Mini projet : Médiathèque

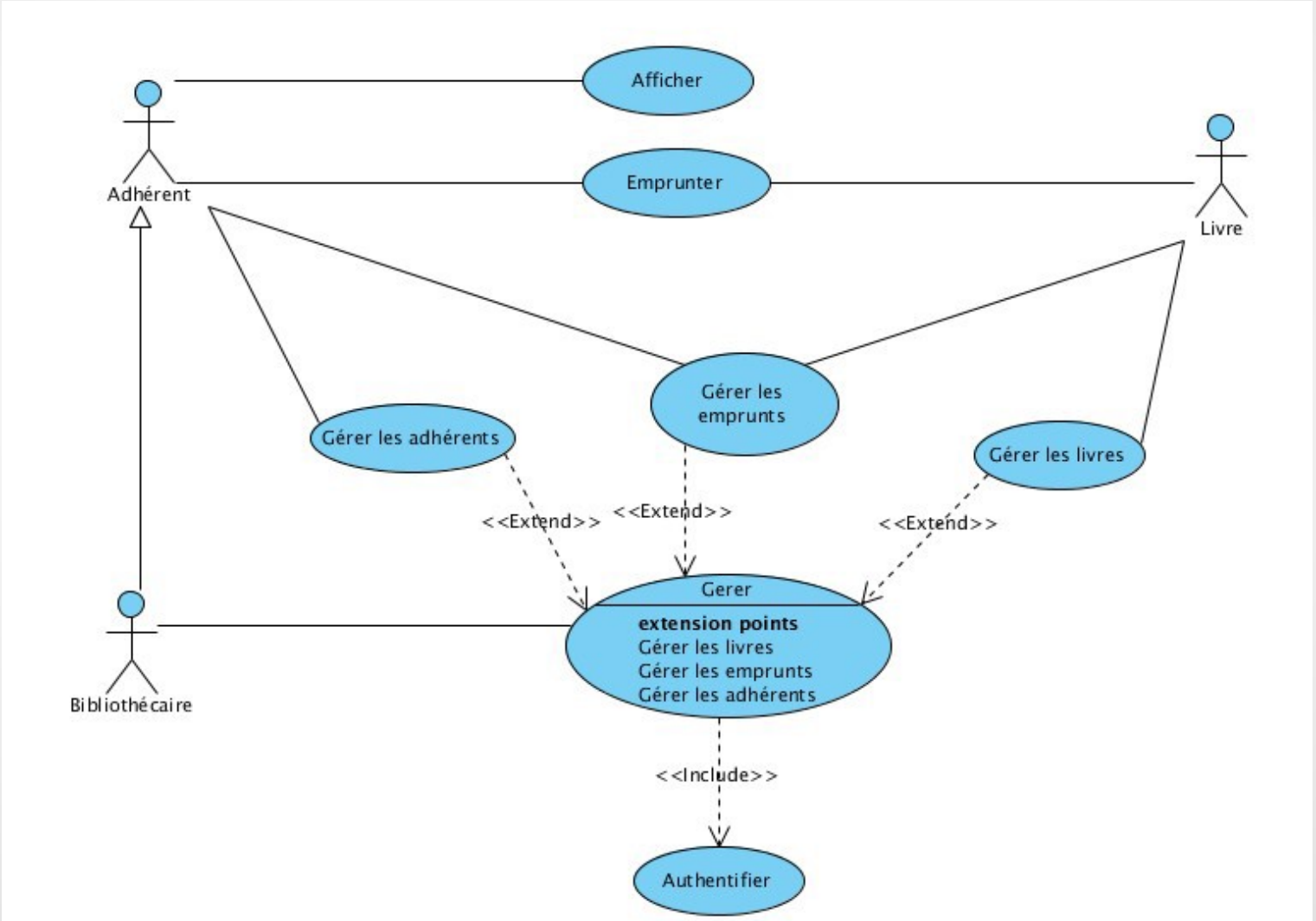
ALONSO Stéphane S1SNir

Présentation du système

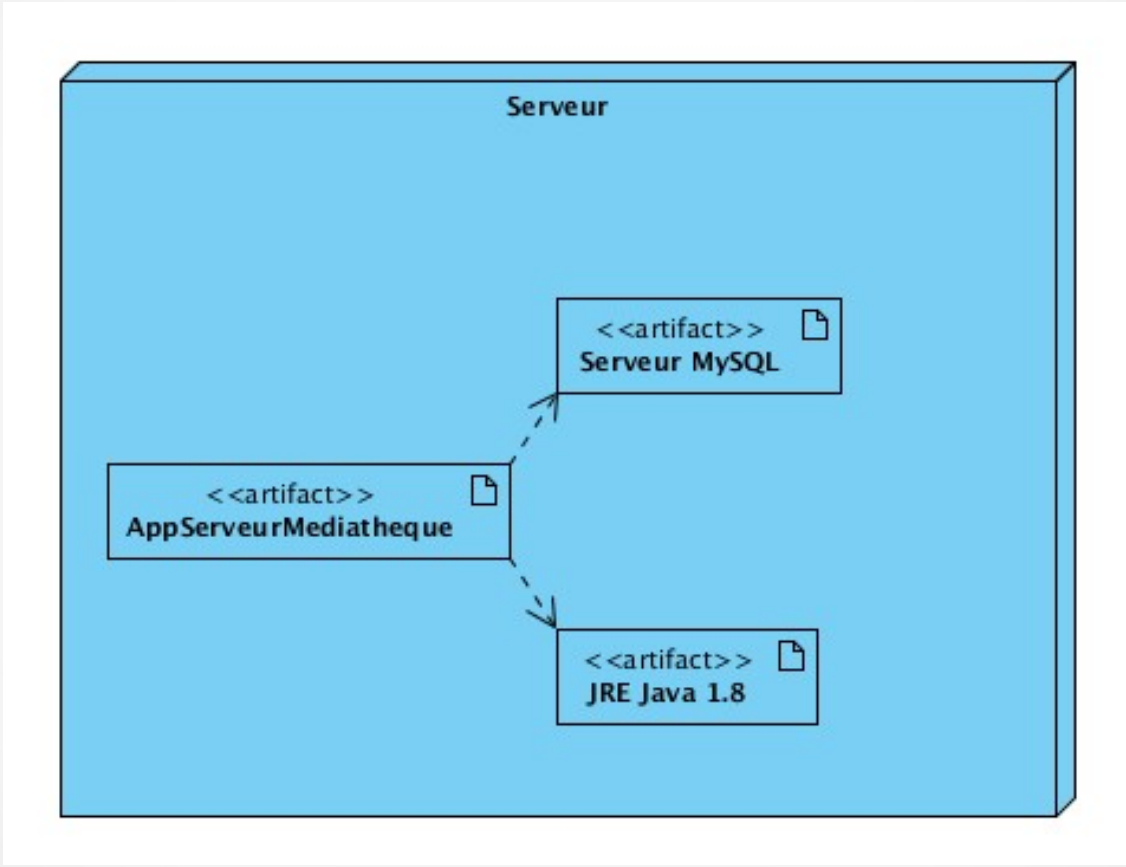
Le système à réaliser permettra de gérer une médiathèque, c’est à dire d’assurer :

- Gestion des livres ;
- Gestion des adhérents ;
- Gestion des emprunts.

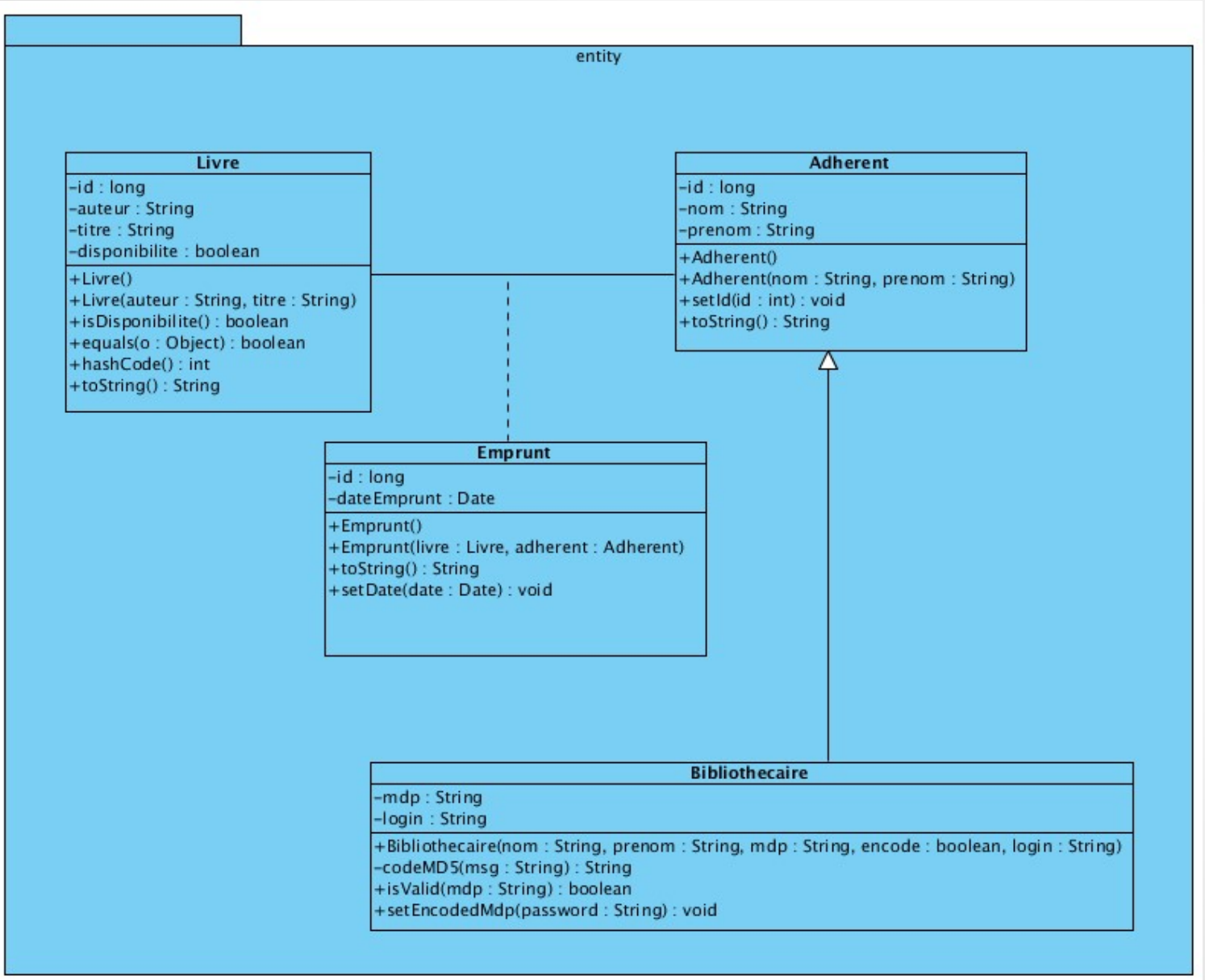
Cas d’utilisation



Déploiement



Classe entités

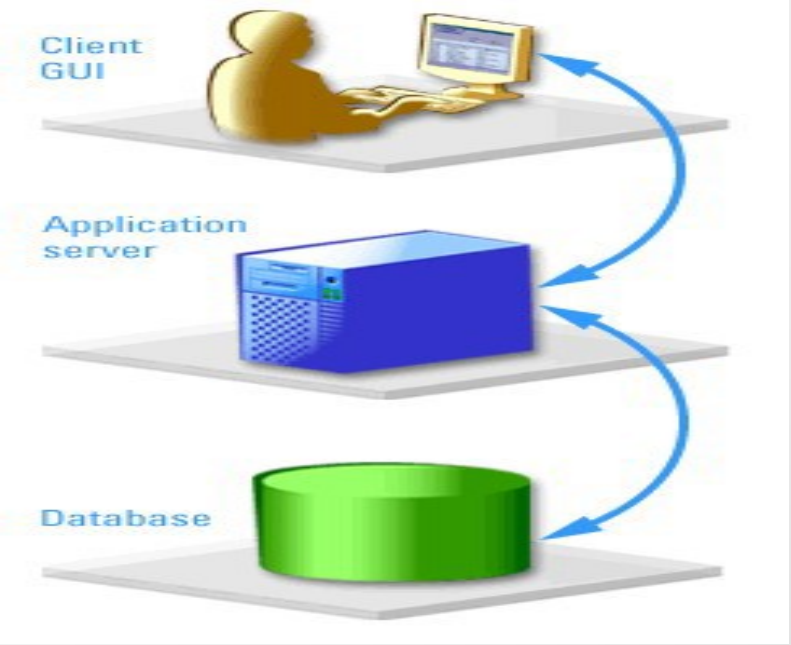


Architecture : n-tiers

Il s’agit d’un modèle logique d’architecture applicative qui vise à modéliser une application comme un empilement de n couches logicielles dont le rôle est clairement défini.

Nous découpons notre application en trois niveaux ou couches :

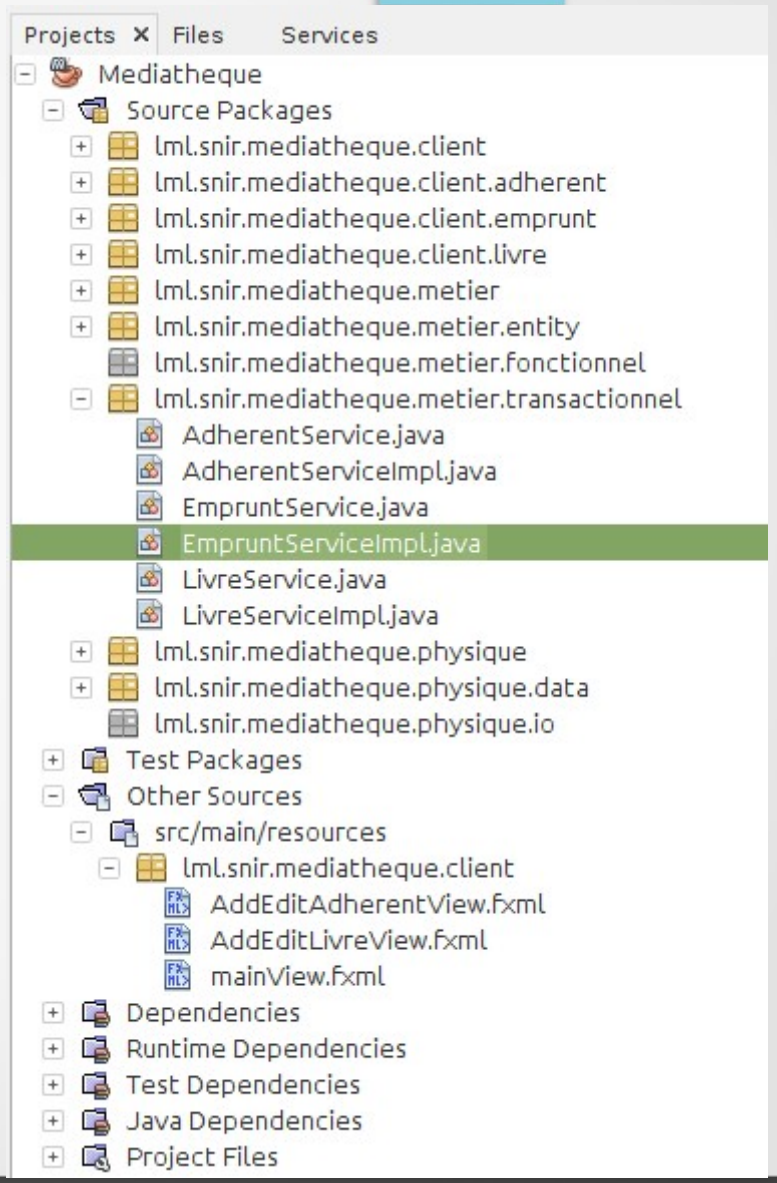
- Couche présentation ou client ;
- couche métier (ou applicative) ;
- couche physique (base de données).



C’est une extension du modèle client-serveur.

Structure du projet : vue générale

- On peut remarquer la présence des 3 couches pour le n-tiers ;
- L'utilisation du pattern MVC pour la couche client avec les vue en fxml ;
- Les services métier ;
- La couche ORM (physique data).



Code métier : ajout d’un emprunt

Start Page × EmpruntServiceImpl.java ×

Source History

```
37
38      @Override
39      public Emprunt add(Emprunt emprunt) throws Exception {
40          // check si max 3 emprunt respecte
41          int count = this.empruntDataService.getCountByAdherent(emprunt.getAdherent());
42          if (count == 3) {
43              throw new Exception("Désolé le nombre maximun d'emprunt est atteint !");
44          }
45
46          // check si pas deux fois meme livre
47          List<Emprunt> emprunts = this.empruntDataService.getByAdherent(emprunt.getAdherent());
48          for (Emprunt e : emprunts) {
49              if (e.getLivre().equals(emprunt.getLivre())) {
50                  throw new Exception("Désolé on ne peut pas emprunter deux livres identiques en même temps !");
51              }
52          }
53
54          // check si livre dispo
55          if (!emprunt.getLivre().isDisponibilite()) {
56              throw new Exception("Désolé livre non disponible !");
57          }
58
59          // tout ok marquer lmivre non dispo et ajouter emprunt dans bdd
60          Livre l = emprunt.getLivre();
61          l.setDisponibilite(false);
62
63          PhysiqueDataFactory.getLivreDataService().update(l);
64
65          return this.empruntDataService.add(emprunt);
66      }
67
```

Base de donnée : JDBC

- Afin de communiquer avec la base de donnée nous utilisons la technologie JDBC de Java ;
- Nous utilisons aussi une librairie écrite par les professeurs de la section pour faciliter et standardiser le mappage relationnel-objet des classe entités.

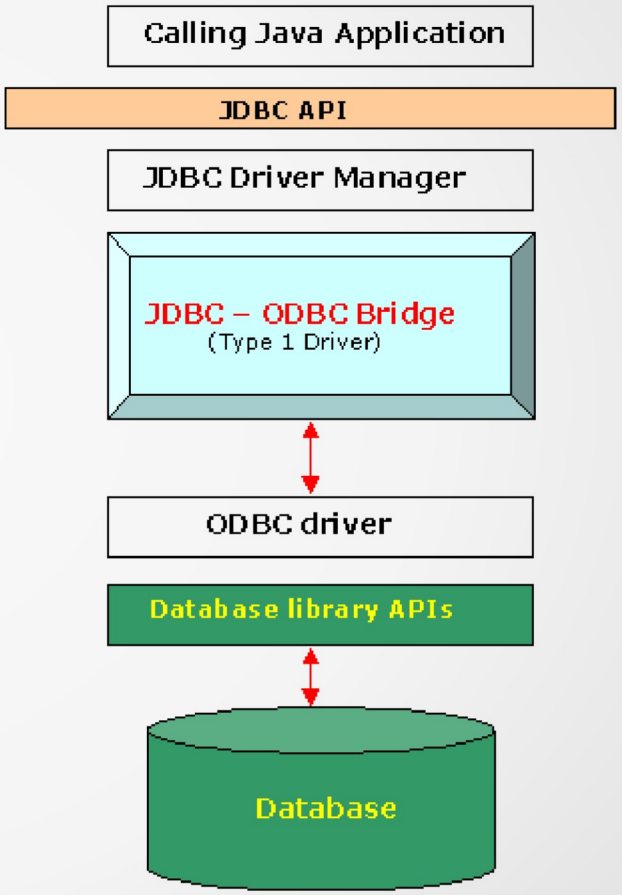
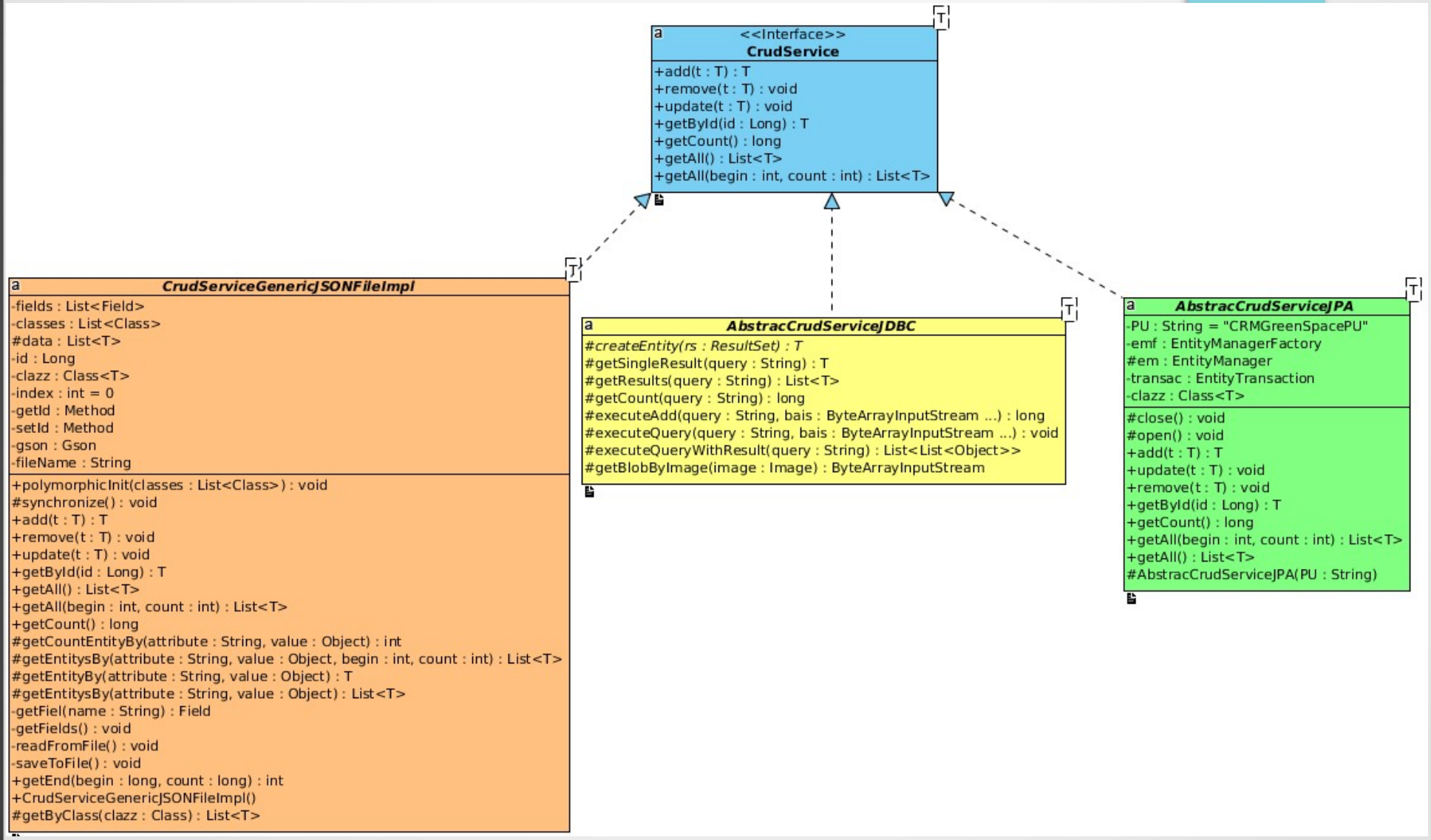


Diagramme de classe lib section



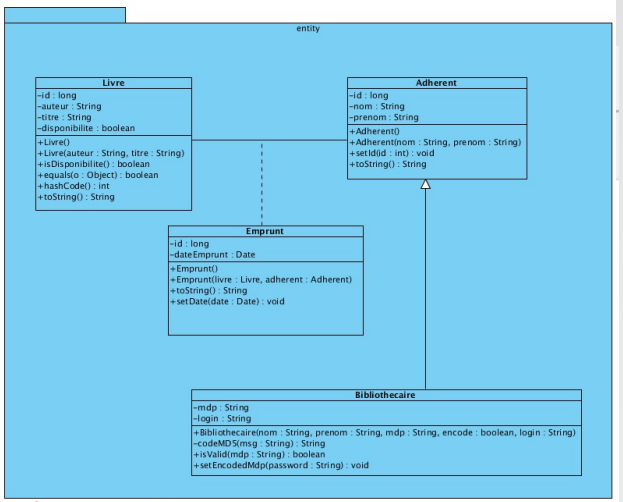
Code ORM : Définition du service

Start Page × EmpruntDataService.java ×

Source History

```
1   package lml.snir.mediatheque.physique.data;
2
3   import java.util.Date;
4   import java.util.List;
5   import lml.snir.mediatheque.metier.entity.Adherent;
6   import lml.snir.mediatheque.metier.entity.Emprunt;
7   import lml.snir.mediatheque.metier.entity.Livre;
8   import lml.snir.persistence.CrudService;
9
10  /**
11   *
12   * @author fanou
13   */
    public interface EmpruntDataService extends CrudService<Emprunt>{
        public Emprunt getByLivre(Livre livre) throws Exception;
        public List<Emprunt> getByAdherent(Adherent adherent) throws Exception;
        public int getCountByAdherent(Adherent adherent) throws Exception;
        public List<Emprunt> getByDate(Date date) throws Exception;
19  }
```

Code ORM : Codage du service - 1

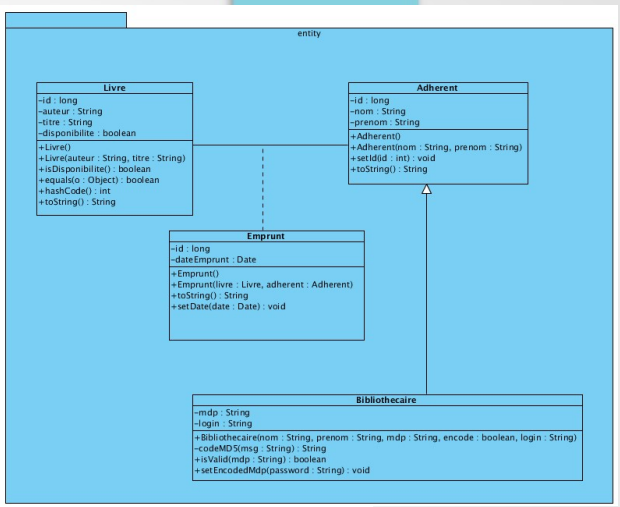


Start Page | EmpruntDataServiceJDBCImpl.java

Source | History

```
7  import lml.snir.mediatheque.metier.entity.Adherent;
8  import lml.snir.mediatheque.metier.entity.Emprunt;
9  import lml.snir.mediatheque.metier.entity.Livre;
10 import lml.snir.persistence.jdbc.AbstractCrudServiceJDBC;
11
12 /**
13  *
14  * @author fanou
15  * @param <T>
16  */
17 public class EmpruntDataServiceJDBCImpl<T> extends AbstractCrudServiceJDBC<Emprunt> implements EmpruntDataService {
18     private SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
19
20     public EmpruntDataServiceJDBCImpl() throws Exception {
21         String query = null;
22         try {
23             switch (super.getDBType()) {
24                 case SQLITE:
25                     query = "CREATE TABLE IF NOT EXISTS `" + super.getEntityName() + "` (\n"
26                             + "  `id` INTEGER PRIMARY KEY AUTOINCREMENT,\n"
27                             + "  `date` varchar(16) NOT NULL,\n"  // Adherent ou Bibliothecaire
28                             + "  `idLivre` INTEGER NOT NULL UNIQUE,\n"
29                             + "  `idAdherent` INTEGER NOT NULL\n"
30                             + ");";
31                     break;
32             }
33             super.executeQuery(query);
34
35         } catch (Exception ex) {
36             System.out.println(this.getClass().getSimpleName() + "\n"  + super.getDBType() + "\n" + ex);
37         }
38     }
```

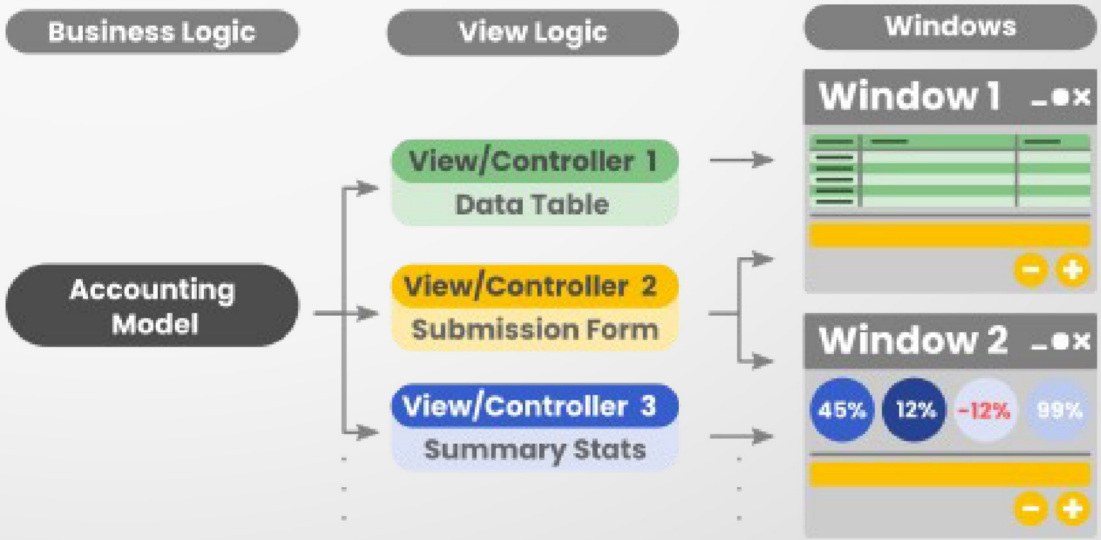
Code ORM : Codage du service - 2



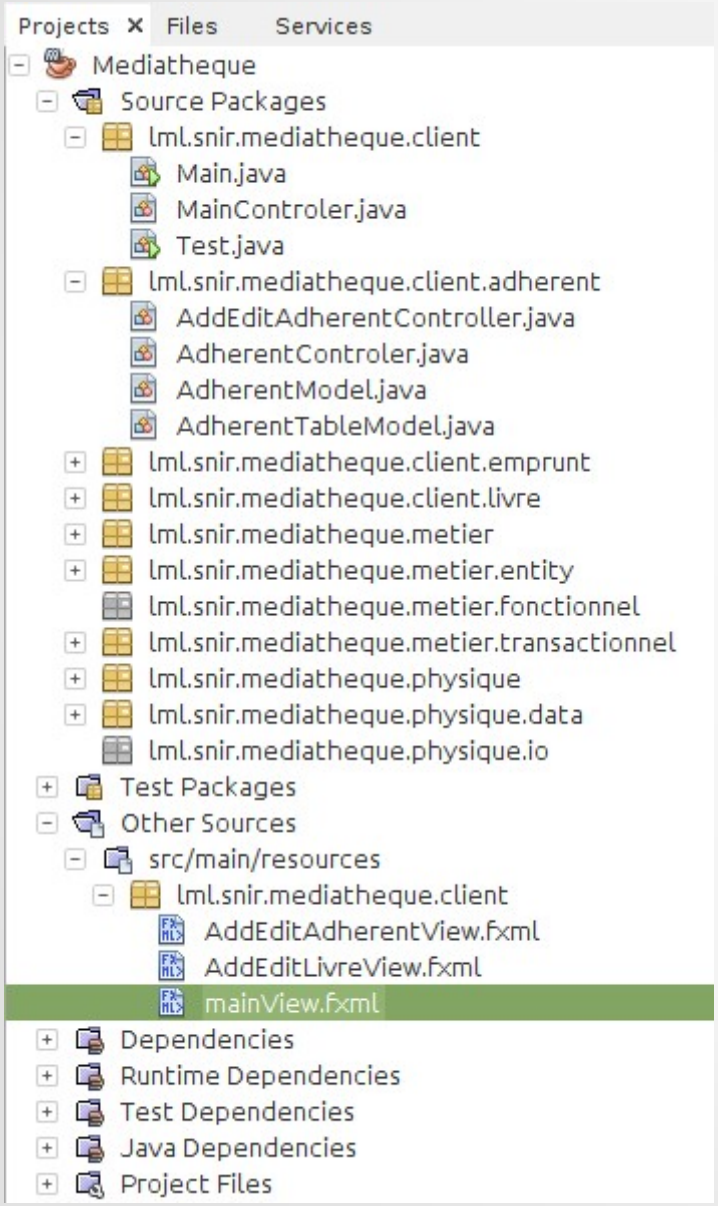

```java
@Override
protected Emprunt createEntity(Map map) throws Exception {
    Emprunt emprunt = null;

    long id = (int) map.get("id");
    String dateStr = (String) map.get("date");
    long idLivre = (int) map.get("idLivre");
    long idAdherent = (int) map.get("idAdherent");

    Livre livre = PhysiqueDataFactory.getLivreDataService().getById(idLivre);
    Adherent adherent = PhysiqueDataFactory.getAdherentDataService().getById(idAdherent);
    Date date = this.sdf.parse(dateStr);

    emprunt = new Emprunt(livre, adherent);
    emprunt.setDate(date);
    emprunt.setId(id);

    return emprunt;
}

@Override
public Emprunt add(Emprunt emprunt) throws Exception {
    String query = "INSERT INTO " + super.getEntityName() + " (date, idLivre, idAdherent) VALUES ('"
                + this.sdf.format(emprunt.getDateEmprunt()) + "','"
                + emprunt.getLivre().getId() + "','"
                + emprunt.getAdherent().getId() + "')";
    emprunt.setId(super.executeAdd(query));
    return emprunt;
}
```

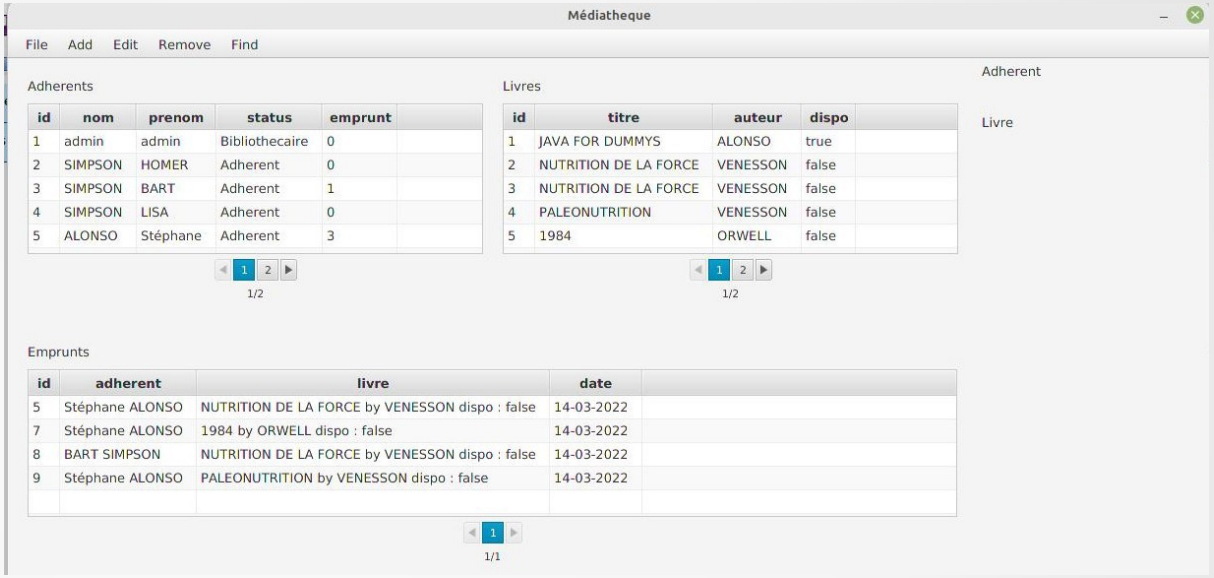
Code client : MVC - JavaFX

- Une des architectures, communément utilisée, et qui comporte de nombreuses variantes, est connue sous l'acronyme MVC qui signifie Model - View - Controller.
- Dans cette architecture on divise le code des applications en entités distinctes (modèles, vues et contrôleurs) qui communiquent entre elles au moyen de divers mécanismes (invocation de méthodes, génération et réception d'événements, etc.).



Code client : Structure générale





Code client : Modèles

```java
public class AdherentModel extends LMLModel<Adherent> {

    private long id;
    private String nom;
    private String prenom;
    private String status;
    private int emprunt;

    @Override
    public void setObjectModel(Adherent t) {
        this.id = t.getId();
        this.nom = t.getNom();
        this.prenom = t.getPrenom();
        this.status = t.getClass().getSimpleName();
        try {
            this.emprunt = MetierFactory.getEmpruntService().getByAdherent(t).size();
        } catch (Exception ex) {

        }
    }
```

Adherents

id	nom	prenom	status	emprunt
1	admin	admin	Bibliothecaire	0
2	SIMPSON	HOMER	Adherent	0
3	SIMPSON	BART	Adherent	1
4	SIMPSON	LISA	Adherent	1
5	ALONSO	Stéphane	Adherent	3

1 2

1/2

```java
12  public class AdherentTableModel extends AbstractJFXTableModel<AdherentModel> {
13
14      public AdherentTableModel(TableView table) throws Exception {
15          super(table, AdherentModel.class);
16          super.setService(MetierFactory.getAdherentService());
17      }
18
19      void findByNom(String nom) throws Exception {
20          List data = MetierFactory.getAdherentService().getByNom(nom);
21          super.update(data);
22      }
23
24  }
25
```

Code client : Contrôleur

```
public class AdherentControler extends LMLController {

    private AdherentService aderentSrv;
    private AdherentModel adherentSelected;
    private final TableView<AdherentModel> table;
    private int index;

    private AdherentTableModel atm;

    public AdherentControler(TableView table) {
        this.table = table;
    }

    @Override
    public void init() throws Exception {
        // to do init here
        this.atm = new AdherentTableModel(this.table);
        this.atm.setNbPerPage(5);
        this.atm.setPagination(this.pagination);
        this.atm.init();

        this.aderentSrv = MetierFactory.getAdherentService();
    }

    @Override
    public void add() throws Exception {...10 lines }

    @Override
    public void update() throws Exception {...14 lines }
```

- Le contrôleur est chargé de réagir aux différentes actions de l'utilisateur ou à d'autres événements qui peuvent survenir.
- Il définit le comportement de l'application et sa logique.
- Dans les applications simples,il gère la synchronisation entre la vue et le modèle (rôle de chef d'orchestre).

Code client : Vue

Start Page × AdherentModel.java × AdherentTableModel.java × AdherentControler.java × mainView.fxml × AddEditAdherentView.fxml ×

Source History

```
<?import javafx.scene.control.CheckBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.RowConstraints?>

<AnchorPane id="AnchorPane" prefHeight="383.0" prefWidth="601.0" xmlns="http://javafx.com/javafx/8.0.121" xmlns:fx="http://javafx.com/fxml/1" fx:controller="lml.
   <children>
      <GridPane prefHeight="400.0" prefWidth="600.0">
        <columnConstraints>
        <rowConstraints>
         <children>
            <Label text="Nom" />
            <Label text="Prénom" GridPane.rowIndex="1" />
            <Label text="Adminsintrateur" GridPane.rowIndex="2" />
            <Label text="Login" GridPane.rowIndex="3" />
            <Label text="Mot de passe" GridPane.rowIndex="4" />
            <PasswordField fx:id="passwordFieldMdp" GridPane.columnIndex="1" GridPane.rowIndex="4" />
            <TextField fx:id="textFieldLogin" GridPane.columnIndex="1" GridPane.rowIndex="3" />
            <TextField fx:id="textFieldPrenom" GridPane.columnIndex="1" GridPane.rowIndex="1" />
            <TextField fx:id="textFieldNom" GridPane.columnIndex="1" />
            <CheckBox fx:id="isAdmin" mnemonicParsing="false" text="Oui" GridPane.columnIndex="1" GridPane.rowIndex="2" />
            <Button fx:id="buttonOK" mnemonicParsing="false" text="OK" GridPane.rowIndex="5" onAction="#buttonOkClick" />
            <Button fx:id="buttonCancel" mnemonicParsing="false" text="Annuler" GridPane.columnIndex="1" GridPane.rowIndex="5" onAction="#buttonCancelClick" />
         </children>
      </GridPane>
   </children>
</AnchorPane>
```

États d’avancement

- Application terminée à 90 %;
- Reste qq bugs ;
- Environ 8h de travail pour terminer.

Conclusion

- Projet qui m’a permis de réviser l’ensemble des notions vue en cours ;
- Mieux appréhender le fonctionnement complexe de JavaFX et du MVC.