X X X X
B X e e e e e X
C X d e c e d X
D X c X c X c X
E X c b b b c X
F X b a a a b X
G X b a a a b X
H X X S S S X X
  0 1 2 3 4 5 6

Please enter row letter:
F
Please enter line number:
4
```

```
A X X X X X X
B X e e e e X
C X d e c e d X
D X c X c X c X
E X c b b b c X
F X b a a A b X
G X b a a a b X
H X X S S S X X
  0 1 2 3 4 5 6
```

```
What do you want to do (h for help)
q
Bye Bye
```

Seconde exécution (Notez comme le programme à retenu notre précédente action 😊!) :

```
Welcome to the Reservation Manager
What do you want to do (h for help)
st
A X X X X X X
B X e e e e X
C X d e c e d X
D X c X c X c X
E X c b b b c X
F X b a a A b X
G X b a a a b X
H X X S S S X X
  0 1 2 3 4 5 6

What do you want to do (h for help)
q
Bye Bye
```

Etape 3 : Création du fichier client

Nous aimerions mémoriser quelles sont les personnes qui ont réservé les places. Considérant qu'une même personne doit pouvoir réserver plusieurs places, mais que nous ne voulons stocker ses informations qu'une seule fois, il va donc nous falloir créer un fichier client.

La classe Client sert à identifier les clients par leur nom, prénom et adresse. Pour gérer les cas des homonymes un identifiant unique interne est associé à chaque nouveau client. Il est basé sur une valeur `currentId` lié à la classe Client (attribut de classe) qui est incrémenté à chaque appel au constructeur.

- la méthode **public** String toString() renvoie le prénom suivi du nom, alors que
- la méthode **public** String getFullString() renvoie le numéro client, le prénom, le nom et l'adresse du client.

Client
-currentID: int -id: int -lastname: String -firstname: String -address: String
+Client(lastname:String,firstname:String, address:String) +getId(): int +getLastname(): String +getFirstname(): String +getAddress(): String +toString(): String +getFullString(): String

ReservationManagerConsole
-scan: Scanner -theater: Theater -clients: LinkedList<Client>
+ReservationManagerConsole() +main(args:String[]): void +makeReservation(): void +cancelReservation(): void +addClient(): void +selectClient(): Client +removeClient(): void

EXERCICE 10



Créez la classe **public class** Client dans le fichier Client.java, en suivant l'UML donné.

Puis rajoutez dans votre classe ReservationManagerConsole une liste chaînée (`LinkedList<Client>`) pour stocker les clients ainsi que des nouvelles actions : lister les clients ("`lc`"), ajouter un nouveau client ("`ac`"), retirer un client ("`rc`").



Identifiant unique Afin de donner un identifiant unique à chaque client, nous utilisons une variable de classe `currentId` qui est incrémentée à chaque création d'une nouvelle instance (à chaque appel du constructeur). L'attribut `id` est initialisé avec la valeur courante de `currentId`.



Sélectionner un client Pour permettre de sélectionner un client, affichez la description longue de tous les clients existants puis demandez à l'utilisateur de choisir le numéro du client à sélectionner. Puis comparez cette valeur aux identifiants de chacun des clients de la liste et conservez la référence client si vous la retrouvez dans la liste. Pensez à gérer la possibilité d'une saisie incorrecte de la part de l'utilisateur.

Exemple :

```
for(Client c:clients) { // Print the potential selection
    System.out.println(c.getFullString());
}
System.out.println("Please enter the id of the client to be removed or -1 to cancel the action."
); // Ask for an id
int id;
Client selectedClient = null; // Make the hypothesis of erroneous id
try {
    id = scan.nextInt(); // Get the id from the user
    for(Client c:clients) { // Search for a matching id
        if(c.getId() == id) { // If found, stop searching
            selectedClient = c;
            break;
        }
    }
    if(selectedClient != null) { // If hypothesis was wrong, you have a valid selected Client
        /* Do something */
    }
    if(selectedClient == null && id != -1) {
        System.err.println("Invalid selection");
    }
} catch(RuntimeException ex) { // Catch potential error
    System.err.println("This is not a valid number !");
    scan.nextLine(); // Remove what provoke the error
}
```

```
Welcome to the Reservation Manager
What do you want to do (h for help)
lc
Client list : []
What do you want to do (h for help)
ac
Lastname : Rabbit
Firstname : Roger
Address : Toonville
Roger Rabbit was added with success.
What do you want to do (h for help)
lc
Client list : [Roger Rabbit]
What do you want to do (h for help)
rc
Client n°0 : Roger Rabbit (Toonville)
Please enter the id of the client to be removed or -1 to cancel the action.
Oops
```

```
This is not a valid number !
What do you want to do (h for help)
rc
Client n°0 : Roger Rabbit (Toonville)
Please enter the id of the client to be removed or -1 to cancel the action.
0
Roger Rabbit was removed with success.
What do you want to do (h for help)
lc
Client list : []
What do you want to do (h for help)
q
Bye Bye
```



Observons cependant que notre programme ne garde pas la mémoire du fichier client pour le moment.

Première exécution :

```
Welcome to the Reservation Manager
What do you want to do (h for help)
lc
Client list : []
What do you want to do (h for help)
ac
Lastname : Rabbit
Firstname : Roger
Address : Toonville
Roger Rabbit was added with success.
What do you want to do (h for help)
lc
Client list : [Roger Rabbit]
What do you want to do (h for help)
q
Bye Bye
```

Seconde exécution :

```
Welcome to the Reservation Manager
What do you want to do (h for help)
lc
Client list : []
What do you want to do (h for help)
q
Bye Bye
```

Etape 4 : Sérialisation, la sauvegarde d'objet en Java




Sérialisation : En informatique, la sérialisation est un processus visant à coder l'état d'une information qui est en mémoire (une structure de donnée, un objet, etc.) sous la forme condensée, par exemple une suite d'octets. Cette suite pourra par exemple être utilisée pour la sauvegarde (persistance) ou le transport sur le réseau (proxy). L'activité symétrique, visant à décoder cette suite pour créer une copie conforme de l'information d'origine, s'appelle la désérialisation.

Considérant que vous avez une liste chaînée d'objet, comment la stocker de manière simple dans un fichier ?

En se basant sur le travail effectué à l'étape 3, on pourrait stocker chaque objet sous forme de texte séparé par des points virgules. Mais que se passe-t-il si l'objet contient lui-même des listes chaînées ? Il devient plus complexe (mais pas impossible) de faire sa propre fonction de sauvegarde/chargement.

 **Java intègre naturellement des méthodes de sérialisation/désérialisation permettant de sauvegarder/charger des objets dans des fichiers.**

Pour cela, la classe que l'on désire sérialiser (et toutes les classes que l'on utilise au sein de cette classe) doivent simplement implémenter l'interface `Serializable`. Heureusement pour nous, c'est le cas de toutes les classes courantes : `String`, `LinkedList`, etc.

 **L'interface `Serializable` ne possède pas de méthode à redéfinir.** Elle sert uniquement à identifier les classes pouvant être sérialisées.

<http://docs.oracle.com/javase/7/docs/api/index.html?java/io/Serializable.html>

Considérons la classe `Serializer` qui définit deux méthodes de classes `saveToFile` et `loadToFile` permettant de sauver ou de charger n'importe quel objet Java (objet, tableau ou collection) dans un fichier.

EXERCICE 11



Rajoutez la classe `public class Serializer` à votre programme en copiant le code ci-dessous dans un fichier `Serializer.java`.

```
import java.io.*;

public final class Serializer {

    /**
     * Private constructor. Serializer has no need to be instanced
     */
    private Serializer() {}

    /**
     * Use an ObjectOutputStream to write a Serializable object into a file.
     *
     * It is a static method, so you need to call it from the class.
     * Ex : Serializer.saveToFile("car.bak", c);
     *
     * @param filename The name of the file where to save the seralized object
     * @param object the object to serialize
     * @throws Potential I/O exception
     * @see Serializable
     * @see ObjectOutputStream
     */
    public static void saveToFile(String filename, Serializable object) throws IOException { //
        Potential I/O exception : FileNotFoundException, InvalidClassException, etc.
        FileOutputStream file = new FileOutputStream(filename); // OutputStream to the file
        ObjectOutputStream oos = new ObjectOutputStream(file); // Serializer
        oos.writeObject(object); // Write the serializable object to the stream
        oos.flush(); // Flush the stream (i.e transmitting all the bytes accumulated in the
            buffer)
        oos.close(); // Close the stream and the file
    }

    /**
     * Use an ObjectInputStream to load a Serializable object from a file.
     *
     * Warning : this method is generic, you need to specify the type of the object
     * during the call.
     * Ex : Car c = Serializer.<Car>loadFromFile("car.bak");
     *
     * @param filename The name of the file where to save the seralized object
     * @return object the object deserialized.
     * @throws Potential I/O exception
     * @see Serializable
     * @see ObjectInputStream
     */
    @SuppressWarnings("unchecked") // Remove the warning provoked by the unchecked cast
```

```

    public static <T> T loadFromFile(String filename) throws ClassNotFoundException, IOException
    {
        FileInputStream file = new FileInputStream(filename); //InputStream, reading from the
        //file
        ObjectInputStream ois = new ObjectInputStream(file); // Deserializer
        return (T) ois.readObject(); //Read the serialized object and cast it to the wanted type
    }
}

```

Exemple d'utilisation de la classe Serializer

```

import java.io.*;
import java.util.*;

public class Main {

    public static void main(String[] args) {

        Scanner scan = new Scanner("mouton;vache coq;zebre");
        scan.useDelimiter(scan.delimiter()+"|;"); // Add ; as a valid delimiter
        System.out.println("Delimiter : "+scan.delimiter());
        while(scan.hasNext()) {
            System.out.println(scan.next());
        }

        String filename = "car.bak"; // The file where to save the data

        /* SAVING THE OBJECT */

        Car c = new Car("Toyota", 2013); // The object to save, class Car implements Serializable
        try {
            Serializer.saveToFile(filename, c); // Save the object with Serializer
        } catch(IOException ex) {
            ex.printStackTrace(); // In case of exception, print the error stack
            System.exit(-1); // then quit
        }

        /* LOADING THE OBJECT */

        Car c2;

        try {
            c2 = Serializer.<Car>loadFromFile("cars.bak"); // Load the object with Serializer
        } catch(ClassNotFoundException ex) { // This exception occurs when trying to load a
            //class that no longer exist
            ex.printStackTrace();
            System.exit(-1);
        } catch(IOException ex) {
            ex.printStackTrace();
            System.exit(-1);
        }
    }
}

```



L'objet à sauvegarder doit implémenter l'interface Serializable, sinon le compilateur ne vous laissera pas utiliser la méthode.

Main.java:14: error: method saveToFile in class Serializer cannot be applied to given types;

```

        Serializer.saveToFile(filename, c); // Save the object with Serializer
        ^
required: String,Serializable
found: String,Car
reason: actual argument Car cannot be converted to Serializable by method invocation
        conversion
1 error

```



Le fichier sérialisé est un fichier binaire il n'est donc pas lisible par un éditeur de texte. On retrouve cependant un certain nombre de référence en clair : le nom de la classe, des attributs (name, type, valeur, etc.).

Exemple de fichier car.bak

```

M-,M-m^@^EsR^@^T^C^Car^NM-^P^S{M-^@&6p^B^@^BI^@^N^productionYearL^@^E^brandt^@^RL^java/lang/String;xp^
^@^@GM-]t^@^F^Toyota

```

Exemple de fichier cars.bak, sauvegarde d'une liste chaînée de voiture (LinkedList<Car>).

```

M-,M-m^@^EsR^@^T^java.util.LinkedList^L)S]J`M-^H"^^C^@^@xpW^D^@^@^@Csr^@^C^Car^NM-^P^S{M-^@&6p^B^@
^BI^@^N^productionYearL^@^E^brandt^@^RL^java/lang/String;xp^@^@GM-]t^@^F^Toyotasq^@~^@^B^@^@GM
->t^@^E^Hondasq^@~^@^B^@^@GM-Wt^@^G^Peugeotx

```

EXERCICE 12



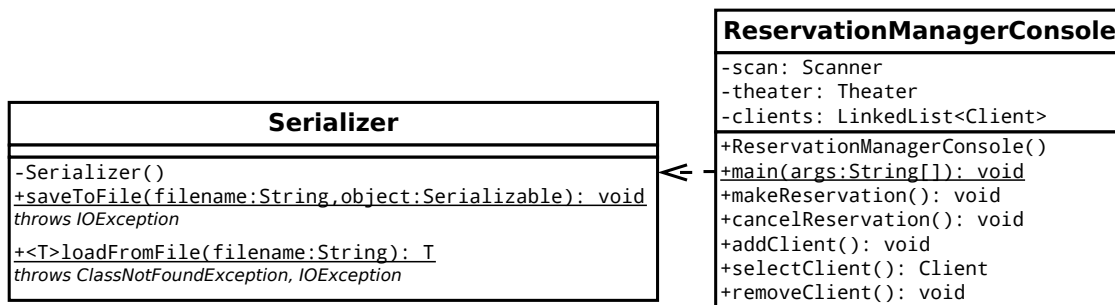
1. Rendez votre classe client sérialisable (c'est-à-dire, faites-la implémenter Serializable).
2. Puis utilisez la classe Serializer dans la classe ReservationManagerConsole pour sauvegarder l'état de la liste de client dans un fichier à chaque modification de la liste (ajout ou suppression de client).
3. Enfin, ajoutez le chargement de la liste de client depuis le fichier sauvegardé au début du constructeur de ReservationManagerConsole pour que l'information soit persistante entre les exécutions du programme.
4. **Attention**, si le fichier de sauvegarde n'existe pas encore (la méthode loadFromFile renvoie une FileNotFoundException par exemple) il faut créer une liste de clients vide (**new** LinkedList<Client>()).



Variable de classe Attention ! Si la sérialisation vous permet de recréer votre liste de client à l'identique, la variable de classe `currentId` n'est, elle pas réinitialisée. C'est pourquoi nous ajoutons une nouvelle méthode **public** `setCurrentId(int id)` qui fixe la valeur de la variable de classe après la désérialisation.

Client
-currentID: int -id: int -lastname: String -firstname: String -address: String
+Client(lastname:String,firstname:String, address:String) +getId(): int +getLastname(): String +getFirstname(): String +getAddress(): String +toString(): String +getFullString(): String +setCurrentId(currentId:int): void





Première exécution :

```

Welcome to the Reservation Manager
What do you want to do (h for help)
lc
Client list : []
What do you want to do (h for help)
ac
Lastname : Rabbit
Firstname : Roger
Address : Toonville
Roger Rabbit was added with success.
What do you want to do (h for help)
lc
Client list : [Roger Rabbit]
What do you want to do (h for help)
q
Bye Bye
    
```

Seconde exécution :

```

Welcome to the Reservation Manager
What do you want to do (h for help)
lc
Client list : [Roger Rabbit]
What do you want to do (h for help)
q
Bye Bye
    
```

Message d'erreur : "Client ; local class incompatible : stream classdesc serialVersionUID = XXX, local class serialVersionUID = YYY" Pas de panique ! Vous avez sauvegardé votre liste client, puis vous avez modifié votre classe `Client` et Java ne reconnaît plus rien ! C'est normal. Et il n'y a pas d'autre solution que de supprimer le fichier `client.bak` pour le recréer avec votre nouvelle définition de `Client`. C'est la contrepartie de la sérialisation automatisée.

Etape 5 : Informations croisées : où est ma place ?

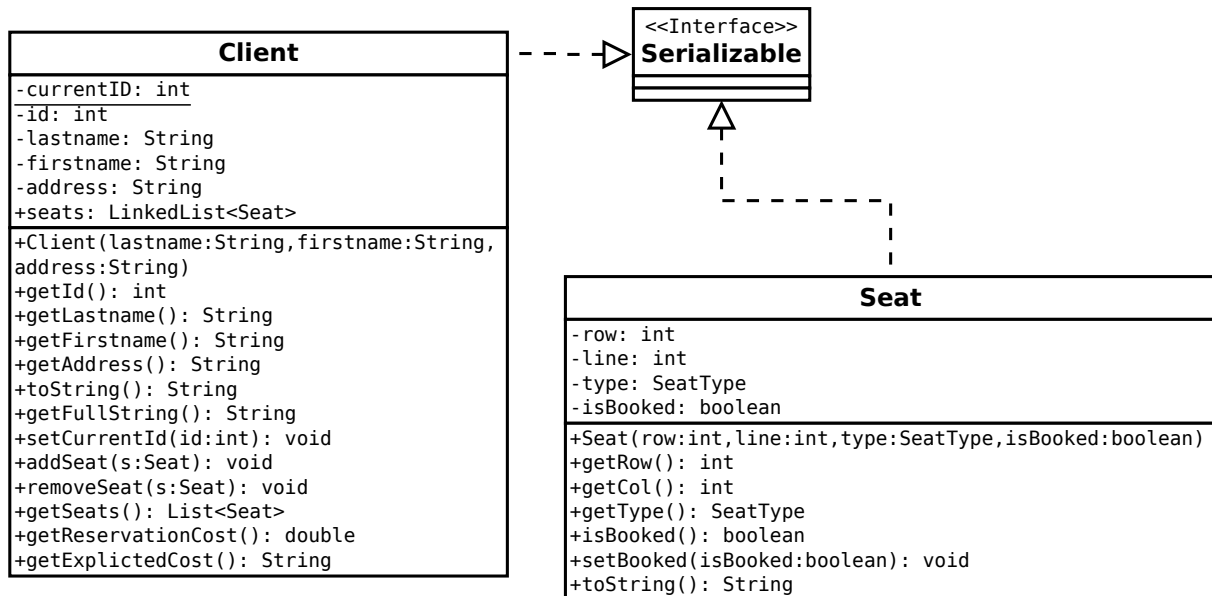
Notre programme est désormais capable de mémoriser une liste de clients ainsi que de sauvegarder la disponibilité des places du théâtre, mais **nous ne mémorisons pas pour qui nous avons réservé un siège !**

EXERCICE 13



- Pour corriger ce défaut, nous allons sauvegarder dans la classe `client`, la liste des places réservées.
- Nous allons mémoriser la liste de siège réservé par le client sous la forme d'une liste chaînée.
 - Rajoutez une méthode **public void** `addSeat(Seat s)` qui ajoute une réservation à la liste du client et une méthode **public void** `removeSeat(Seat s)` qui supprime le siège correspondant.
 - Ajoutez une méthode **public** `List<Seat>getSeats()` qui renvoie la liste de siège réservés.

! N'oubliez pas d'implémenter `Serializable` dans la classe `Seat` De cette manière la sauvegarde de la liste de client par le `Serializer` se fera de manière transparente !



Coût de la réservation

Afin de pouvoir facturer notre client, nous allons rajouter deux autres méthodes :

EXERCICE 14



- getReservationCost() va renvoyer la somme du prix des places réservés par le client. Rappelez-vous qu'un prix a été associé à chaque type de place (SeatType).
- getExplicatedCost() construit une chaîne de caractère permettant au client de visualiser la calcul de sa facture. Cette chaîne de caractère est composée de plusieurs lignes affichant le numéro de la place et son prix entre parenthèse suivi d'une ligne de total.

Exemple de sortie de la méthode getExplicatedCost() :

```
G3 (125.0€)
C1 (20.0€)
Total : 145.0€
```

EXERCICE 15



- Modifiez les méthodes makeReservation et cancelReservation pour prévenir le client sélectionné de l'ajout ou de l'annulation d'une réservation le concernant.
- Ajoutez aussi une action "sr : Show Reservation" qui permet de sélectionner un client et d'afficher la liste de ses réservations personnelles.

```
Welcome to the Reservation Manager
What do you want to do (h for help)
lc
Client list : [Roger Rabbit]
What do you want to do (h for help)
mr
Client list :
Client n°0 : Roger Rabbit (Toonville)
Please enter the id of the wanted client or -1 to cancel :
0
```

```
A X X X X X X
B X e e e e X
C X d e c e d X
D X c X c X c X
E X c b b b c X
F X b a a a b X
G X b a a a b X
H X X S S S X X
  0 1 2 3 4 5 6
```

Please enter row letter:

F

Please enter line number:

4

```
A X X X X X X
B X e e e e X
C X d e c e d X
D X c X c X c X
E X c b b b c X
F X b a a A b X
G X b a a a b X
H X X S S S X X
  0 1 2 3 4 5 6
```

What do you want to do (h for help)

mr

Client list :

Client n°0 : Roger Rabbit (Toonville)

Please enter the id of the wanted client or -1 to cancel :

0

```
A X X X X X X
B X e e e e X
C X d e c e d X
D X c X c X c X
E X c b b b c X
F X b a a A b X
G X b a a a b X
H X X S S S X X
  0 1 2 3 4 5 6
```

Please enter row letter:

F

Please enter line number:

3

```
A X X X X X X
B X e e e e X
C X d e c e d X
D X c X c X c X
E X c b b b c X
F X b a A A b X
G X b a a a b X
H X X S S S X X
  0 1 2 3 4 5 6
```

What do you want to do (h for help)

sr

Client list :

Client n°0 : Roger Rabbit (Toonville)

Please enter the id of the wanted client or -1 to cancel :

0

Roger Rabbit has reserved seats number :

F4 (125.0€)

F4 (125.0€)

```
Total : 250.0€  
What do you want to do (h for help)  
q  
Bye Bye !
```

Seconde exécution :

```
Welcome to the Reservation Manager  
What do you want to do (h for help)  
sr  
Client list :  
Client n°0 : Roger Rabbit (Toonville)  
Please enter the id of the wanted client or -1 to cancel :  
0  
Roger Rabbit has reserved seats number :  
F4 (125.0€)  
F4 (125.0€)  
Total : 250.0€  
What do you want to do (h for help)  
q  
Bye Bye !
```

Félicitations ! Nous avons désormais un logiciel de réservation fonctionnel avec persistance des données ! 😊

Etape 6 : Client VIP et Group

Nous aimerions pouvoir rajouter la possibilité de fidéliser certains client en leur accordant des réductions (clients VIP) ainsi que d'accorder au client commandant un grand nombre de place un tarif de groupe.

Mais nous voudrions faire évoluer notre programme **en modifiant le moins possible** le code existant.

Nous allons utiliser pour cela les concepts **héritage** et de **polymorphisme**.

Nous avons vu dans la partie théorique du cours :

- qu'une classe fille hérite de tous les attributs et méthodes de la classe mère.
- qu'une classe fille peut redéfinir certaines méthodes de la classe mère.
- que les instances de la classe fille peuvent être utilisées à la place de n'importe qu'elle instance de la classe mère (polymorphisme).

Nous allons donc créer des classes filles `ClientVIP` et `ClientGroup` qui hérite de `Client` et redéfinissent les méthodes qui concerne la facturation : `getReservationCost()` et `getExplicatedCost()` !

EXERCICE 16

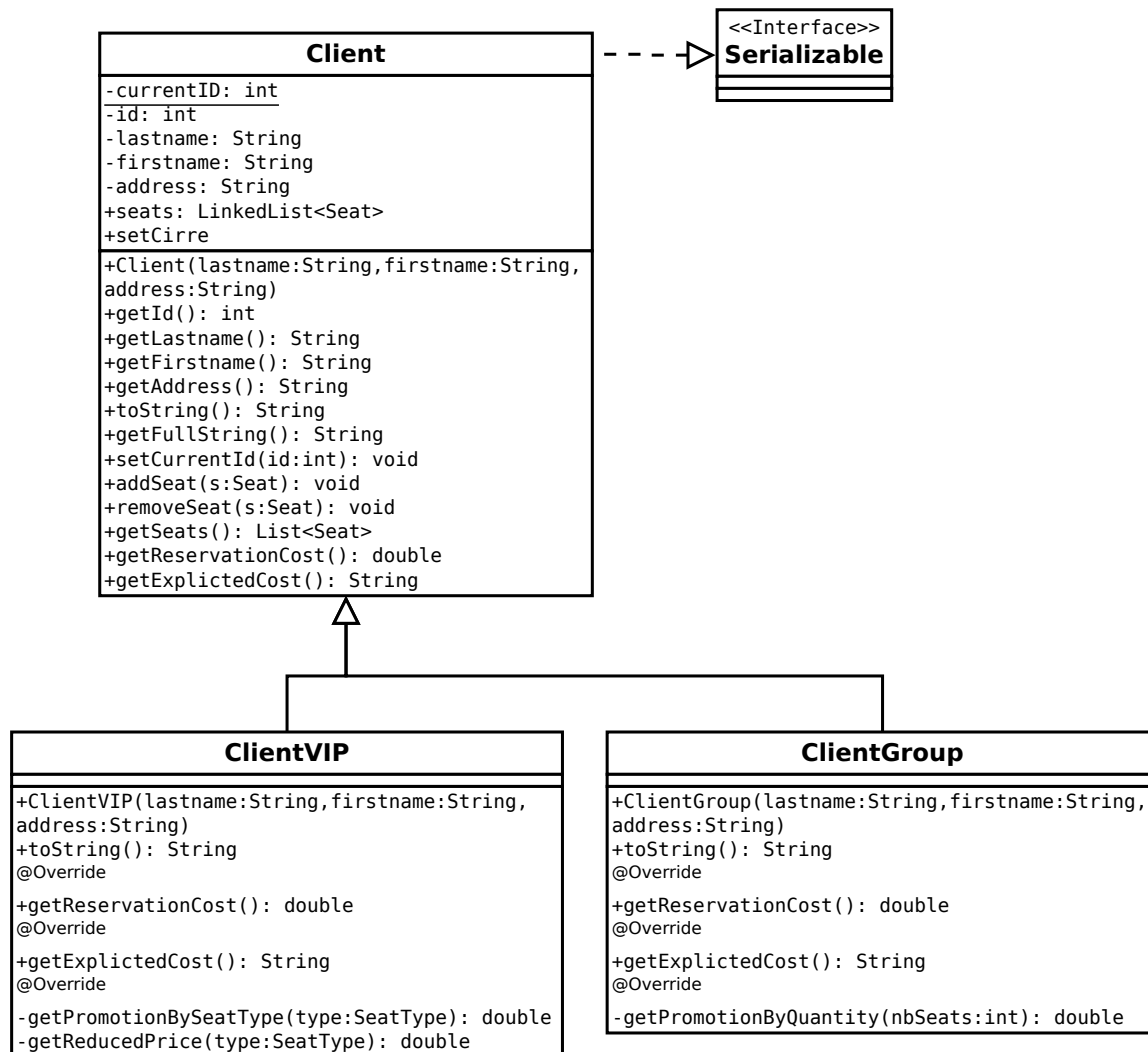


Créez la classe **public class** `ClientVIP` dans le fichier `ClientVIP.java`, en suivant l'UML ci-après.

- le constructeur `ClientVIP` se contente d'appeler le constructeur de la classe mère.
- la méthode **privée** `getPromotionBySeatType` renvoie la promotion attendue sous la forme d'un nombre entre 0 et 1. Les clients VIP ont 30% de réduction sur les places de première catégories, 20% sur les secondes, 10% sur les troisièmes catégories et rien sur les autres.
- la méthode **privée** `getReducedPrice` renvoie le prix de la catégorie une fois la promotion calculée par `getPromotionBySeatType` appliquée.
- `toString` renvoie la chaîne calculée par la classe mère (`super.toString()`) auquel on rajoute le mot clef VIP entre guillemet.
- `getReservationCost` renvoie le coût total de la facture en tenant compte des réductions expliqués ci-avant.
- `getExplicatedCost` renvoie une chaîne de caractères composée de plusieurs ligne affichant le numéro de la place, soit prix d'origine puis le cas échéant la réduction et le prix appliqué, suivit d'une ligne de total.

Exemple d'affichage :

```
F4 (125.0€ -30.0% => 87.5€)  
C5 (20.0€)  
Total : 107.5€
```



EXERCICE 17



Créez la classe **public class** ClientGroup dans le fichier ClientGroup.java, en suivant l'UML ci-avant.

- le constructeur ClientGroup se contente d'appeler le constructeur de la classe mère.
- la méthode **privée** getPromotionQuantity renvoie la promotion attendue sous la forme d'un nombre entre 0 et 1. La réduction de groupe est de 10% entre 5 et 10 places et 20% au dessus de 10 places achetés. Cette réduction s'applique sur l'ensemble de la facture.
- toString renvoie la chaîne calculée par la classe mère (**super.toString()**) auquel on rajoute le mot clef Group entre guillemet.
- getReservationCost renvoie le coût total de la facture en tenant compte des réductions expliqués ci-avant.
- getExplictedCost renvoie une chaîne de caractères composée de plusieurs ligne affichant le numéro de la place, le prix d'origine suivi, le cas échéant de la réduction puis d'une ligne de total.

Exemple d'affichage :

```

F1 (80.0€)
F2 (125.0€)
F3 (125.0€)
F4 (125.0€)
Total : 455.0€
    
```

```

F1 (80.0€)
F2 (125.0€)
F3 (125.0€)
F4 (125.0€)
    
```

```
F5 (80.0€)
-10.0%
Total : 481.5€
```

EXERCICE 18



Modifiez la méthode `addClient` de la classe `ReservationManagerConsole` pour demander à l'utilisateur de choisir le type de client au moment de l'ajout. **C'est tout ce que nous avons à modifier ! Le polymorphisme fait le reste !**

```
Welcome to the Reservation Manager
What do you want to do (h for help)
ac
Lastname : Wayne
Firstname : Bruce
Address : Gotham
Choose client type :
1 - Client
2 - VIP
3 - Group
2
What do you want to do (h for help)
lc
Client list : [Roger Rabbit, Bruce Wayne "VIP"]

...

What do you want to do (h for help)
sr
Client list :
Client n°0 : Roger Rabbit (ToonVille)
Client n°1 : Bruce Wayne (Gotham)
Please enter the id of the wanted client or -1 to cancel :
1
Bruce Wayne "VIP" has reserved seats number :
G2 (125.0€ -30.0% => 87.5€)
G3 (125.0€ -30.0% => 87.5€)
G4 (125.0€ -30.0% => 87.5€)
F2 (125.0€ -30.0% => 87.5€)
F3 (125.0€ -30.0% => 87.5€)
F4 (125.0€ -30.0% => 87.5€)
Total : 525.0€
What do you want to do (h for help)
q
Bye bye !
```

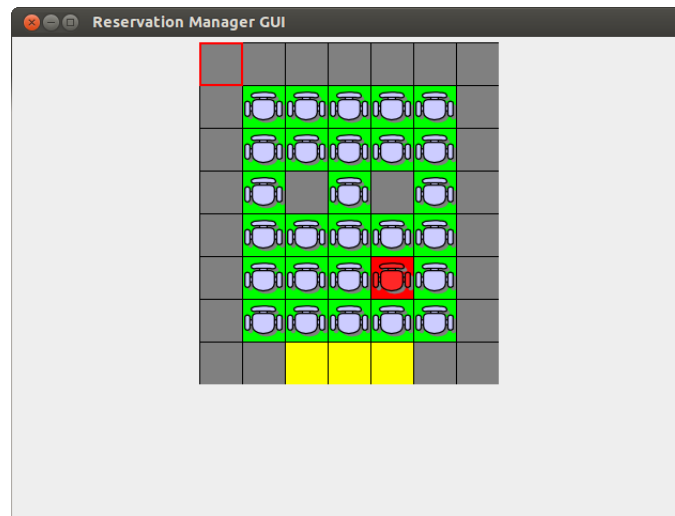


Un code évolutif

En choisissant le polymorphisme plutôt que des nouveaux attributs (`boolean isVIP` et `boolean isGroup` par ex.), cela nous évite de modifier la classe `Client` elle-même et nous donne de la souplesse. Si nous voulons supprimer la notion de tarif de groupe on faire évoluer la classe `ClientVIP` pour prendre en compte des tarifs promotionnels personnalisés, nous pouvons le faire sans modifier le reste du code et donc en limitant le risque d'introduction de nouveaux bugs. Le polymorphisme permet aussi facilement à quelqu'un qui arrive ultérieurement de rajouter des fonctionnalités sans modifier le code de base. Notez cependant que cette souplesse de redéfinition peut aussi être vu comme une faille de sécurité (imaginé une classe `Hacker` qui redéfinit la méthode `getReservationCost` comme renvoyant une valeur négative et se faisant par ce biais rembourser) ! C'est pourquoi on peut choisir de déclarer une classe comme **final**, c'est-à-dire ne pouvant être héritée.

Etape 7 : Le théâtre comme si on y était (Bonus)

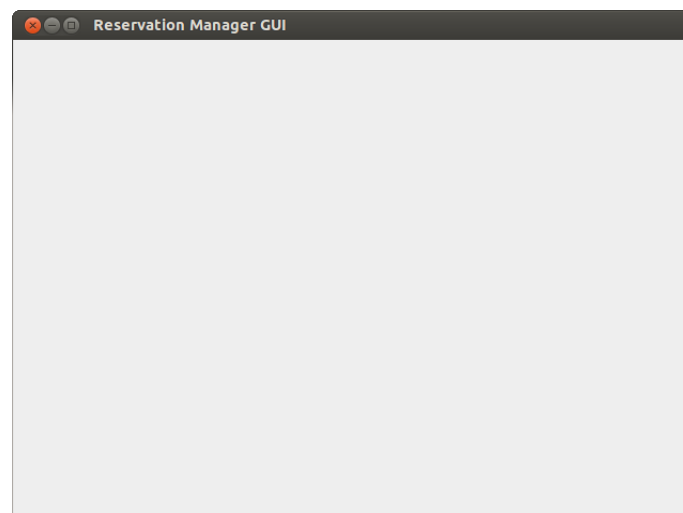
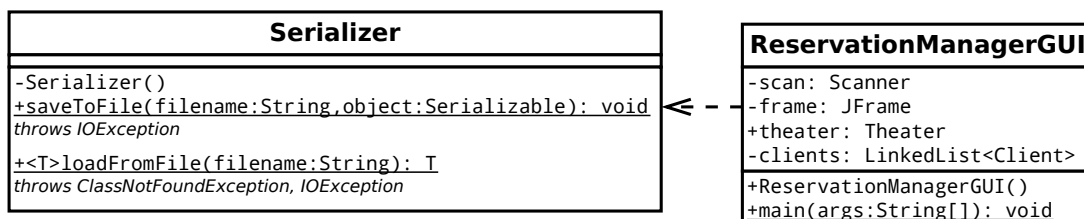
Nous aimerions afficher la disposition du théâtre dans notre interface graphique et pouvoir cliquer sur les cases affichées pour sélectionner les places :



EXERCICE 19



Commencez par créer une classe **public class** `ReservationManagerGUI` (dans un fichier `ReservationManagerGUI.java`) avec une interface graphique minimale (cf. TD1 et TD2) qui initialise une `JFrame` vide et charge les objets `Theater` et `LinkedList<Client>` à partir des fichiers correspondants.



Dessiner son propre composant graphique Lorsque l'on désire avoir un contrôle complet sur l'affichage de son propre composant graphique tout en profitant des facilités de la bibliothèque Swing on utilise généralement l'astuce suivante : **créer une classe fille héritant de `JPanel` et redéfinissant les méthodes suivantes :**

- **public int** `getWidth()`, qui spécifie la largeur du composant pour ses interlocuteurs
- **public int** `getHeight()`, qui spécifie la hauteur du composant pour ses interlocuteurs

- `public Dimension getPreferredSize()`, qui spécifie les dimensions préférées du composant pour ses interlocuteurs
- `public void paint(Graphics g)` qui régit l’affichage du composant.

<http://docs.oracle.com/javase/7/docs/api/index.html?javax/swing/JPanel.html>

Exemple de composant graphique personnalisé :

```
import java.awt.*;
import javax.swing.*;

public class CustomPanel extends JPanel {

    private ImageIcon image; // Internal attribute

    public CustomPanel() { // Constructor
        this.image = new ImageIcon("emosquare.png"); // Image to use during paint
    }

    @Override
    public int getWidth() { // Width of the component
        return 640;
    }

    @Override
    public int getHeight() { // Height of the component
        return 480;
    }

    @Override
    public Dimension getPreferredSize() { // Preferred size of the component
        return new Dimension(getWidth(), getHeight());
    }

    @Override
    public void paint(Graphics g) { // How the component is drawn
        super.paint(g); // Firstly, draw the super-class (the original JPanel)

        // Custom drawing by method calls to the Graphics instance (see java.awt.Graphics)
        g.setColor(Color.BLACK); // Set the color to Black
        g.fillRect(0,0, getWidth(), getHeight()); // Fill a rectangle where top-left corner is at
            // (0,0) and of the same size than the component
        g.setColor(Color.YELLOW); // Set the color to Yellow
        g.drawRect(50,100, 200,300); // Draw a rectangle where top-left corner is at (50, 100)
            // and of size 200x300.
        this.image.paintIcon(this, g, 50, 100); // Paint the image where top-left corner is at
            // (50,100);
    }
}
```

<http://docs.oracle.com/javase/7/docs/api/index.html?javax/swing/ImageIcon.html>

<http://docs.oracle.com/javase/7/docs/api/index.html?java/awt/Graphics.html>

Exemple d’interface utilisant ce composant graphique personnalisé :

```
import java.awt.*;
import javax.swing.*;

public class MainGUI {

    public MainGUI() {
        JFrame frame = new JFrame("CustomPanel GUI");
        frame.setMinimumSize(new Dimension(640,480));
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());

        CustomPanel cp = new CustomPanel(); // Create a new CustomPanel
    }
}
```



```
frame.add(cp, BorderLayout.CENTER); // Add it to the frame

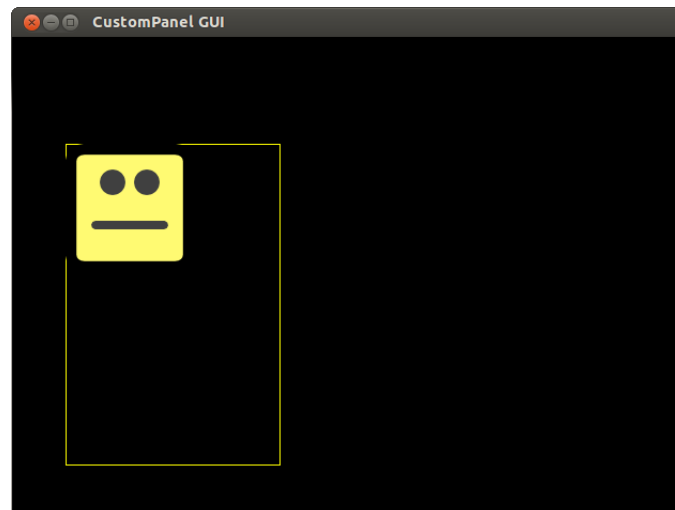
frame.pack();
frame.setVisible(true);
}

public static void main(String[] args) {
    new MainGUI();
}
}
```

Exemple d'affichage (si `emosquare.png` est absent)



Exemple d'affichage (si `emosquare.png` est présent)



Fichier image absent Remarquez que si votre fichier image est absent cela ne provoque pas d'exception. Le programme se contente de ne pas afficher l'image.

EXERCICE 20



Copiez/Collez les classes `public class CustomPanel` et `public class MainGUI` dans leurs fichiers correspondants pour vous familiariser avec la création de composant personnalisé. N'hésitez pas à les modifier pour tester le fonctionnement des différents paramètres.

Dessiner notre théâtre La classe Theater mémorise les informations relatives aux emplacements sous la forme d'un tableau 2D. Nous désirons retranscrire ce tableau sous une forme graphique.

Nous allons considérer que chaque case du tableau sera représentée par un carré dont la couleur dépendra du type d'emplacement (obstacle, scène, siège libre, siège réservé, etc.) auquel on adjoindra éventuellement une image.

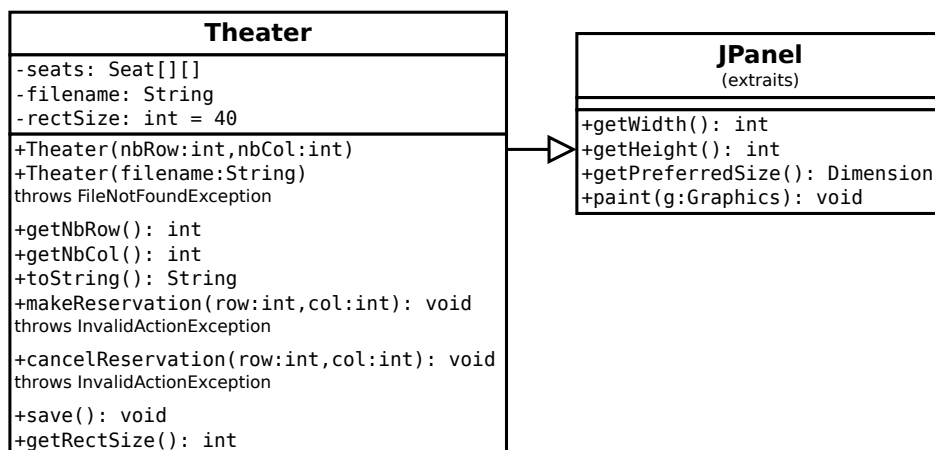
Nous décidons arbitrairement qu'une case de tableau sera affichée avec une largeur de 40 pixels (que nous noterons `rectSize`). Notre composant aura donc pour largeur ($rectSize \times nbCol$) et pour hauteur ($rectSize \times nbRow$).

Pour l'affichage lui-même nous utiliserons la méthode `fillRect` (comme vu dans l'exemple `CustomPanel`) pour dessiner chacune de nos cases une par une (avec une double boucle donc). La couleur et l'image choisie dépendront de la type de siège et de s'il est réservé ou non (**switch** est votre ami).

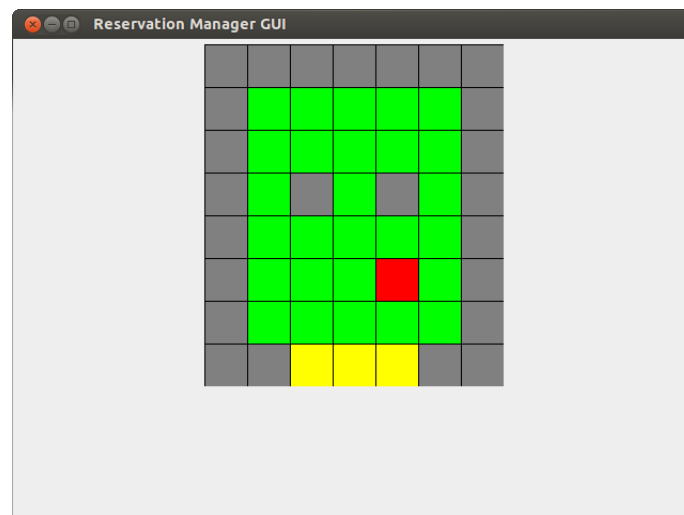
EXERCICE 21



Modifiez votre classe `Theater` pour en faire un composant graphique.



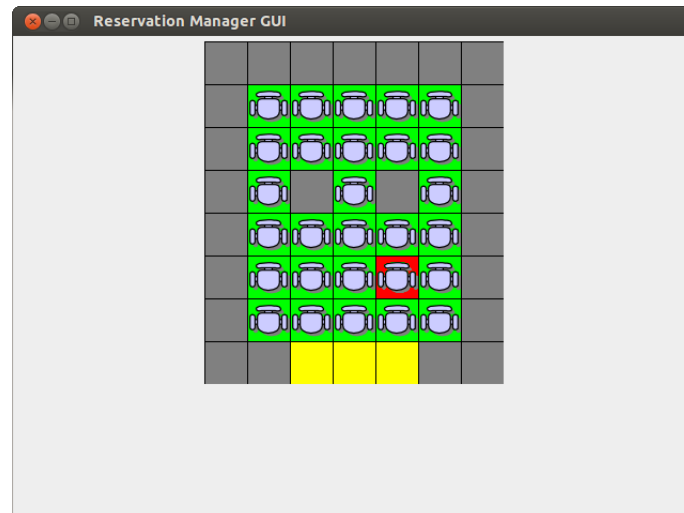
Exemple d'affichage de votre théâtre (le carré rouge correspondant à un siège réservé dans l'interface console) :



Récupérez l'image de siège sur <http://learning.esiea.fr/>



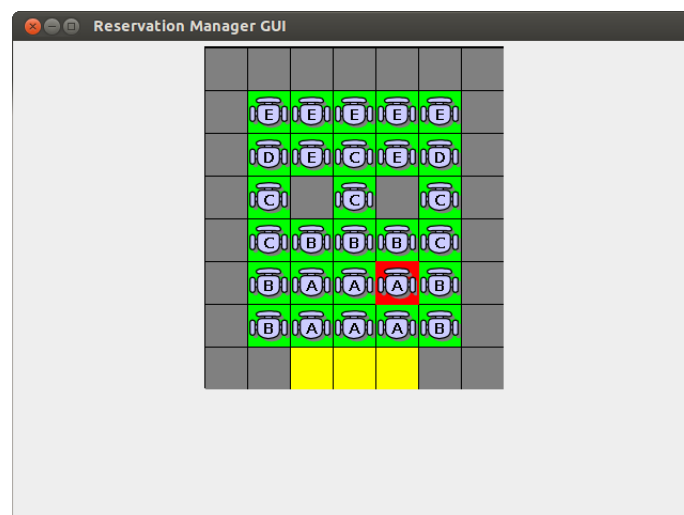
Exemple d'affichage de votre théâtre (avec l'image de siège) :



Afficher les catégories

Afficher les catégories au niveau des sièges nécessite de nombreuses astuces... c'est pourquoi je vous propose de copier/coller les lignes suivantes dans votre fonction `paint(Graphics g)`

```
/* Loading a new font (14pt, bold) */
g.setFont(new Font("default", Font.BOLD, 14));
/* Getting the category String */
String category = seats[row][col].getType().toString().toUpperCase();
/* Calculating the String dimension in pixels */
Rectangle2D stringDim = g.getFontMetrics().getStringBounds(category, g);
/* Drawing the string centered within the rectangle and centered from the string itself */
g.drawString(category, col * rectSize + rectSize / 2 - (int) stringDim.getWidth() / 2, row *
    rectSize + rectSize / 2 + (int) stringDim.getHeight() / 2);
```



Interaction souris Pour pouvoir sélectionner un siège dans notre interface il nous faut récupérer les clics souris. Pour cela nous allons rajouter un `MouseListener`. Ce gestionnaire d'événements s'ajoute sur les composants (héritant de `java.awt.Component`) et donc par extension sur les `JPanel`.

<http://docs.oracle.com/javase/7/docs/api/index.html?java/awt/Component.html>

<http://docs.oracle.com/javase/7/docs/api/index.html?java/awt/event/MouseListener.html>

Cette interface demande d'implémenter les cinq fonctions suivantes :

- `public void` `mouseEntered(MouseEvent e)`, lorsque la souris entre dans la zone du composant,
- `public void` `mouseExited(MouseEvent e)`, lorsque la souris sort de la zone du composant,
- `public void` `mousePressed(MouseEvent e)`, lorsque le clic souris est appuyé,
- `public void` `mouseReleased(MouseEvent e)`, lorsque le clic souris est relâché,
- `public void` `mouseClicked(MouseEvent e)`, lorsque le clic souris est appuyé puis relâché.

Les informations concernant la position de la souris et le bouton cliqué sont contenues dans l'objet `MouseEvent`.

<http://docs.oracle.com/javase/7/docs/api/index.html?java/awt/event/MouseEvent.html>

Exemple :

```
public void mouseClicked(MouseEvent ev) {

    switch(ev.getButton()) {
        case MouseEvent.BUTTON1 :
            System.out.println("BUTTON1");
            break;
        case MouseEvent.BUTTON2 :
            System.out.println("BUTTON2");
            break;
        case MouseEvent.BUTTON3 :
            System.out.println("BUTTON3");
            break;
    }
    System.out.println(ev.getClickCount()+ " clicks");
    Point p = ev.getPoint();
    System.out.println("Mouse is localized at (" + p.x + ", " + p.y + ") pixels from the top-left
        corner of the SOURCE");
    Point p2 = ev.getLocationOnScreen();
    System.out.println("Mouse is localized at (" + p2.x + ", " + p2.y + ") pixels from the top-left
        corner of the SCREEN");
}
```

Ajout du `MouseListener` à l'interface graphique.

```
public class ReservationManagerGUI implements MouseListener {

    public ReservationManagerGUI() {
        /* ... */
        theater.addMouseListener(this);
        /* ... */
    }

    public void mouseClicked(MouseEvent ev) {
        /* ... */
    }
}
```

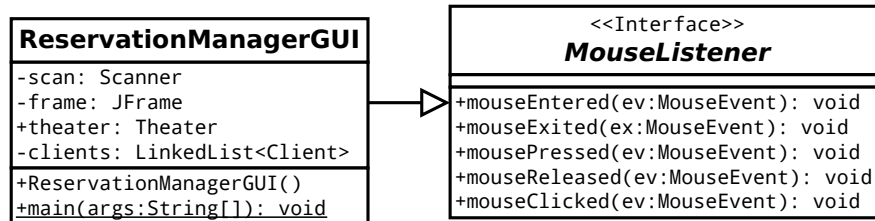
Exemple d'affichage :

```
LEFT BUTTON
2 clicks
Mouse is localized at (102, 97) pixels from the top-left corner of the SOURCE
Mouse is localized at (282, 154) pixels from the top-left corner of the SCREEN
```

EXERCICE 22



Implémentez l'interface `MouseListener` dans votre classe `ReservationManagerGUI`.

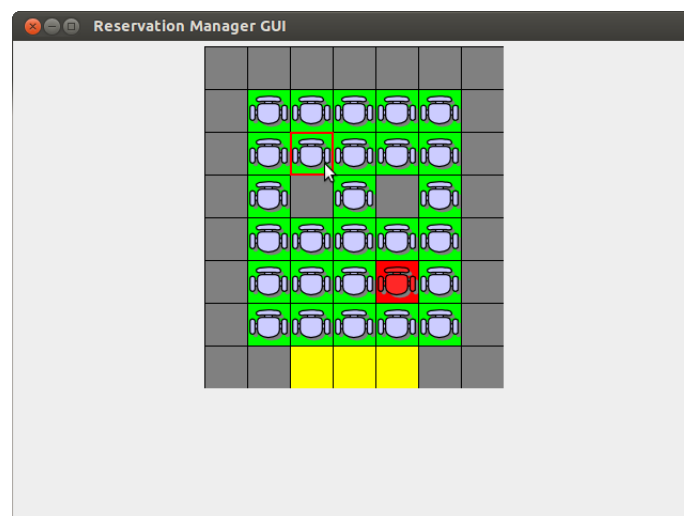
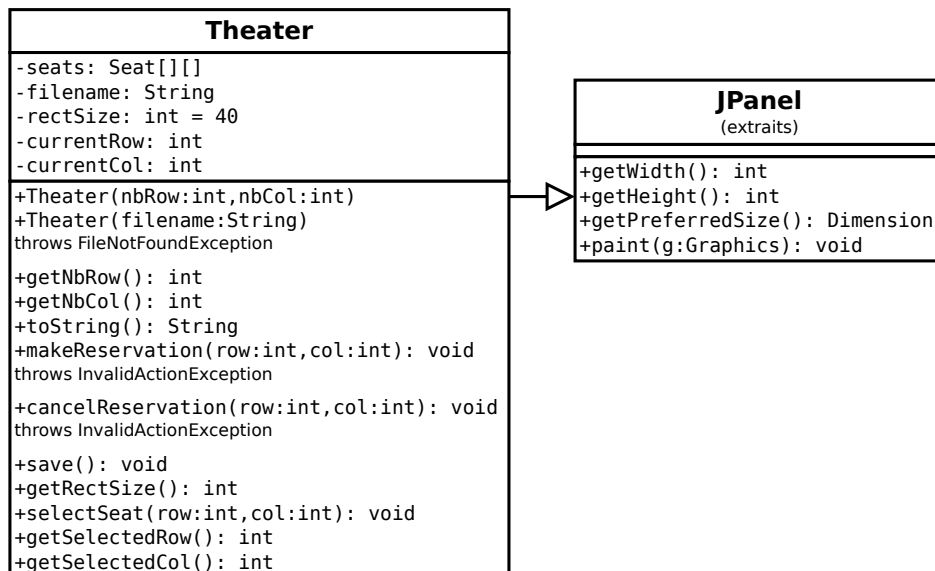


Mémoriser et visualiser le siège sélectionné

EXERCICE 23



- Rajoutez deux attributs `currentRow` et `currentCol` à votre classe `Theater`.
- Rajoutez les méthodes `public selectSeat(int row, int col)`, ainsi que les accesseurs `public int getSelectedRow()` et `public int getSelectedCol()`
- Modifiez la méthode `paint` pour rajouter un rectangle rouge autour de l'emplacement sélectionné.
- Modifiez la méthode `mouseClicked` de la classe `ReservationManagerGUI` pour modifier le siège sélectionné du théâtre.



Etape 8 : Proposer des choix

Si notre interface nous permet désormais de sélectionner un emplacement, il nous manque encore la possibilité :

1. de créer un nouveau client
2. de sélectionner un client
3. de supprimer le client sélectionné
4. d'afficher les réservations du client sélectionné
5. d'établir une réservation pour le client et l'emplacement sélectionnés
6. d'annuler une réservation pour le client et l'emplacement sélectionnés

Éruption de boutons

EXERCICE 24



Rajoutez des boutons dans votre interfaces qui déclencheront les actions 1, 3, 4 et 5, décrites ci-dessus. Ces boutons sont reliés à la classe ReservationManagerGUI qui implémentera donc l'interface ActionListener.

Remarque : Pour le moment, la méthode actionPerformed ne fait rien.

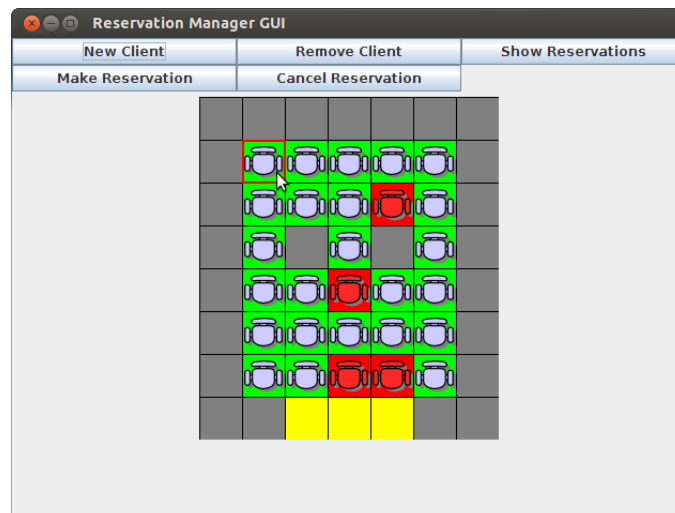
<http://docs.oracle.com/javase/7/docs/api/index.html?javax/swing/JButton.html>

<http://docs.oracle.com/javase/7/docs/api/index.html?java/awt/event/ActionListener.html>



Pour afficher les boutons sur plusieurs lignes Pensez à utiliser un GridLayout par exemple.

<http://docs.oracle.com/javase/7/docs/api/index.html?java/awt/GridLayout.html>



Liste de clients et sélection Pour afficher les clients existants (que nous avons crée avec l'interface console par exemple) nous allons utiliser une **liste déroulante** (JComboBox<T>).

<http://docs.oracle.com/javase/7/docs/api/index.html?javax/swing/JComboBox.html>

Exemple (tiré de la doc Java) :

```
String[] petStrings = { "Bird", "Cat", "Dog", "Rabbit", "Pig" };

//Create the combo box, select item at index 4.
//Indices start at 0, so 4 specifies the pig.
JComboBox<String> petList = new JComboBox<String>(petStrings);
petList.setSelectedIndex(4);
petList.setActionCommand("combo");
petList.addActionListener(this); // Notify when a selection has been made
```



Notez que vous pouvez aussi passer des objets complexes à votre JComboBox<T> (notez le caractère générique de la classe) :

```
// INIT
JComboBox<Car> carCombo = new JComboBox<Car>(); // Empty combobox

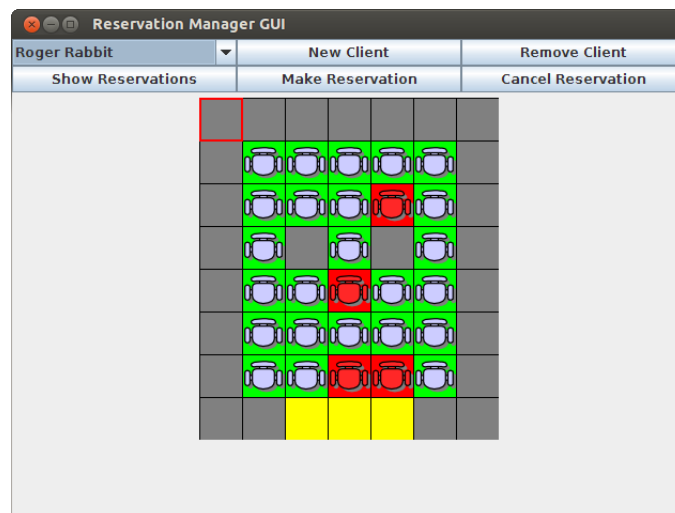
// REFRESH
carCombo.removeAllItems(); // Remove previous items
for(Car c : carList) {
    carCombo.addItem(c); // Add each car to the combobox
}
carCombo.updateUI(); // Refresh the component graphics

// GET SELECTED ITEM
Car selectedCar = carCombo.getItemAt(carCombo.getSelectedIndex()); // Get the car that is
    currently selected
```

EXERCICE 25



- Rajoutez une liste déroulante d'objet Client dans votre interface graphique
- Créez une méthode **public void** refreshCombo() dans votre classe ReservationManagerGUI, qui rafraîchit la liste déroulante en fonction de la liste chaînée de clients. Cette méthode est appelée à chaque modification de la liste chaînée.



EXERCICE 26



Modifiez la méthode actionPerformed de la classe ReservationManagerGUI pour gérer les actions suivantes :

- supprimer le client sélectionné de la liste
- faire une réservation (en utilisant le client sélectionné et l'emplacement sélectionné).
- annuler une réservation (en utilisant le client sélectionné et l'emplacement sélectionné).

Remarque : En cas de `InvalidActionException` vous afficherez une fenêtre de dialogue d'erreur (voir un peu plus bas).



Rafraîchir l'interface Après avoir modifié les données d'un composant graphique il est essentiel de lui demander de se rafraîchir/de se redessiner. Par exemple, après avoir ajouter un nouveau client, on voudra rafraîchir la liste déroulante associée ou après avoir fait une réservation, on voudra rafraîchir le composant Theater. On utilise pour cela la méthode `updateUI` lorsqu'elle existe (`JComboBox`, `JPanel`, mais pas `JFrame` !)

Exemple :

```
theater.makeReservation(theater.getSelectedRow(), theater.getSelectedCol()); // Data
Modification
theater.updateUI(); // Refresh User Interface
```

Interface de dialogue : Créer un nouveau client Pour créer un nouveau client il nous faut demander trois informations à l'utilisateur (Nom, Prénom, Adresse). Pour éviter d'alourdir la fenêtre principale nous n'allons pas ajouter les champs `TextField` à l'interface générale, mais nous allons utiliser une fenêtre de dialogue.

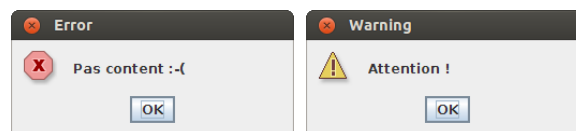
Les fenêtres de dialogue permettent de fournir des messages plus ou moins simples à l'utilisateur.

Nous avons vu dans les TD précédents l'usage de la classe `JOptionPane` pour afficher des messages simples à l'utilisateur :

<http://docs.oracle.com/javase/7/docs/api/index.html?javax/swing/JOptionPane.html>

Exemple :

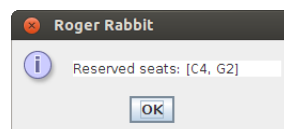
```
JOptionPane.showMessageDialog(null, "Pas content :-(", "Error", JOptionPane.ERROR_MESSAGE);
JOptionPane.showMessageDialog(null, "Attention !", "Warning", JOptionPane.WARNING_MESSAGE);
```



EXERCICE 27



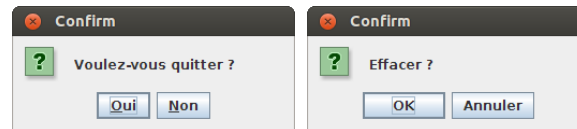
Modifiez la méthode `actionPerformed` de votre classe `ReservationManagerGUI` pour réagir au clic sur le bouton "Show Reservation" en affichant une fenêtre de dialogue présentant la liste des sièges réservés pour le client sélectionné. (Remarque : l'icône utilisée est celle associée à `JOptionPane.INFORMATION_MESSAGE`)



On peut aussi utiliser les fenêtres de dialogue pour afficher des dialogues interactifs :

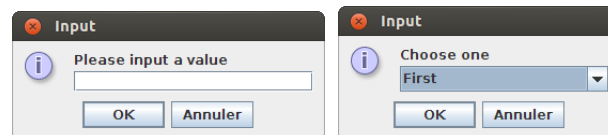
```
// YES/NO DIALOG
int val = JOptionPane.showConfirmDialog(null, "Voulez-vous quitter ?", "Confirm", JOptionPane.
    YES_NO_OPTION);
switch(val) {
    case JOptionPane.YES_OPTION: /* Quit */ break;
    case JOptionPane.CLOSED_OPTION: // If the dialog is closed
    case JOptionPane.NO_OPTION:
        break;
}

// OK/CANCEL DIALOG
val = JOptionPane.showConfirmDialog(null, "Effacer ?", "Confirm", JOptionPane.OK_CANCEL_OPTION);
switch(val) {
    case JOptionPane.OK_OPTION: /* Go ! */ break;
    case JOptionPane.CLOSED_OPTION: // If the dialog is closed
    case JOptionPane.CANCEL_OPTION:
        break;
}
```

```
// ASK FOR A STRING
String inputValue = JOptionPane.showInputDialog(null, "Please input a value", "Input",
    JOptionPane.INFORMATION_MESSAGE); // To ask for a String

// ASK FOR A CHOICE
Object[] possibleValues = { "First", "Second", "Third" }; // items for the combobox
Object selectedValue = JOptionPane.showInputDialog(null, "Choose one", "Input", JOptionPane.
    INFORMATION_MESSAGE, null, possibleValues, possibleValues[0]);
if(selectedValue != null) { // If closed, the dialog return null
    System.out.println("Chosen index : "+Arrays.binarySearch(possibleValues,selectedValue));
}
```



Interface de dialogue personnalisée Pour personnaliser notre interface de dialogue nous allons créer un JPanel et lui ajouter les composants désirés : des JLabel pour afficher le nom du champ et des JTextField pour les champs de saisie.

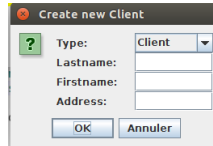
! Référence sur les champs de saisie Pour pouvoir récupérer les informations saisies, il est important de garder une référence

```
/* DATA FIELD */
JComboBox<String> clientTypeChoice = new JComboBox<>();
clientTypeChoice.addItem("Client");
clientTypeChoice.addItem("VIP");
clientTypeChoice.addItem("Group");
JTextField lastnameField = new JTextField(5);
JTextField firstnameField = new JTextField(5);
JTextField addressField = new JTextField(5);

/* BUILDING OF THE FORM */
JPanel myPanel = new JPanel(new GridLayout(4,2));
myPanel.add(new JLabel("Type:"));
myPanel.add(clientTypeChoice);
myPanel.add(new JLabel("Lastname:"));
myPanel.add(lastnameField);
myPanel.add(new JLabel("Firstname:"));
myPanel.add(firstnameField);
myPanel.add(new JLabel("Address:"));
myPanel.add(addressField);

/* CALLING THE DIALOG */
int result = JOptionPane.showConfirmDialog(null, myPanel,
    "Create new Client", JOptionPane.OK_CANCEL_OPTION);

/* ANALYSING RESULT */
if (result == JOptionPane.OK_OPTION) {
    System.out.println("Type :"+clientTypeChoice.getItemAt(clientTypeChoice.getSelectedIndex()));
    System.out.println("Lastname :"+lastnameField.getText());
    System.out.println("Firstname :"+firstnameField.getText());
    System.out.println("Address :"+addressField.getText());
}
```



EXERCICE 28



Modifiez la méthode `actionPerformed` de votre classe `ReservationManagerGUI` pour réagir au clic sur le bouton "New Client" en affichant une fenêtre de dialogue. Selon la réponse de l'utilisateur, la méthode ajoutera ou non un nouveau client à la liste.

Félicitations ! Vous avez désormais un gestionnaire de réservation fonctionnel !

Pour aller plus loin

Parce qu'on peut toujours avoir d'autres envies, on peut envisager de :

- gérer différentes salles d'un même théâtre
- gérer la notion de séances (1 spectacle à 1 heure donnée dans 1 salle donnée)
- etc.