PROJET SOAP

Architecture Logicielle Distribuée

Etienne MARTIN-CHANTEREAU
Christophe HOUBRON

Master Informatique Parcours ICO

Table des matières

Version Non Distribuée

Dans un premier temps, nous créons en dur toutes les instances de classes, dans MainApp afin de pouvoir tester notre algorithme qui va rechercher des chambres de libres:

- les hôtels
- les chambres
- les adresses pour chaque hotel

- des clients
- des reservations

Nous associons les adresses et les chambres aux différents hôtels, les réservations à différentes chambres et à différents clients.
Nous regroupons nos différents hôtels dans une ArrayList afin de pouvoir rechercher des chambres disponibles en fonction de différents critères (adresse de l'hôtel, nombre d'étoiles par exemple).
L'éxécution de MainApp.java lance la méthode saisieRecherche() . Cette méthode reçoit en paramètre une ArrayList d'hôtel sur laquelle nous allons itérer, puis sur chaque chambre de chaque hôtel.
Dans un premier temps, nous demandons de renseigner les paramètres qui vont permettrent de rechercher une chambre de libre en fonction de différent critères :

- nom de la ville, qui doit correspondre à l'une des énumérations de la classe énumération Ville
- date d'arrivée
- date de départ
- nombre de place voulant être réservé
- prix minimun
- prix maximum
- le nombre d'étoiles désiré

Pour chacun de ses paramètres, nous avons mis en place une boucle do....while avec une vérification de l'entrée afin que cela corresponde à une entrée valide, avec le système try/catch .
Explications du système de vérification de l'entrée :

- nous entrons une première fois dans la boucle via le do , et affichons un message d'invite d'entrée.
- nous récupérons via un scanner l'entrée utilisateur :
  - soit il n'y a pas de message d'erreur qui est capturé par le catch et nous sortons de la boucle
  - while afin de passer à l'entrée suivante
  - soit l'entrée utilisateur ne correspond pas à une entrée valide, il y a alors une exception qui est
  - capturée par le catch et qui va afficher un message d'erreur. l'utilisateur va alors revenir au début de la boucle afin de saisir de nouveau une entrée valide.

Voici un exemple pour le prix maximum :

```
isDone = false; //réinitialisation de isDone
    prixMax = 0;
    do {
      try {
        System.out.println("Veuillez indiquer un prix maximum (supérieur à " +
prixMini +" euros) :");
        //prixMax = input.nextDouble();
        prixMax = Double.parseDouble(input.nextLine());
        isDone = true;
      } catch (InputMismatchException e) {
        System.err.println("Erreur de saisie, veuillez entrer un réel.");
        input.nextLine();
      }
    } while (! (isDone && prixMax > prixMini));
```

Une fois toutes les entrées valides et stockées dans des variables, nous pouvons créer une réservation :

```
resa = new Reservation(dateArr, dateDepart, nbPlaces);
```

Méthode RechercheChambre () :

Nous lançons ensuite la méthode rechercheChambre() , qui prend en paramètre :

- la listes des hôtels
- la reservation nouvellement créé
- ainsi que tous les autres paramètres renseignés par l'utilisateur
  - prix minimum
  - prix maximum
  - ville

- nombre d'étoile

Cette méthode rechercheChambre() va renvoyer une HashMap <Hotel, ArrayList> de chambres libres en fonction des critères définis par l'utilisateur.
La méthode itère sur la liste d'hôtel, et vérifie les critères de l'emplacement (Ville) et du nombre d'étoiles. Si ces critères correspondent aux critères de l'utilisateur, on peut alors itérer sur les listes des chambres :

```
if (hotel.getAdresse().getVille().equals(villeI) &&
(hotel.getEtoiles().equals(etoilesI))) {
  for (Chambre chambre : hotel.getListChambres()){...
```

Si il n'y a aucune réservation pour la chambre, et que le prix minimum, maximum et le nombre de places correspondent, cette chambre est alors directement enregistrée dans la HashMap :

```
if ((chambre.getListReservationsChambre().size() == 0)
              && (chambre.getNbLits() >= reservationI.getNbLitsResa())
              && (prixMin <= chambre.getPrixChambre() && chambre.getPrixChambre() <=
PrixMax)){
            valueHashMap.add(chambre);
          }
```

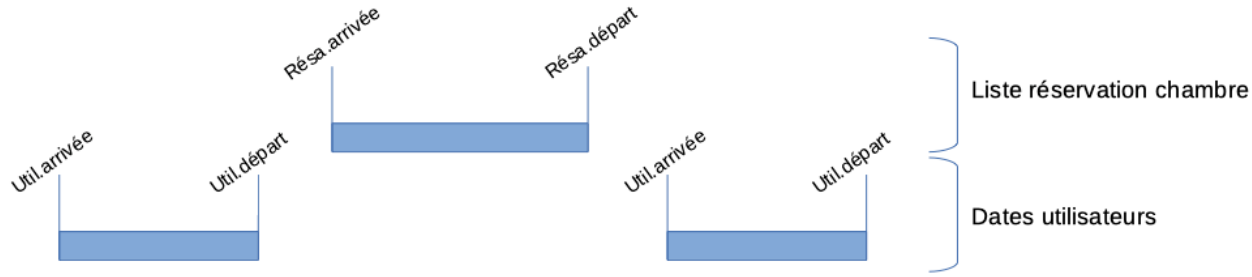
Si il y a des réservations pour la chambre, on itère sur les réservations de la chambre pour vérifier qu'il n'y a pas de réservation à la date choisie par l'utilisateur :

```
boolean isAvailable = true;
for (Reservation reservationChambre : chambre.getListReservationsChambre()) {
              if ( !
(reservationI.getDateDepart().isBefore(reservationChambre.getDateArrivee())
                ||
reservationChambre.getDateDepart().isBefore(reservationI.getDateArrivee()))
                || (chambre.getNbLits() < reservationI.getNbLitsResa())
                || (prixMin > chambre.getPrixChambre() && chambre.getPrixChambre() >
PrixMax)){
                isAvailable = false;
              }
```

Explication du test des critères sur la chambre :
Si tous les critères de réservation sont respecté, on entre jamais dans le if , et on ne modifie donc pas la valeur isAvailable .

```
if (isAvailable) {
  valueHashMap.add(chambre);
}
```

On ajoutera alors la chambre à la ArrayList.
A l'inverse, si un seul des critères de réservation (prix, nombre de place..) n'est pas respecter , on entre dans le if et on change la valeur isAvailable = false . La chambre ne sera pas ajouter à l'ArrayList valueHashMap.
Explication de la vérification des dates de réservations des chambres :



La chambre peut être réservé si :

- la date de départ de réservation de l'utilisateur (Util.départ) a lieu avant la date d'arrivée d'une
- réservation de chambre (Résa.arrivée)
- ou si la date d'arrivée d'un utilisateur (Util.arrivée) a lieu après la date de départ d'une réservation de chambre (Résa.départ)

Donc, à l'inverse, si cela ne correspond pas à ces critères, la chambre ne pourra pas être réservé. C'est ce qui est exprimé dans cette condition if :

```
if ( ! (reservationI.getDateDepart().isBefore(reservationChambre.getDateArrivee())
        || reservationChambre.getDateDepart().isBefore(reservationI.getDateArrivee()))
```

Comme précédement, on test le nombre de place ainsi que le prix minimum et maximum :

```
        || (chambre.getNbLits() < reservationI.getNbLitsResa())
        || (prixMin > chambre.getPrixChambre() && chambre.getPrixChambre() > PrixMax)){
        isAvailable = false;
        }
```

Si l'une de ses conditions est respectée (ex: le prix de la chambre est trop élevée par rapport à la demande utilisateur), alors on entre dans la condition et on change la valeur isAvailable =false . La chambre ne sera alors pas ajoutée à l'ArrayList valueHashMap .

Une fois que tous les hôtels ont été testé, et que l'on a récupérer la HashMap des chambres libres, on utilise la méthode affiRetourListeChambreLibre() qui va permettre d'afficher les chambres disponibles en fonctions des hôtels. L'utilisateur va ainsi avoir une vue globale de toutes les chambres disponibles.

L'utilisateur sera ensuite invité a sélectionner la chambre de son choix en renseignant le nom de l'hotel puis la chambre. Nous avons également ici utiliser le système try/catch afin de s'assurer des entrées de l'utilisateur. En effet, si l'entrée de l'utilisateur ne correspond pas à la clé du dictionnaire associée à un nom d'hôtel, l'utilisateur est alors invité à renouveler son entrée.

Exemple avec le nom de l'hôtel :

```
isDone = false;
    do {
      try {
          System.out.println("Entrée un Hotel parmi les choix proposés");
          hotelName = input.nextLine();
          for (Entry<Hotel, ArrayList<Chambre>> entryHotelName :
chambreDispo.entrySet()) {
            if(entryHotelName.getKey().getNom().equals(hotelName)) {
              isDone = true;
              for (Chambre chambre : entryHotelName.getValue()) {
                System.out.println("La "+chambre+" est disponible");
              }
            }
          }
```

```
            if (isDone == false) {System.out.println("erreur vous n'avez pas sélectionné
un hotel parmi la liste choisi");}
        } catch (Exception e) {}
    } while (! isDone);
```

Losque la chambre est choisie, l'utilisateur doit alors entrer son nom, prénom et toutes les données concernant sa carte bancaire. Nous pouvons ainsi instancier la classe CarteBancaire et la classe Client :

```
CarteBancaire cb = new CarteBancaire(numeroCarte, dateExpiration, cryptogramme);
Client testClient = new Client(nom, prenom,cb);
Reservation testResa = new Reservation(dateArr, dateDepart, nbPlaces,testClient);
```

Ensuite, nous ajoutons la réservation à la chambre :

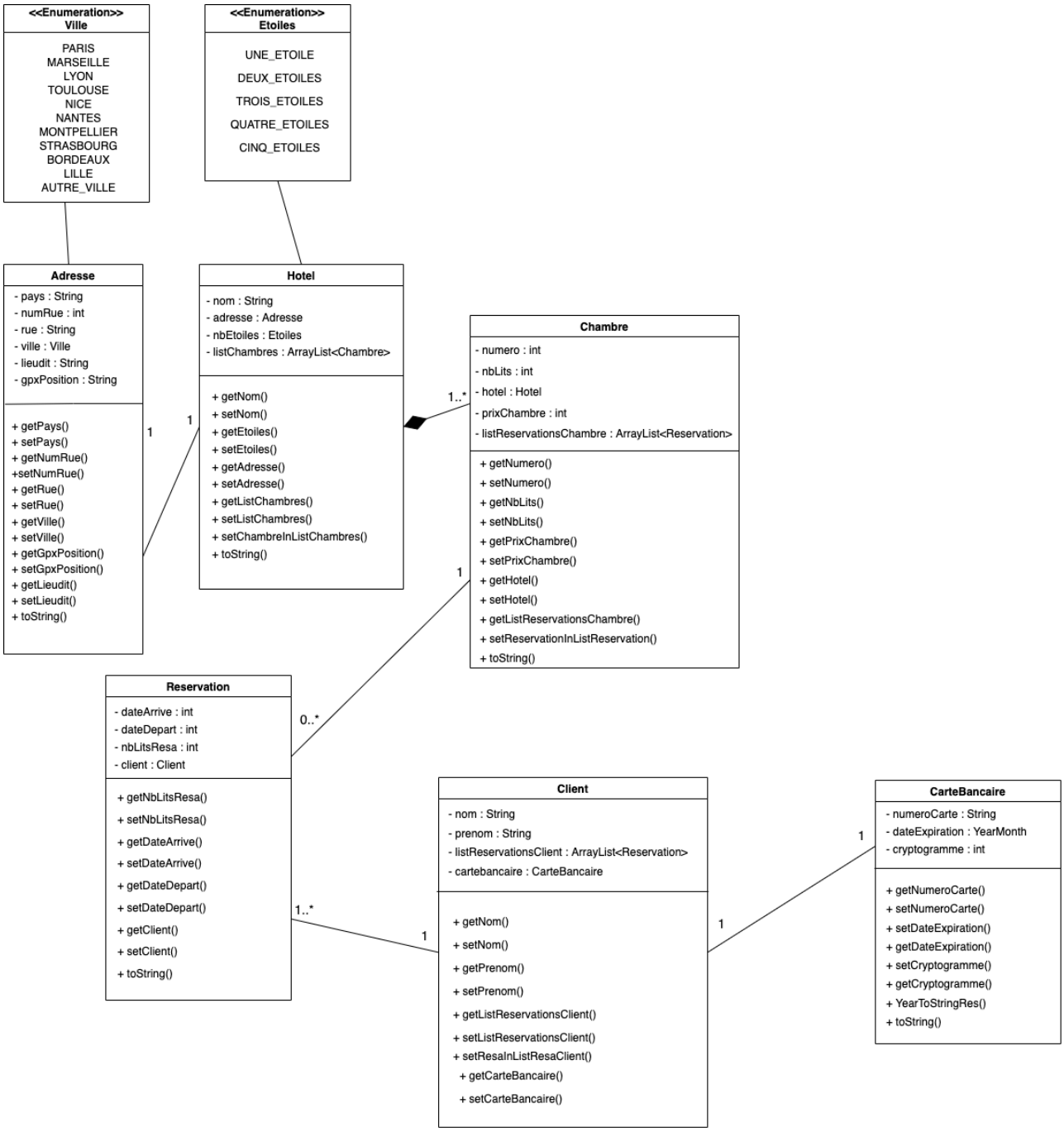
```
addResaToChambre(chambreSelect, testResa);
```

Pour finir, on affiche un récapitulatif de la commande :

```
Voici un récapitulatif de votre commande :

- Nom : XXXXXXXXXX
- Prénom : XXXXXXXXXXX
- Date de réservation : Arrivée le xx-xx-xxxx, départ le xx-xx-xxxx
- Nom de l'hôtel : XXXX Adresse : XXXXXXXXXX
- Numéro de chambre : XX
```

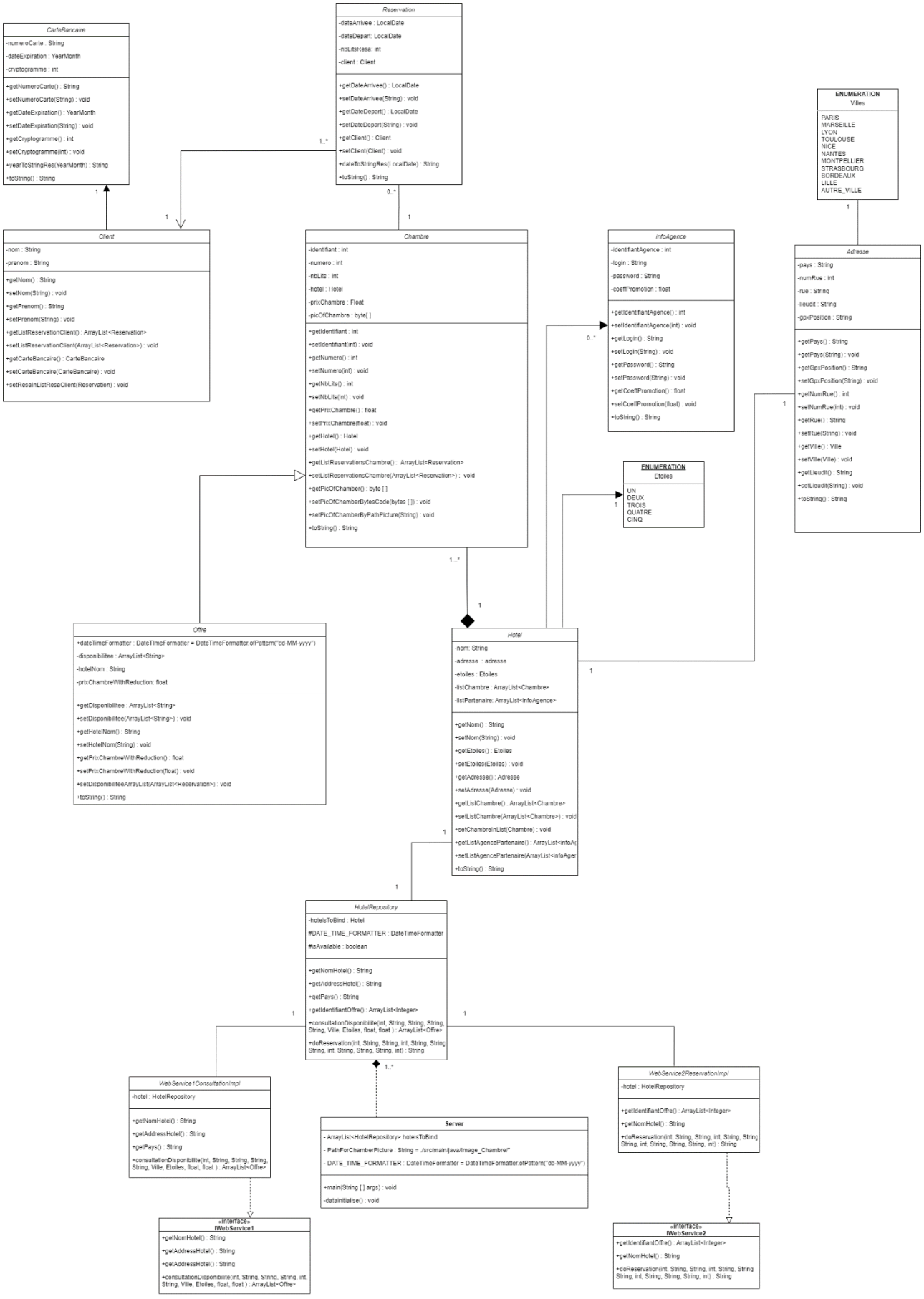
UML



Version Distribuée

Version avec 1 HOTEL ET 1 AGENCE : DESIGN SINGLETON

UML Serveur


CarteBancaire
-numeroCarte : String
-dateExpiration : YearMonth
-cryptogramme : int
+getNumeroCarte() : String
+setNumeroCarte(String) : void
+getDateExpiration() : YearMonth
+setDateExpiration(String) : void
+getCryptogramme() : int
+setCryptogramme(int) : void
+yearToStringRes(YearMonth) : String
+toString() : String
Reservation
-dateArrivee : LocalDate
-dateDepart: LocalDate
-nbLitsResa: int
-client : Client
+getDateArrivee() : LocalDate
+setDateArrivee(String) : void
+getDateDepart() : LocalDate
+setDateDepart(String) : void
+getClient() : Client
+setClient(Client) : void
+dateToStringRes(LocalDate) : String
+toString() : String
ENUMERATION
Villes
PARIS
MARSEILLE
LYON
TOULOUSE
NICE
NANTES
MONTPELLIER
STRASBOURG
BORDEAUX
LILLE
AUTRE_VILLE
Client
-nom : String
-prenom : String
+getNom() : String
+setNom(String) : void
+getPrenom() : String
+setPrenom(String) : void
+getListReservationClient() : ArrayList<Reservation>
+setListReservationClient(ArrayList<Reservation>) : void
+getCarteBancaire() : CarteBancaire
+setCarteBancaire(CarteBancaire) : void
+setResaInListResaClient(Reservation) : void
Chambre
-identifiant : int
-numero : int
-nbLits : int
-hotel : Hotel
-prixChambre : Float
-picOfChambre : byte[ ]
+getIdentifiant : int
+setIdentifiant(int) : void
+getNumero() : int
+setNumero(int) : void
+getNbLits() : int
+setNbLits(int) : void
+getPrixChambre() : float
+setPrixChambre(float) : void
+getHotel() : Hotel
+setHotel(Hotel) : void
+getListReservationsChambre() : ArrayList<Reservation>
+setListReservationsChambre(ArrayList<Reservation>) : void
+getPicOfChamber() : byte [ ]
+setPicOfChamberBytesCode(bytes [ ]) : void
+setPicOfChamberByPathPicture(String) : void
+toString() : String
infoAgence
-identifiantAgence : int
-login : String
-password : String
-coeffPromotion : float
+getIdentifiantAgence() : int
+setIdentifiantAgence(int) : void
+getLogin() : String
+setLogin(String) : void
+getPassword() : String
+setPassword(String) : void
+getCoeffPromotion() : float
+setCoeffPromotion(float) : void
+toString() : String
Adresse
-pays : String
-numRue : int
-rue : String
-lieudit : String
-gpxPosition : String
+getPays() : String
+getPays(String) : void
+getGpxPosition() : String
+setGpxPosition(String) : void
+getNumRue() : int
+setNumRue(int) : void
+getRue() : String
+setRue(String) : void
+getVille() : Ville
+setVille(Ville) : void
+getLieudit() : String
+setLieudit(String) : void
+toString() : String
ENUMERATION
Etoiles
UN
DEUX
TROIS
QUATRE
CINQ
Offre
+dateTimeFormatter : DateTimeFormatter = DateTimeFormatter.ofPattern("dd-MM-yyyy")
-disponibilitee : ArrayList<String>
-hotelNom : String
-prixChambreWithReduction: float
+getDisponibilitee : ArrayList<String>
+setDisponibilitee(ArrayList<String>) : void
+getHotelNom() : String
+setHotelNom(String) : void
+getPrixChambreWithReduction() : float
+setPrixChambreWithReduction(float) : void
+setDisponibiliteeArrayList(ArrayList<Reservation>) : void
+toString() : String
Hotel
-nom: String
-adresse : adresse
-etoiles : Etoiles
-listChambre : ArrayList<Chambre>
-listPartenaire: ArrayList<infoAgence>
+getNom() : String
+setNom(String) : void
+getEtoiles() : Etoiles
+setEtoiles(Etoiles) : void
+getAdresse() : Adresse
+setAdresse(Adresse) : void
+getListChambre() : ArrayList<Chambre>
+setListChambre(ArrayList<Chambre>) : void
+setChambreInList(Chambre) : void
+getListAgencePartenaire() : ArrayList<infoA
+setListAgencePartenaire(ArrayList<infoAger
+toString() : String
HotelRepository
-hotelsToBind : Hotel
#DATE_TIME_FORMATTER : DateTimeFormatter
#isAvailable : boolean
+getNomHotel() : String
+getAddressHotel() : String
+getPays() : String
+getIdentifiantOffre() : ArrayList<Integer>
+consultationDisponibilite(int, String, String, String, String, Ville, Etoiles, float, float) : ArrayList<Offre>
+doReservation(int, String, String, int, String, String String, int, String, String, String, int) : String
WebService1ConsultationImpl
-hotel : HotelRepository
+getNomHotel() : String
+getAddressHotel() : String
+getPays() : String
+consultationDisponibilite(int, String, String, String, String, Ville, Etoiles, float, float) : ArrayList<Offre>
Server
- ArrayList<HotelRepository> hotelsToBind
- PathForChamberPicture : String = "/src/main/java/Image_Chambre/"
- DATE_TIME_FORMATTER : DateTimeFormatter = DateTimeFormatter.ofPattern("dd-MM-yyyy")
+main(String [ ] args) : void
-datainitialise() : void
WebService2ReservationImpl
-hotel : HotelRepository
+getIdentifiantOffre() : ArrayList<Integer>
+getNomHotel() : String
+doReservation(int, String, String, int, String, String String, int, String, String, String, int) : String
«interface»
IWebService1
+getNomHotel() : String
+getAddressHotel() : String
+getAddressHotel() : String
+consultationDisponibilite(int, String, String, String, int, String, Ville, Etoiles, float, float) : ArrayList<Offre>
«interface»
IWebService2
+getIdentifiantOffre() : ArrayList<Integer>
+getNomHotel() : String
+doReservation(int, String, String, int, String, String String, int, String, String, String, int) : String

Serveur :

Singleton :

Après avoir discuté de la problématique de soap le lundi 7 novembre avec Monsieur SERIAI, il nous a dit que comme il nous accordé une simplification plus simple nous pourrions implémenter un singleton pour la gestion de l'HotelRepository.

HotelRepository et notre class patron qui contient les différents algorithmes du webService afin de consulter les chambres et de réserver un hôtel.

Avant d'implémenter le singleton, lorsque nous instancions les implémentations de la consultation deux hotels étaient créés ce qui n'était pas le résultat attendu.

```
System.out.println("test fini");
Endpoint.publish("http://localhost:8080/HotelWebService1Consultation", new WebService1ConsultationImpl());
Endpoint.publish("http://localhost:8080/HotelWebService2Reservation", new WebService2ReservationImpl());
System.out.println("hotel 1 ready");
```

De ce fait nous avons modifié notre class HotelRepository afin qu'elle implémente le design pattern Singleton. Pour éviter un maximum d'erreur nous avons choisi la solution double check-login avec un accès à une version volatile, cependant après recherche lors de l'écriture du rapport il s'avère que le singleton volatile rencontre beaucoup de problèmes pour le multi-threading nous aurions du opter pour un autre…

```
21
22  public final class SingletonHotelRepository  {
23
24          private static volatile SingletonHotelRepository instance ;
25
26      public String value;
27      public static Hotel hotelsToBind;
28
29          private static final String PathForChamberPicture = "./src/main/java/Image_Chambre/";
30          protected static final DateTimeFormatter DATE_TIME_FORMATTER = DateTimeFormatter.ofPattern("dd-MM-yyyy");
31          protected static boolean isAvailable;
32
33
34
35      private SingletonHotelRepository(String value) throws IOException {
36          this.value = value;
37          initialiseDataBaseEnDur();
38                  //Test Singleton par instanciation des Chambres resultat attendu identifiant 1,2,3 et non 1,2,3 ... 4,5,6
39                  for (Chambre chambre : hotelsToBind.getListChambres()) {System.out.println(chambre.getIdentifiant());}
40      }
41
42      //Singleton
43      public static SingletonHotelRepository getInstance(String value)  {
44          SingletonHotelRepository result = instance;
45          if (result != null) {
46              return result;
47          }
48          synchronized(Singleton.class)  {
49              if (instance == null) {
50                  try {
51                                          instance = new SingletonHotelRepository(value);
52                                  } catch (IOException e) {
53                                          e.printStackTrace();
54                                  }
55              }
56              return instance;
57          }
58      }
```

WebService 1 Consultation de Chambre

Lors de la mise en distribution de notre programme interne nous avons découvert le problème de compatibilité d'envoi de HashMap via SOAP. Afin de transformer notre algorithme rechercheChambre (cf page 3) en Consultation nous avons crée une classe intermédiaire Offre qui hérite des informations nécessaires à connaitre de la classe chambre, informations qui ne sont que de types simples

Super () :

- l'identifiant de la chambre : int
- le numéro de la chambre : int
- le nombre de lits disponibles : int
- le bytes code de l'image de la chambre : bytes

Mais contient aussi des informations complémentaires à envoyer

- Le nom de l'hotel : String
- La disponibilité de la chambre : ArrayList<String>
- Le tarif de la chambre après la réduction de l'agence : float

La première partie de l'algorithme consultation vient tester l'existence de l'agence dans l'arraylist des partenaires de l'hotel grâce à un filtrage par la méthode stream.
Deux cas sont possibles : Soit l'agence est connu dans ce cas on vient récupérer son coefficient de promotion, Soit elle n'est pas connu dans ce cas la variable coeffPromotion reste à 1 afin de garder le prix de base des chambres

```
public ArrayList<Offre> consultationDisponiblite(int identifiantAgence, String loginAgence, String dateDebut,
        String dateFin, int nbPersonne, String pays, Ville ville, Etoiles etoiles, float prixMin, float prixMax) {


    Reservation reservation = new Reservation(dateDebut, dateFin, nbPersonne);
    ArrayList<Offre> OffreDeChambreDispo  = new ArrayList<>();

    if (hotelsToBind.getAdresse().getPays().equals(pays) && hotelsToBind.getAdresse().getVille().equals(ville) && hotelsToBind.getEtoiles().equals(etoiles)) {

        float coeffPromotion = 1;

        //Test de la réduction
        Optional<infoAgence> filteredAgence = hotelsToBind.getListAgencesPartenaire().stream()
                .filter(agence -> agence.getIdentifiantAgence()== identifiantAgence
                        && agence.getPassword().equals(loginAgence))
                .findFirst();
        System.out.println(filteredAgence.get().getPassword());
        if (filteredAgence.isPresent()) {
            System.out.printf("Super une réduction de %.2f pourcent ! \n", ((1-filteredAgence.get().getCoeffPromotion())*100));
            coeffPromotion = filteredAgence.get().getCoeffPromotion();
        }else {
            System.out.println("Pas de promotion !, on ne connait pas cette agence");

        }
```

La deuxième partie de l'algorithme fonctionne comme rechercheChambre qui a été expliqué plus haut lors de la version non distribuée, je vous invite à revenir plus haut si vous n'avez pas compris.
Ici ce que l'on vient juste changer c'est qu'à la place d'insérer dans la hashmap la chambre disponible on instancie un nouvel objet offre.
Celui-ci étant directement ajouté dans la liste des offres qui va être retourné par notre WebService.

```
        for (Chambre chambre : hotelsToBind.getListChambres()) {
            System.out.println(chambre.getNumero());
            if (chambre.getNbLits() >= nbPersonne && (prixMin <= chambre.getPrixChambre()*coeffPromotion && chambre.getPrixChambre() <= prixMax * coeffPromotion )) {
                if (chambre.getListReservationsChambre().isEmpty()) {
                    System.out.println(chambre + "add directement car vide" );
                    Offre offreEmpty = new Offre(filteredAgence.get().getCoeffPromotion(), chambre);
                    OffreDeChambreDispo.add(offreEmpty);
                }else {
                    for (Reservation reservationAlreadyBind : chambre.getListReservationsChambre()) {
                        System.out.println(chambre + "test si pas vide");
                        if ( ! (reservation.getDateDepart().isBefore(reservationAlreadyBind.getDateArrivee())
                                || reservationAlreadyBind.getDateDepart().isBefore(reservation.getDateArrivee()))){
                            System.out.println(chambre.toString()+ " est prise au date suivante arrivée : "+reservation.getDateArrivee()+ " , départ "+reservation.getDateDepart());
                            System.out.println(chambre.toString() + isAvailable);
                        }else {
                            System.err.println(chambre.toString()+" ne passe pas dans le test");
                            Offre offreEmpty = new Offre(filteredAgence.get().getCoeffPromotion(), chambre);
                            OffreDeChambreDispo.add(offreEmpty);
                        }
                    }
                }
            }
        }
        System.out.println(OffreDeChambreDispo.toString());
        return OffreDeChambreDispo;
    }
```

WebService 2 : HotelRepository public doReservation

Dans le WebService2 afin de trouver si l'agence est intégrée dans la base de donnée (crée en brut) de l'hôtel nous utilisons la même fonction stream filtrant la liste d'agence partenaire comme lors du 1er webService (explication ci-dessus).

```
public String doReservation(int identifiantAgence, String loginAgence, String passwordAgence, int idOffre, String prenomClient, String nomClient, String numeroCarte,
        int cryptogramme, String cardDateExpiration , String dateDebut, String dateFin, int nbLits) {

    Reservation reservation = new Reservation(dateDebut, dateFin, nbLits);
    String response =  "Erreur il n'y a aucune offre qui correspond avec l'identifiant de réservation renseigner. \n";
    String testAgence =  "";
```

Ensuite afin de garder le concept de SOAP qui est l'envoi de type simple et non d'objet, nous renvoyons avec le WebService2 un string response qui par défaut ressort qu'il n'y a pas d'offre disponible à l'identifiant renseigner.

L'utilisateur ayant entrée un identifiant d'offre on va opérer une boucle dans chacunes des chambres des hôtels. Si une des chambres corresponds à l'identifiant renseigné alors deux cas de figure peuvent être opéré. Nota Bene : le test de l'identifiant implique que l'identifiant de la chambre soit enregistré comme une clé primaire. Il ne faut pas que deux hôtels différents et le même identifiant de chambre.

Premier cas ligne 134 la liste des réservations de la chambre n'est pas vide elle peut être réservé si :

- la date de départ de réservation de l'utilisateur (Util.départ) a lieu avant la date d'arrivée d'une
- réservation de chambre (Résa.arrivée)
- ou si la date d'arrivée d'un utilisateur (Util.arrivée) a lieu après la date de départ d'une réservation de chambre (Résa.départ)

Donc, à l'inverse, si cela ne correspond pas à ces critères, la chambre ne pourra pas être réservé.

```
if (! chambre.getListReservationsChambre().isEmpty()) {
    isAvailable = true;
    for (Reservation reservationAlreadyBind : chambre.getListReservationsChambre()) {
        System.out.println(reservationAlreadyBind);
        System.out.println(reservationAlreadyBind.toString());
        if ( ! (reservation.getDateDepart().isBefore(reservationAlreadyBind.getDateArrivee())
                || reservationAlreadyBind.getDateDepart().isBefore(reservation.getDateArrivee()))) {
            isAvailable = false;
        }
    }
    if (isAvailable) {
        addReservationChambre(chambre, reservation, numeroCarte, cardDateExpiration, cryptogramme, nomClient, prenomClient);
        response = "Merci d'avoir réservé la chambre n°"+chambre.getNumero()+" pour les dates suivantes : "+reservation.getDateArrivee().format(DATE_TIME_FORMATTER)+" - "
                +reservation.getDateDepart().format(DATE_TIME_FORMATTER)+ " elle correspond à l'identifiant d'offre "+chambre.getIdentifiant()+"\n";
        System.out.println("isAvailable test boucle \n");
        }
```

Deuxième cas de figure else ligne 148 la liste est vide de réservation est vide donc on ajoute directement la reservation et on appelle notre to String personnalisée comme indiqué dans le if (isAvailable) ci-dessus.

Comme nous utilisions deux fois la méthode réservation dans l'algorithme nous avons repris celle de la version non distribuée et l'avons isolée :

```
private static void addReservationChambre (Chambre chambre, Reservation reservation, String numeroCarte, String carteDateExpiration,
                                           int cryptogramme, String nomClient, String prenomClient)
{
    CarteBancaire cb = new CarteBancaire(numeroCarte, carteDateExpiration, cryptogramme);
    Client client = new Client(nomClient, prenomClient, cb);
    reservation.setClient(client);
    chambre.getListReservationsChambre().add(reservation);
    System.out.println("j'ai fait la "+reservation);
}
```

Question 3 : Transfert d'image partie Serveur

Pour que le client puisse recevoir une image envoyée par le Serveur il est nécessaire d'envoyer du bytesCode, qui est le format compressé en langage binaire de l'image. Pour cela nous utilisons un tableau de bytes.
Tout d'abord nous récupérons le fichier grâce au package File, puis nous faisons lire le ficher par le serveur qui va ensuite le convertir en BytesCode grâce à la fonction ByteArrayOutputStream et toByteArray.

```java
import java.awt.image.BufferedImage;

public class Chambre {

    private int  identifiant;
    private int numero;
    private int nbLits;
    private Hotel hotel;
    private float prixChambre;
    private byte[] picOfChamber;
    private ArrayList<Reservation> listReservationsChambre;


public void setPicOfChamberByPathPicture(String pathPicure) throws IOException {
    File fichier = new File(pathPicure);
    BufferedImage fileToByteCode = ImageIO.read(fichier);
    ByteArrayOutputStream ByteForXML = new ByteArrayOutputStream();
    ImageIO.write(fileToByteCode, "jpg", ByteForXML);
    this.picOfChamber = ByteForXML.toByteArray();
}
```

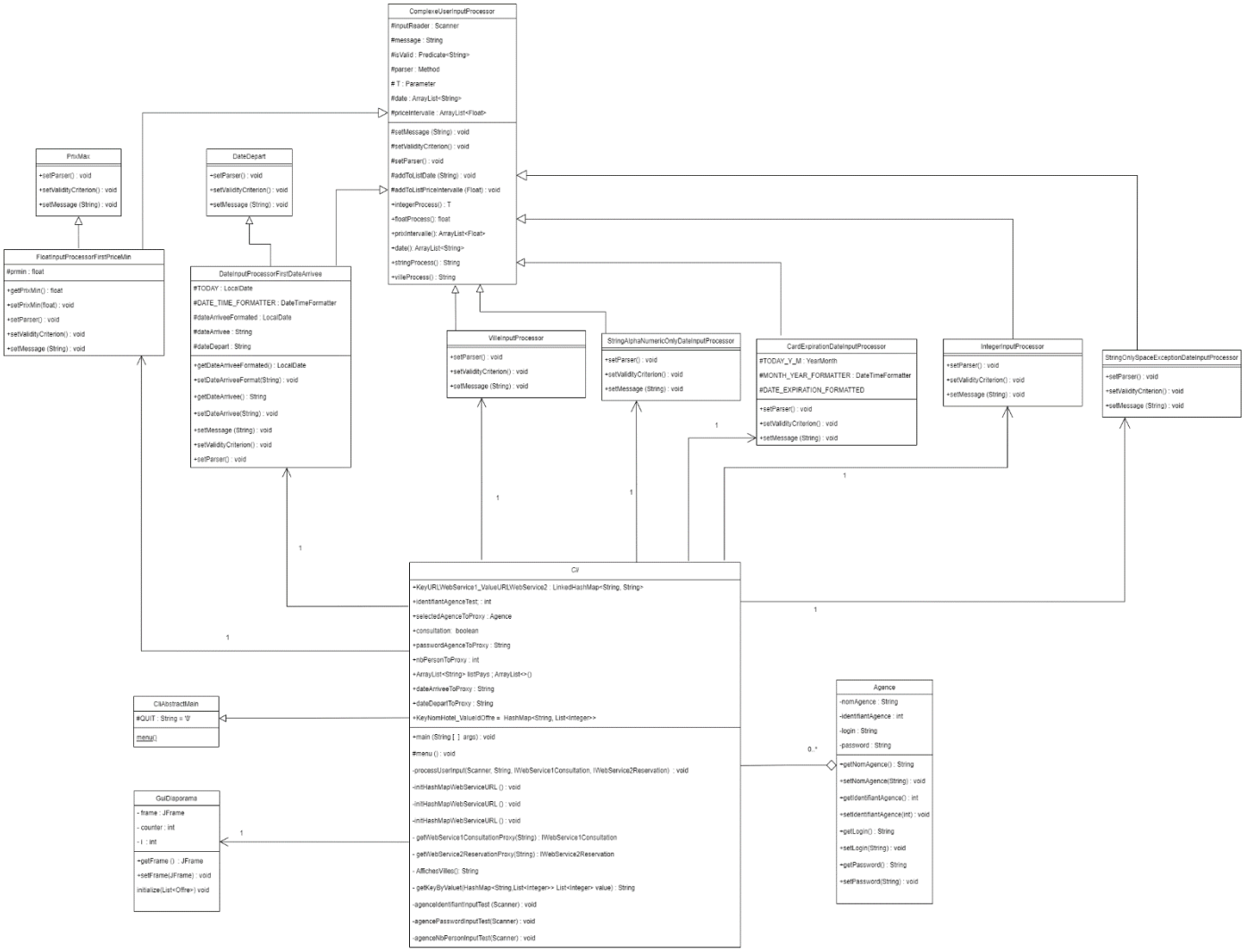
```java
public class Offre extends Chambre{

   public static final DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("dd-MM-yyyy");
   private ArrayList<String> disponibilitee = new ArrayList<>();
   private String hotelNom;
   private float prixChambreWithReduction;

   public Offre(float coeffPromotion, Chambre chambre) {
       super(chambre.getIdentifiant(), chambre.getNumero(), chambre.getNbLits(), chambre.getPicOfChamberBytesCode());
       setHotelNom(chambre.getHotel().getNom());
       setDisponibiliteeWithArrayList(chambre.getListReservationsChambre());
       setPrixChambreWithReduction(chambre.getPrixChambre()*coeffPromotion);
   }
```

Client

UML :



Package Input Processor et CLI

Afin de simplifier les différents input dans le cli nous avons gérer les différentes exceptions grâce au modèles que vous avez présentez dans le workflow du TP SOAP, nous l'avons simplement modifier en ajoutant différent module de retour en fonction du type que nous souhaitions rechercher et avons modifier la fonction setMessage afin que lorsqu'elle soit appelée dans le cli nous pouvons modifier le message qui va être répétée.

De plus lorsque deux données en input étaient en corrélation nous avons crée une ArrayList pour qu'elle soit renvoyée dans le cli (pour les dates et le prix).
Afin que la date départ et connaissance de la date arrivée nous avons utiliser le schéma de class interne qui permet de créer une classe dans une classe principale, cette classe interne ne pouvant être instanciée que dans la classe contenante afin que la classe interne est connaissance des attributs de la classe contenant.

Un exemple de code est montré page 14 pour la DateInputProcessor.

```
public abstract class ComplexeUserInputProcessor<T> {

	/* ATTRIBUTES */
	protected Scanner inputReader;
	protected String message;
	protected Predicate<String> isValid;
	protected Method parser;
	protected T parameter;

	//Optimisation possible en précisant que c'est une liste de deux éléments
	protected ArrayList<String> date = new ArrayList<>();
	protected ArrayList<Float> priceIntervalle = new ArrayList<>();

	/* CONSTRUCTOR */
	public ComplexeUserInputProcessor(Scanner inputReader, String message) {
		this.inputReader = inputReader;
		setMessage(message);
		setValidityCriterion();
		setParser();
	}

	/* METHODS */

	protected abstract void setMessage(String message);
	protected abstract void setValidityCriterion();
	protected abstract void setParser();
	protected void addToListDate(String addDate) {date.add(addDate);}
	protected void addToListPriceIntervalle(Float priceAdd){priceIntervalle.add(priceAdd);}


/* IntegerProcess Retourne un int dans le cli */

	public T integerProcess() throws IOException {
		System.out.println(message);
		String userInput = inputReader.nextLine();
		while (!isValid.test(userInput)) {
			System.err.println("Désolé, votre précédente donnée entrée est fausse,
			veuillez réessayer.");
			System.out.println();
			System.out.println(message);
			userInput = inputReader.nextLine();
		}
		try {
			parameter = (T) parser.invoke(null, userInput);
		} catch (SecurityException | IllegalAccessException |
				IllegalArgumentException | InvocationTargetException e) {
			e.printStackTrace();
		}
		return parameter;
	}

	/* FloatProcess */
	public Float floatProcess() throws IOException {
		.............
		return Float.parseFloat(userInput);
	}

	/* ArrayList Intervalle Prix*/

	public ArrayList<Float> prixIntervalle() throws IOException {
		.............
		return priceIntervalle;
	}

	/*ArrayList Date Test*/

	public ArrayList<String> date() throws IOException {
		.....................
		return date;
	}
```

Date Input Processor : Class interne

```
public class DateInputProcessorFirstDateArrivee extends ComplexeUserInputProcessor<String> {
	/*Attribut*/
	protected static final LocalDate TODAY = LocalDate.now();
	protected static final DateTimeFormatter DATE_TIME_FORMATTER =
		DateTimeFormatter.ofPattern("dd-MM-yyyy");
	protected LocalDate dateArriveeFormated = null;
	protected String dateArrivee = "null";
	protected String dateDepart = "null";


/*Classe interne DateDepart*/
class DateDepart extends DateInputProcessorFirstDateArrivee{

	public DateDepart(Scanner inputReader, String message) {
		super(inputReader, message);
	}


	@Override
	protected void setMessage(String message) {
		this.message = message;
	}


	@Override
	protected void setValidityCriterion() {
		isValid = str -> {
			try {
				String valueOfDateDepart = str;
				LocalDate dateDepartFormated= LocalDate.parse(valueOfDateDepart,
				DATE_TIME_FORMATTER);
				if (dateDepartFormated.getYear() > TODAY.getYear()+1) {
					System.err.println("Désolé mais il n'est pas possible de
					réserver au delà de l'année "+(TODAY.getYear()+1));
					return false;
				}
				if(DateInputProcessorFirstDateArrivee.this.dateArriveeFormated.
				isAfter(dateDepartFormated)) {
					System.err.println("Entrer une date sous la forme jj-mm-
					yyyy, date qui ne peut être"+ "inférieur à la date que
					vous avez entrée pour votre arrivée"
					+DateInputProcessorFirstDateArrivee.this.dateArrivee);
					return false;
				}else {
					return true;
				}
			} catch (DateTimeParseException e) {
				System.err.println("Entrer une date sous la forme jj-mm-yyyy,
				date qui ne peut être"+" inférieur à votre date d'arrivee
				"+DateInputProcessorFirstDateArrivee.this.dateArrivee);
				return false;
			}
		};
	}
```

```
@Override
protected void setValidityCriterion() {
        isValid = str -> {
                try {
                        String valueOfDateArrivee = str;
                        setDateArriveeFormat(valueOfDateArrivee);
                        if (dateArriveeFormated.getYear() > TODAY.getYear()+1) {
                                System.err.println("Désolé mais il n'est pas possible de
                                réserver au delà de l'année."+(TODAY.getYear()+1));
                                return false;
                        }

                        //|| TODAY.equals(dateArriveeFormated)
                        if (TODAY.isAfter(dateArriveeFormated) ) {
                                System.err.println("Merci de renseigner une date d'arrivée qui
                                se situe après le " + TODAY.format(DATE_TIME_FORMATTER) + " date
                                d'aujourd'hui.");
                                return false;
                        }else {

                        //NE PAS ToUCHER PERMET DE SET LA VARIABLE GLOBAL
                                setDateArrivee(valueOfDateArrivee);
                                addToListDate(valueOfDateArrivee);
                                DateDepart dateDepart = new DateDepart(inputReader,
                                "Veuillez entrer une date de départ (jj-mm-yyyy) qui se situe
                                après votre date d'arrivée le : "+getDateArrivee());
                                addToListDate(dateDepart.stringProcess());
                                return true;

                                //A garder au cas où si cela ne fonctionne pas du côté serveur
//                              return valueOfDateArrivee instanceof String;
                                }
                     } catch (DateTimeParseException | IOException  e) {
                                System.err.println("Entrer une date sous la forme jj-mm-yyyy,
                                date qui ne peut être"+" inférieur à la date d'aujourd'hui
                                "+TODAY.format(DATE_TIME_FORMATTER));
                                return false;
                        }

                };
        }
```

SetValidityCriterion pour input principal :

```
@Override
protected void setValidityCriterion() {
        isValid = str -> {
                try {
                     Float prixMin = Float.parseFloat(str);
                     if (prixMin < 0){
                                return false;
                     }else {
                             setPrixMin(prixMin);
                             addToListPriceIntervalle(prixMin);
                             FloatPriceMax prixMax = new FloatPriceMax(inputReader, "Veuillez
                               indiquer le prix maximum pour un intervalle de prix : \n"
                               +"sachant que vous avez indiqué le prix minimum est de "
                               +getPrixMin());
                             addToListPriceIntervalle(prixMax.floatProcess());
                             return true;
                            }
                     } catch (NumberFormatException | IOException e) {
                                System.err.println("Merci de bien vouloir entrée un nombre entier et
                                non un "+ "caractère spécial ou des lettres");
                                return false;
                     }
                };
        }
```

GUI et lecture du bytecode

Enfin afin de lire le bytesCode côté Client nous avons implémentée une GUI à l'aide du package JFrame, Lorsque que le GUI est instancié il va stocker deux variables tableaux d'une part les différents bytesCodes et d'autres part dans l'autre variables informations correspondant à la chambre grâce au paramètre (listeOffre) qui lui a été passée. Il est ensuite possible de faire défiler les différentes éléments des tableaux par un mécanisme de compteur qui s'additionne (bouton suivant) ou se soustrait (bouton précédent) de 1 lorsque l'on clique sur un bouton du GUI

```
public class GuiDiaporama {

	private void initialize(List<Offre> listOffre) {
		frame = new JFrame();
		frame.setBounds(100, 100, 1240, 720);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.getContentPane().setLayout(null);


		//List stockant les bytes codes des différentes images
		ImageIcon picture[] = new ImageIcon[listOffre.size()];

		//List stockant les strings à afficher pour chaque image
		String stringPrint[] = new String[listOffre.size()];

		//String.format("%2.f", offre.getPrixChambreWithReduction())

		for (Offre offre : listOffre) {
			ImageIcon icon = new ImageIcon(offre.getPicOfChamberBytesCode());
			picture[i] = icon;
			String toPrint = (offre.getHotelNom()+" Chambre n°"+offre.getNumero()
			+"\n" +"Prix Chambre : "+offre.getPrixChambreWithReduction()+" €\n"
			+"Identifiant de l'offre : "+offre.getIdentifiant()+"\n"
			+"disponibilité : \n") ;
			for (int i = 0; i < offre.getDisponibilitee().size(); i++){
				toPrint += offre.getDisponibilitee().get(i)+" - ";
				toPrint += offre.getDisponibilitee().get(i+=1)+"\n";
			}
			stringPrint[i] = toPrint;
			i++;
		}


	JButton btnPrecedent = new JButton("Précedent");
	btnPrecedent.setFont(new Font("Tahoma", Font.PLAIN, 35));
	btnPrecedent.addMouseListener(new MouseAdapter() {
		@Override
		public void mouseClicked(MouseEvent e) {
			if (counter!=0) {
				counter -=1;
				Picslider.setIcon(picture[counter]);
				textPanel.setText(stringPrint[counter]);
			}
		}
	});
	btnPrecedent.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
			if(e.getSource()==btnPrecedent) {
				if (counter==0) {
					JOptionPane.showMessageDialog(null,"Ceci est la
					première image");
				}
			}
		}
	});

	btnPrecedent.setBounds(59, 513, 210, 85);
	frame.getContentPane().add(btnPrecedent);

	JButton btnSuivant = new JButton("Suivant");
	btnSuivant.setFont(new Font("Tahoma", Font.PLAIN, 35));
	btnSuivant.addMouseListener(new MouseAdapter() {
		@Override
		public void mouseClicked(MouseEvent e) {
			if (counter != picture.length) {
				counter +=1;
				Picslider.setIcon(picture[counter]);
				textPanel.setText(stringPrint[counter]);
			}
		}
	});
	btnSuivant.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
			if(e.getSource()==btnSuivant) {
				if (counter==picture.length-1) {
					JOptionPane.showMessageDialog(null,"Ceci est la
					dernière image");
				}
			}
		}
	});
	btnSuivant.setBounds(926, 513, 210, 85);
	frame.getContentPane().add(btnSuivant);
```

Version avec PLUSIEURS HOTELS ET AGENCES : VERSION FINALE

Serveur

Repository

Afin de créer plusieurs hotels nous supprimons simplement le singleton et créons un paramètre hotel dans le constructeur de la classe HotelRepository.
Ensuite dans la classe serveur (main) nous codons une ArrayList d'HotelRepository, puis nous instancions différents Hotel et HotelRepository dans la fonction datainitiliase.

Chaque élément de l'arrayList HotelRepository est affecté à un port différent avec qu'il n'y est pas de conflit.

```
public static void main(String[] args) throws IOException {

                        datainitialise();

                        Endpoint.publish("http://localhost:8888/HotelWebService1Consultation", new
WebService1ConsultationImpl(hotelsToBind.get(0)));
                        Endpoint.publish("http://localhost:8888/HotelWebService2Reservation", new
WebService2ReservationImpl(hotelsToBind.get(0)));
                        System.out.println("hotel 1 ready");
                        Endpoint.publish("http://localhost:4444/HotelWebService1Consultation", new
WebService1ConsultationImpl(hotelsToBind.get(1)));
                        Endpoint.publish("http://localhost:4444/HotelWebService2Reservation", new
WebService2ReservationImpl(hotelsToBind.get(1)));
                        System.out.println("hotel 2 ready");

        }

        private static void datainitialise() throws IOException {
                        Client Dupond = new Client("Dupond", "Robert");
                        Client Raïssanna = new Client("Slimane", "Raïssanna");


                        Hotel daysHotel = new Hotel("DaysHotel", Etoiles.DEUX);
                        daysHotel.setAdresse("France",10, "Route du Midi", Ville.MONTPELLIER);

                        daysHotel.getListChambres().addAll(Arrays.asList(
                                        new Chambre(1, 1, 2, 90, PathForChamberPicture+"pic1.jpg", daysHotel),
                                        new Chambre(2, 2, 2, 77,  PathForChamberPicture+"pic2.jpg", daysHotel),
                                        new Chambre(3, 3, 4, 90, PathForChamberPicture+"pic3.jpg", daysHotel)
                                        ));

                        daysHotel.getListChambres().get(0).getListReservationsChambre().addAll(Arrays.asList(
                                        new Reservation(LocalDate.now().format(DATE_TIME_FORMATTER),(
LocalDate.now().plusDays(1)).format(DATE_TIME_FORMATTER), 2, Dupond)));

                        daysHotel.getListChambres().get(1).getListReservationsChambre().addAll(Arrays.asList(new Reservation("05-01-2023",
"08-01-2023", 2, Raïssanna)));

                        //Instantiation d'objet InfoAgence dans la liste des Agences partenaires
                        daysHotel.getListAgencesPartenaire().addAll(Arrays.asList(new infoAgence(1, "sel_log", "selectour", 10),

                                        new infoAgence(2, "trip_log", "tripavis", 20),

                                new infoAgence(3, "via_log", "viahotel", 15)));

                        Hotel laPaix = new Hotel("LaPaix", Etoiles.DEUX);
                        laPaix.setAdresse("France",20, "Rue de la Paix", Ville.MONTPELLIER);

                        laPaix.getListChambres().addAll(Arrays.asList(
                                        new Chambre(4, 10, 5, 225, PathForChamberPicture+"pic4.jpg", laPaix),
                                        new Chambre(5, 20, 2, 140,  PathForChamberPicture+"pic5.jpg", laPaix),
                                        new Chambre(6, 3, 2, 160, PathForChamberPicture+"pic6.jpg", laPaix)
                                        ));

//
                        //Instantiation d'objet InfoAgence dans la liste des Agences partenaires
                        laPaix.getListAgencesPartenaire().addAll(Arrays.asList(new infoAgence(1, "sel_log", "selectour", 5),
```

```
                                        new infoAgence(2, "trip_log", "tripavis", 15),

                                    new infoAgence(3, "via_log", "viahotel", 25)));

                HotelRepository dayshotelRepo = new HotelRepository(daysHotel);
                HotelRepository lapaixRepo = new HotelRepository(laPaix);

                hotelsToBind.add(lapaixRepo);
                hotelsToBind.add(dayshotelRepo);
        }
```

Client

Du côté Client nous créons une nouvelle HashMap qui va contenir l'adresse du WebService1 en tant que clé et l'adresse du WebService2 en tant que valeur.

```
public static LinkedHashMap<String, String> KeyURLWebService1_ValueURLWebService2 = new LinkedHashMap<>();

IWebService1Consultation proxyConsultation = null;
IWebService2Reservation proxyReservation = null;
```

Il suffit dès lors d'ajouter les valeurs à la HashMap avec la fonction initHashMapWebServiceURL chaque put dans la HashMap étant effectué un hotel différent.

```
private static void initHashMapWebServiceURL () {
        KeyURLWebService1_ValueURLWebService2.put("http://localhost:8888/HotelWebService1Consultation",
        "http://localhost:8888/HotelWebService2Reservation");
        KeyURLWebService1_ValueURLWebService2.put("http://localhost:4444/HotelWebService1Consultation",
        "http://localhost:4444/HotelWebService2Reservation");
}
```

Puis chacun des proxys pourra être instancier à la volée en fonction des besoins des méthodes du CLI grâce aux deux fonctions suivantes :

```
private static IWebService1Consultation getWebService1ConsultationProxy(String urlWebService1) throws MalformedURLException {
        return new WebService1ConsultationImplService(new URL(urlWebService1)).getWebService1ConsultationImplPort();
}

private static IWebService2Reservation getWebService2ReservationProxy(String urlWebService2) throws MalformedURLException {
            return new WebService2ReservationImplService(new URL(urlWebService2)).getWebService2ReservationImplPort();
}
```

Nous codons en supplément de cela une petite fonction de récupération d'une clé de la HashMap en fonction de la valeur insérer pour les besoins du WebService2 afin que lorsque l'identifiant de réservation a été sélectionné le proxy ne renverra la demande de réservation que sur le serveur (l'hotel) sélectionné.

```
public static String getKeyByValue(HashMap<String,List<Integer>> keyNomHotel_ValueIdOffre2, List<Integer> value) {
            String toReturn = null;
            for (Entry<String, List<Integer>> entry : keyNomHotel_ValueIdOffre2.entrySet()) {
                if (Objects.equals(value, entry.getValue())) {
                    toReturn = entry.getKey();

                }
            }
            return toReturn;
}
```

Exemple Webservice 2 de la récupération de clé

```
for (String urlWebService2Value : KeyURLWebService1_ValueURLWebService2.values()) {
    System.out.println(urlWebService2Value);
    proxyReservation = getWebService2ReservationProxy(urlWebService2Value);
    System.out.println("Hotel : "+proxyReservation.getNomHotel());
    System.out.println(proxyReservation.listIdentifiantOffre().toString());
    KeyNomHotel_ValueIdOffre.put(urlWebService2Value, proxyReservation.listIdentifiantOffre());
    System.out.println(KeyNomHotel_ValueIdOffre.values());
}
```

```
do {
    identifiantOffreToProxy = integerInputProcessor.integerProcess();
    for (List<Integer> t : KeyNomHotel_ValueIdOffre.values()) {
                for (int i : t) {
                    if (t.contains(i)) {
                                        test = true;
                                        break;
            }
                    urlSelected = getKeyByValue(KeyNomHotel_ValueIdOffre, t);
                    System.out.println(urlSelected);
                            break;
                }
} while (test == false);
```