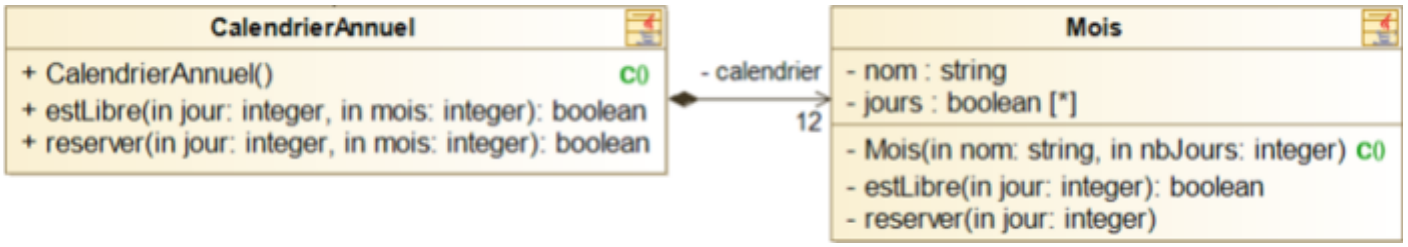
Projet commun

cas “réserver”

Entité : le calendrier (classe interne)

Ci-dessous la classe *CalendrierAnnuel*



Repérer sur le diagramme quelle est la classe englobante et quelle est la classe interne.

La classe Mois

Cette classe a un attribut *nom* et un attribut *jour* : un tableau de booléen dont la taille dépend du nombre de jours dans le mois.
Exemple d’utilisation : si le booléen de la cinquième case du tableau *jours* est à vrai, cela signifie que pour le 6ème jour du mois une réservation a été effectuée, s’il est à faux cela signifie qu’il n’y a aucune réservation ce jour-ci.

Les méthodes sont :

- *estLibre* permet de savoir s’il y a une réservation ce jour–là,
- *reserver* permet de réserver un jour dans le calendrier.

Dans le logiciel la méthode réserver est systématiquement appelée après s’être assurée par l’appel à la méthode *estLibre* qu’il n’y a pas de réservation ce jour-là.
Concrètement dans la méthode *reserver* si le jour à reservé n’est pas libre, jeter l’exception *IllegalStateException*.

La classe Calendrier

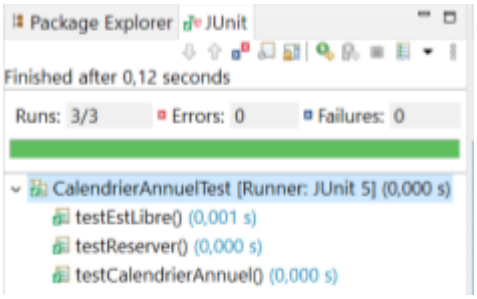
Le constructeur initialise le tableau *calendrier* avec les 12 mois de l’année.
Pour rappel, ci-dessous, le nombre de jour par mois :
Janvier : 31, Février : 28, Mars : 31, Avril 30, Mai : 31, Juin : 30, Juillet : 31, Août : 31, Septembre : 30, Octobre : 31, Novembre : 30, Décembre : 31.

Pour simplifier le TP nous ne gérerons pas les années bissextiles.

Les méthodes : *estLibre* et *reserver* retourne des booléens, la méthode *reserver* vérifie qu'il n'y a pas de réservation ce jour là avant de faire la réservation est retourne *true*, s'il y en a une elle retourne *false* pour préciser que la réservation n'a pas pu être effectuée.

Tester votre classe : cliquer droit sur la classe *CalendrierAnnuelTest* puis "Run As" et "JUnit Test".

Vous devez obtenir le résultat suivant :



Si vous n'avez pas dans "Runs" 3/3 c'est qu'une de vos méthodes ne fonctionne pas.

Entité : les réservations (héritage et abstraction)

Une réservation correspond à l'acceptation par le restaurateur (ou l'organisateur de spectacle...) de retenir une place dans son établissement. Une réservation est caractérisée par une date (jour / mois), mais aussi par d'autres informations spécifiques qui dépendent du contexte. Une réservation doit ainsi nécessairement se spécialiser : réservation de spectacle, de table...

Les classes à implémenter sont : *Reservation*, *ReservationHotel*, *ReservationRestaurant* et *ReservationSpectacle*.
Pour effectuer une réservation nous avons besoin, selon s'il s'agit d'une réservation :

- d'une table de restaurant : du jour et du mois de la réservation, du numéro du service (premier ou deuxième service) et du numéro de la table réservée,
- d'une chambre d'hôtel : du jour et du mois de la réservation, du nombre de lit simple, du nombre de lit double et du numéro de la chambre,
- d'une place de spectacle : du jour et du mois de la réservation, du numéro de la zone où se situe la place réservée et le numéro de la chaise réservée.

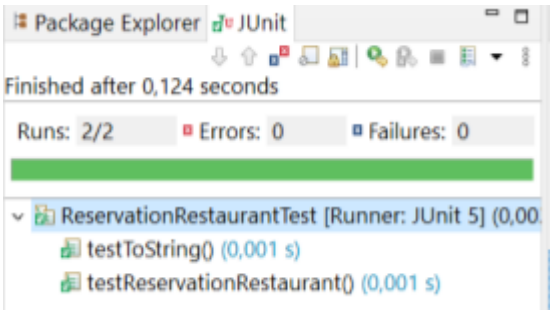
Tous les attributs sont initialisés à la création de la réservation.
Seules les reservations *ReservationHotel*, *ReservationRestaurant* et *ReservationSpectacle* seront utilisés, il n'est donc pas souhaitable d'instancier une *Reservation*.

Par la suite, nous aurons besoin de vérifier l'état des réservations. Ajouter les méthodes *toString()* nécessaires. Exemple d'affichage pour une réservation d'une table au restaurant :

```
Le 15/4 : table n°1 pour le deuxième service.
```

Tester votre classe : cliquer droit sur la classe *ReservationRestaurantTest* puis "Run As" et "JUnit Test".

Vous devez obtenir le résultat suivant :



Si vous n'avez pas dans "Runs" 2/2 c'est qu'une de vos méthodes ne fonctionne pas.

Entité : les entités réservables (héritage, abstraction et généricité)

Nous avons vu que les données nécessaires à une réservation dépendent du type d'entité à reserver (chambre, table ou place). Nous avons donc besoin d'avoir divers formulaire de réservation. Les formulaires correspondent à des demandes de réservation du client. Elles ont vocation à se traduire par une *Reservation* lorsque le propriétaire accepte la demande.
Ainsi, ces formulaires sont des entités qui **seront créées par le contrôleur** du cas "réserver" afin de stocker les informations entrées par l'utilisateur au niveau du boundary (console) ou de la présentation (IHM). Nous reprendrons ce découpage en partie 5 lorsque nous implémenterons le cas "réserver une table". Pour l'instant nous allons implémenter ces formulaires.

Les formulaires

Les classes à implémenter sont : *Formulaire*, *FormulaireHotel*, *FormulaireRestaurant* et *FormulaireSpectacle*.
Le formulaire doit pouvoir stocker, selon s'il s'agit d'un formulaire contenant les données :

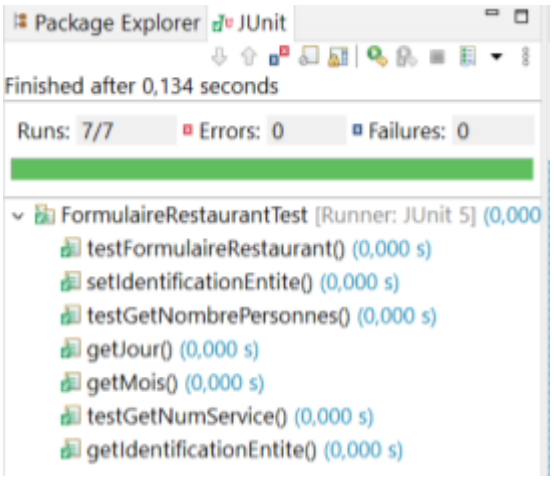
- d'une table de restaurant : du jour et du mois de la réservation, du numéro de l'entité réservé (exemple le numéro de la table), du nombre de personnes qui sont conviées, et du numéro du service.
- d'une chambre d'hôtel : du jour et du mois de la réservation, du numéro de l'entité réservé (exemple le numéro de la chambre), du nombre de lit simple et du nombre de lit double,
- d'une place de spectacle : du jour et du mois de la réservation, du numéro de l'entité réservé (exemple le numéro de la place de spectacle), du numéro de la zone où se situe la place réservée.

Tous les attributs, excepté le numéro de l'entité réservée) sont initialisés à la création de la réservation.
Seuls les formulaires *FormulaireHotel*, *FormulaireRestaurant* et *FormulaireSpectacle* seront utilisés il n'est donc pas souhaitable de pouvoir instancier un *Formulaire*.

Par la suite nous aurons besoin de tous les getteurs et un setteur sur le numéro de l'entité réservée.

Tester votre classe : cliquer droit sur la classe *FormulaireRestaurantTest* puis "Run As" et "JUnit Test".

Vous devez obtenir le résultat suivant :



Si vous n'avez pas dans "Runs" 7/7 c'est qu'une de vos méthodes ne fonctionne pas.

Une entité réservable

La classe à implémenter est : *EntiteReservable*.
Une entité réservable peut être une table, une chambre ou une place de spectacle.
Elle possède :

- un calendrier personnel (son carnet de réservation),
- un numéro (son identification).

Par la suite, nous aurons besoin d'un getteur et un setteur sur son numéro.

Les trois méthodes doivent prendre en paramètre d'entrée le formulaire de réservation correspondant à l'entité à réserver, mais attention :

- si l'entité est une table alors il faut utiliser le formulaire de réservation d'une table,
- si l'entité est une chambre d'hôtel alors il faut utiliser le formulaire de réservation d'une chambre,
- si l'entité est une table alors il faut utiliser le formulaire de réservation d'une table

Donc le type du formulaire dépend de l'entité, il est donc choisi au niveau de la création de l'objet.

Les méthodes :

- *estLibre*, retourne un booléen si l'entité est libre pour le jour et le mois indiqués dans le formulaire,
- *compatible*, permet de savoir si l'entité est réservable par rapport aux données du formulaire,
- *reserver*, permet de réserver l'entité par rapport aux données du formulaire : si l'état interne de l'entité est compatible avec le formulaire la méthode retourne une nouvelle *Reservation*, sinon elle retournera *null*.

Les deux dernières méthodes sont trop dépendantes de l'entité elle-même pour être implémentés ici, mais toutes les entités réservables possèdent ces méthodes.

Entité : la centrale de réservation (généricité)

La centrale de réservation est une centrale vendue soit à un restaurateur soit à un hôtelier soit à une organisation de spectacle. Donc quand on crée la centrale, on connaît le client cible. Les entités réservables sont donc soit uniquement des chambres, soit uniquement des tables, soit uniquement des places.

La centrale de réservation possède deux attributs :

- un tableau des entités à reservés (soit uniquement des chambres, soit uniquement des tables, soit uniquement des places)
- le nombre d'entités dans le tableau.

Le tableau des entités à réserver est donné en paramètre d'entrée du constructeur, il est vide.

Les trois méthodes sont :

- *ajouterEntite* : ajoute une entité passée en paramètre d'entrée dans le tableau, incrémente le numéro de l'entité et le retourne (ce nombre sera le numéro de l'entité),
- *donnerPossibilites* : prend en paramètre d'entrée le formulaire rempli par l'utilisateur et correspondant au formulaire associé au type de l'entité. Cette méthode retourne autant de valeurs entières que d'entités (le type de retour est un tableau d'entiers). Si une entité est disponible selon les critères précisés par le formulaire, la valeur stockée dans le tableau est égale au numéro d'entité. Si l'entité n'est pas disponible, la valeur est égale à 0.
- *reserver* : prend en paramètre d'entrée le numéro de l'entité à réserver et le formulaire rempli par l'utilisateur et correspondant au formulaire associé au type de l'entité. Cette méthode retourne un objet de type *Reservation*. Cette méthode récupère l'entité à réserver dans le tableau et place le numéro de cette entité dans le formulaire (le setteur du numéro dans le formulaire) et enfin réserve l'entité.

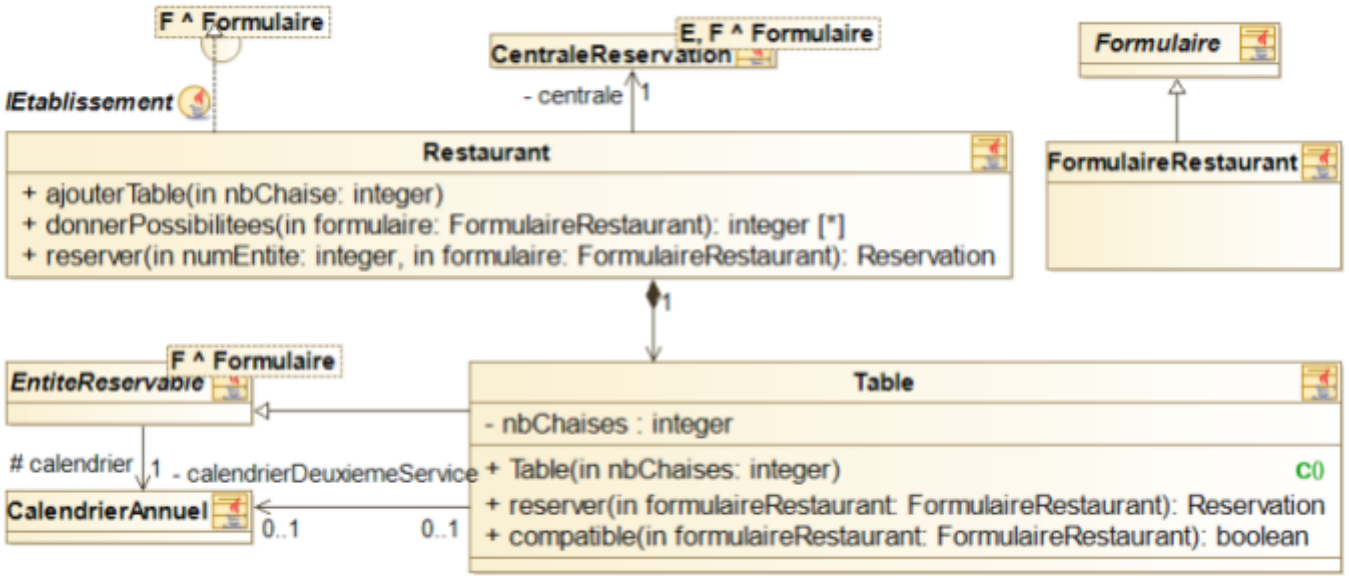
Entité : les établissements (classe interne, généricité)

Dans le sujet nous avons trois établissements possibles : le restaurant, l'hôtel et la salle de spectacle. Dans cette partie nous allons nous concentrer sur le restaurant, sachant que les autres établissements seront codé de manière équivalente.

Le restaurant

Les restaurants concernés n'ouvrent que le soir (il n'y a pas de service le midi), par contre le soir ils offrent deux services, le premier vers 19H00 et le second vers 21H00.

Le diagramme de classe ci-dessous donne une vue globale du restaurant dans son contexte.



Repérer la classe englobante et la classe interne.

Classe Table

Une table à la particularité de pouvoir être réservée 2 fois le même jour : pour le premier et pour le deuxième service. Contrairement aux autres entitées réservables il lui faut donc deux calendriers : l'attribut *calendrier* hérité par sa classe mère *EntiteReservable* et son attribut spécialisé *calendrierDeuxiemeService*.
Chacune des tables possède un nombre de chaises qui lui sont associées, leur nombre est donné au moment de la création de la table.

Pour savoir si une table est compatible par rapport à un formulaire de réservation, il faut vérifier :

- que la table possède le même nombre de chaises (ou une en plus) que de convives,
- que la table est libre à la date souhaitée et pour le service souhaité.

Pour réserver une table il faut réserver la date dans le calendrier correspondant au service demandé. Si la réservation dans le calendrier à abouti alors un objet de type *ReservationRestaurant* est créé et retourné.

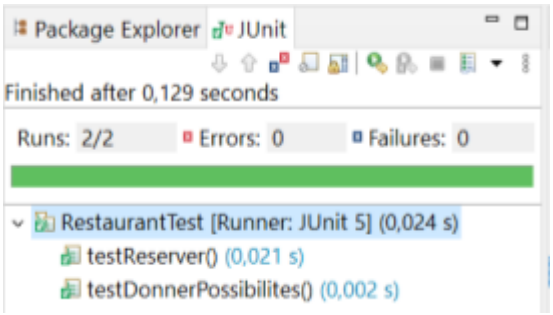
Classe Restaurant

Les méthodes sont :

- *ajouterTable* : créer une table et l'ajouter à la centrale de réservation, le numéro de l'entité est affecté comme le numéro de la table.
- *donnerPossibilite* & *reserver* appellent les méthodes de même nom dans la centrale de réservation.

Tester votre classe : cliquer droit sur la classe *RestaurantTest* puis "Run As" et "JUnit Test".

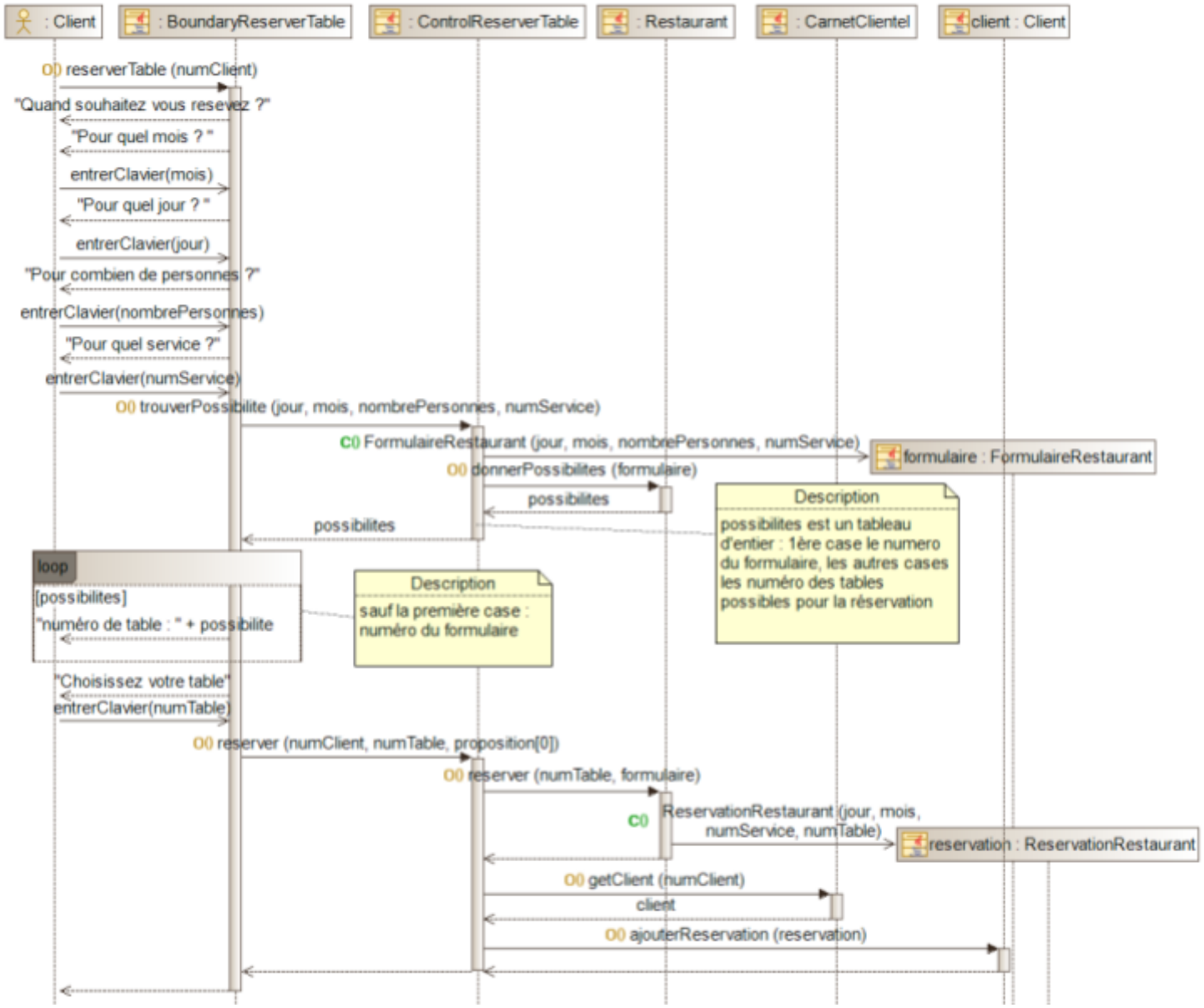
Vous devez obtenir le résultat suivant :



Si vous n'avez pas dans "Runs" 2/2 c'est qu'une de vos méthodes ne fonctionne pas.

Cas réserver table (console & IHM)

Toutes les entités sont implémentées, ci-dessous le diagramme de séquence détaillé permettant d'implémenter la frontière et le contrôleur.



Les méthodes du contrôleur vous sont données presque entièrement.

```
public int[] trouverPossibilite(int jour, int mois, int nombrePersonnes,
   int numService) {
   FormulaireRestaurant formulaire = //A COMPLETER;
   boolean formulaireEnregistrer = false;
   int numeroFormulaire = -1;

   for (int i = 0; i < formulaires.length && !formulaireEnregistrer; i++) {
      if (formulaires[i] == null) {
         formulaires[i] = formulaire;
         formulaireEnregistrer = true;
         numeroFormulaire = i;
      }
   }
   int[] possibilites = //A COMPLETER;

   int[] retour = new int[possibilites.length + 1];
   retour[0] = numeroFormulaire;
   for (int i = 1; i < possibilites.length + 1; i++) {
      retour[i] = possibilites[i - 1];
   }
   return retour;
}

public void reserver(int numClient, int numProposition, int numeroFormulaire) {
   FormulaireRestaurant formulaireRestaurant = formulaires[numeroFormulaire];
   formulaires[numeroFormulaire] = null;
   Reservation reservation
      = restaurant.reserver(numProposition, formulaireRestaurant);
   if (reservation instanceof ReservationRestaurant) {
      ReservationRestaurant reservationRestaurant
         = (ReservationRestaurant) reservation;
      Client client = //A COMPLETER;
      //A COMPLETER;
   }
}
```

Sortie console

Pour la sortie console vous pouvez vous baser sur le diagramme de séquence afin d'implémenter le boundary.

Scénario de Test

```
---------- CREATION CLIENT ----------
Nom :
CHAUDET
Prénom :
Christelle
Adresse mail :
Christelle.Chaudet@irit.fr
```

```
Mot de Passe :
mdp

---------- CONNECTION CLIENT ----------
Adresse mail :
Christelle.Chaudet@irit.fr
Mot de passe :
mdp
Vous êtes connecté

---------- RESERVER TABLE ----------
Quand souhaitez vous resevez ?
Pour quel mois ?
05
Pour quel jour ?
19
Pour combien de personnes ?
3
Pour quel service ?
1
Numéro de table : 1
Numéro de table : 2
Choisissez votre table
2
---------- CONTROL DES DONNEES ----------
Client :
nom=CHAUDET, prenom=Christelle, adresseMail=Christelle.Chaudet@irit.fr, mdp=mdp
Reservation :
Le 19/5 : table n°2 pour le premier service.
```

Sortie IHM

Actuellement, au vu de vos connaissances en ILU2 sur la partie IHM, vous ne pouvez tester qu'un seul cas d'utilisation de manière graphique. Par contre, la partie graphique doit utiliser le contrôleur correspondant.
De plus, les données récupérées des cas d'utilisation précédents peuvent être nécessaires dans le cas d'utilisation que vous souhaitez implémenter en version graphique.

Pour cela vous devrez lancer tous les cas nécessaires avec les boundary de la version console, créer votre dialogue avec le ou les constructeur adéquat puis transmettre vos données nécessaires au fonctionnement du dialogue.

Exemple pour le cas *"Réserver table"*.

Créer la classe *TestReserverTable_IHM* dans le paquetage *test_fonctionnel*.

```
public class TestReserverTable_IHM {
  public static void main(String[] args) {
    System.out.println("---------- CREER CLIENT ----------");
    // ENTITE : Creation du carnet de clientele
    CarnetClientele carnetClientele = new CarnetClientele();
    //

    ControlCreerClient controlCreerClient = new ControlCreerClient(carnetClientele);
    BoundaryCreerClient boundaryCreerClient
      = new BoundaryCreerClient(controlCreerClient);
    boundaryCreerClient.creerClient();

    System.out.println("\n---------- CONNECTER CLIENT ----------");
    ControlConnecterClient controlConnecterClient =
      new ControlConnecterClient(carnetClientele);
    BoundaryConnecterClient boundaryConnecterClient
      = new BoundaryConnecterClient(controlConnecterClient);
    int numClient = boundaryConnecterClient.connecterClient();

    System.out.println("\n---------- RESERVER TABLE ----------");
    // ENTITE : Creation et configuration du restaurant
    Restaurant restaurant = new Restaurant();
    restaurant.ajouterTable(2);
    restaurant.ajouterTable(4);
    restaurant.ajouterTable(4);
    restaurant.ajouterTable(4);
    restaurant.ajouterTable(4);
    restaurant.ajouterTable(8);
    //

    ControlReserverTable controlReserverTable
      = new ControlReserverTable(restaurant, carnetClientele);
    ControlVisualiserCarnetClientele controlVisualiserCarnetClientele
      = new ControlVisualiserCarnetClientele(carnetClientele);
    DialogReservation dialogReservation
      = new DialogReservation(controlReserverTable, controlVisualiserCarnetClientele);
    EventQueue.invokeLater(new Runnable() {
      public void run() {
        try {
          dialogReservation.initDialog();
          dialogReservation.handleUserConnected(numClient);
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    });
  }
}
```

Par rapport à votre travail d'IHM, il faut donc :

- ajouter un constructeur dans le dialogue pour permettre l'injection des contrôleurs,
- ajouter la méthode *handleUserConnected(numClient)* pour permettre d'initialiser le contexte de la partie graphique.

Votre dialogue s'enrichit d'attributs permettant de gérer les données provenant de l'application (attention par rapport à votre TP IHM des types ont changés) :

```
private int numClient;
private int numService;
private int nombrePersonnes;
private int jour;
private int mois;
private int numTable;
private int[] proposition;
private ControlReserverTable control;
private ControlVisualiserCarnetClientele controlVisualiserCarnetClientele;
```

A chaque fois que vous avez besoin de transmettre des données à votre modèle métier (les entités) ou de récupérer des données pour affichage vous devez utiliser le contrôleur.

Dans l'exemple de la réservation vous devrez :

1. appeler le contrôleur pour récupérer les disponibilités des tables (afin de construire la liste des tables à afficher). Pour cet appel vous aurez besoin de la date, du numéro de service et du nombre de personnes. Vous devrez donc :
   - modifier les méthodes *handleXSelectedEvent*, X pour *Date*, *Time*, *NumofPersons*

   - compléter la méthode :

```
public void handleNumofPersonsSelectedEvent(String numPersonnes) {
   this.nombrePersonnes = Integer.parseInt(numPersonnes);
   proposition
      = control.trouverPossibilite(jour, mois, nombrePersonnes, numService);
   presentationReservation.initTableMap(proposition);
   presentationReservation.enableTableMap();
}
```

avec initTableMap :

```
public void initTableMap(int[] propositions) {
     int nbProposition = 0;
     int[] propositionRetenue = new int[propositions.length-1];
     for(int i = 1 ; i<propositions.length -1; i++) {
       if(propositions[i] != 0) {
         propositionRetenue[nbProposition] = propositions[i];
         nbProposition++;
       }
```

```
        }
        propString = new String[nbProposition];
        for (int i = 0; i < nbProposition; i++) {
          propString[i] = "" + propositionRetenue[i];
        }
        listTableMap.setModel(new AbstractListModel() {
          String[] values = propString;
          public int getSize() {
            return values.length;
          }
          public Object getElementAt(int index) {
            return values[index];
          }
        });
    }
```

2. Vous devrez faire de nouveau appel au constructeur pour effectuer la réservation de la table choisie, et visualiser l'état du carnet clientèle.
   - Compléter la méthode *handleValidateEvent* :

```
public void handleValidateEvent() {
   //Reserver table
   control.reserver(...);
   String donnees = (...);
   presentationReservation.enableValidationInformation(donnees);
}
```

   - Dans la présentation on aura :

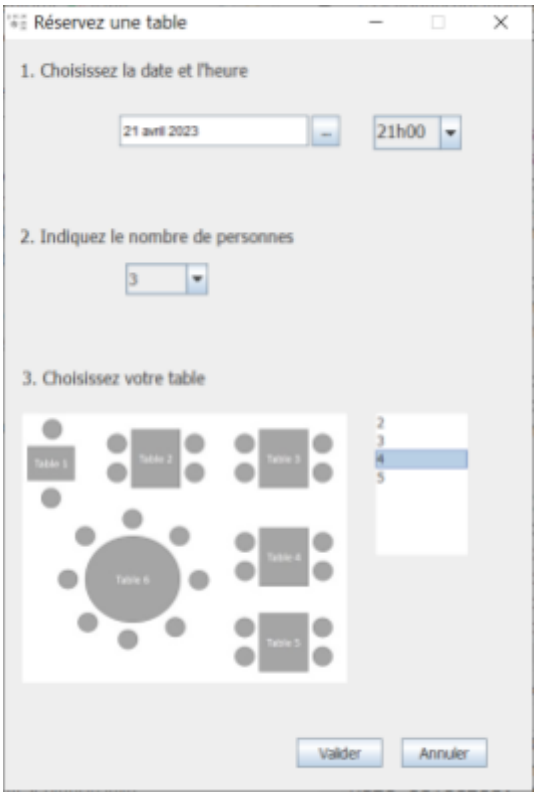
```
public void enableValidationInformation(String donnees) {
   JOptionPane.showMessageDialog(this, donnees);
}
```

Exemple de trace console et de sortie graphique obtenues

```
---------- CREATION CLIENT ----------
Nom :
Chaudet
Prénom :
Christelle
Adresse mail :
Christelle.Chaudet@irit.fr
Mot de Passe :
mdp

---------- CONNECTION CLIENT ----------
Adresse mail :
Christelle.Chaudet@irit.fr
Mot de passe :
mdp
Vous êtes connecté
```

```
---------- RESERVER TABLE ----------
InitPresentation done
```




```
InitPresentation done
Date selected: 2023-04-21
Time selected: 21h00
Number of persons selected: 3
Table selected: 4
Table selected: 4
```