Mapping Objet Relationnel avec JDBC (suite)

- **Exercice sur le Mapping Objet Relationnel avec JDBC**

Exemple deMapping Objet Relationnel avec JDBC

```
public class Team implements Serializable{
  private Long id;←❶
  private String name;←❷

/**
 * Contructeur par défaut←❻
 */
 public Team() {}

// méthodes métier←❼
// getters & setters←❽
```

Écrire en JDBC une méthode *persist(Team …)* qui permet d'enregistrer un **nouvel objet de type Team** dans la base ?

Exemple d'utilisation de la méthode ***persist(Team …)*** pour enregistrer l'équipe «Olympique de Marseilles» dans la base**:**

```
Team om = new Team();
om.setId(001);
om.setName("Olympique de Marseilles");
Class.forName("org.hsqldb.jdbcDriver");
persist(om) ;
```

Code de la Méthode: persist(*Team* myTeam){ …

```
try {
  // étape 1: récupération de la connexion
  con = DriverManager.getConnection("jdbc:hsqldb:test","sa","");
  // étape 2: le PreparedStatement
  PreparedStatement createTeamStmt;
  String s = "INSERT INTO TEAM VALUES (?, ?         )";
  createTeamStmt = con.prepareStatement(s);
  createTeamStmt.setInt(1, myTeam.getId());
  createTeamStmt.setString(2, myTeam.getName());



  // étape 3: exécution de la requête
  createTeamStmt.executeUpdate();
  // étape 4: fermeture du statement
  createTeamStmt.close();
  con.commit();
} catch (SQLException ex) {
  if (con != null) {
    try {
      con.rollback();
    } catch (SQLException inEx) {
      throw new Error("Rollback failure", inEx);
    }
  }
  throw ex;
} finally {
  if (con != null) {
    try {
      con.setAutoCommit(true);
      // étape 5: fermeture de la connexion
      con.close();
```

Exercice: Ajout de la table **Player qui stocke les joueurs d'une équipe**

```
create table Team (
  TEAM_ID integer generated by default as identity (start with 1),
  name varchar(255),
  coach_id integer,
  primary key (TEAM_ID)
)
create table Player (
  id integer generated by default as identity (start with 1),
  name varchar(255),
  TEAM_ID integer,
  primary key (id)
)
```

Donner le diagramme de classe correspondant au mapping de ces deux Tables ?

avec « **foreign key(TEAM_ID) references Team(TEAM_ID)** » est une contrainte **de la table Player**:

ALTER TABLE *player* ***ADD CONSTRAINT*** *fk_player_team*
FOREIGN KEY (TEAM_ID) *REFERENCES* ***Team(TEAM_ID)****;*

Exercice: Le Diagramme de classes
(*exemple d'une Association bidirectionnelle*)

C'est un exemple d'association bidirectionnelle dans le diagramme de classes:

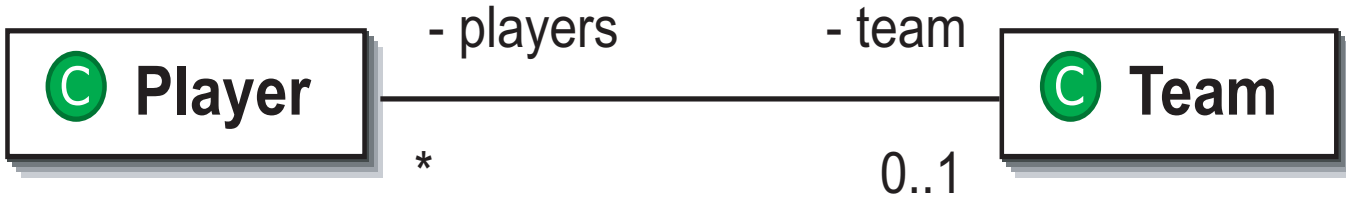
- la **navigabilité est active sur les deux classes** qui sont les deux «extrémités» de l'association.

- En base de données, **les tables** étant simplement **liées par une clé étrangère**

- La colonne sur laquelle porte **la clé étrangère représente le lien entre les deux classes**

Exercice: Le Diagramme de classes

(*exemple d'une Association bidirectionnelle*)




```
public class Team {
        private int id;
        private String name;
        private Set<Player> players = new HashSet<Player>();
        …
}
```

```
public class Player {
        private int id;
        private String name;
        private Team team;
        …
}
```

Utilisation des classes *Team* et *Player*

```
Team om = new Team();
…
Player cisse = new Player();
cisse.setName( "CISSE");
om.getPlayers().add(cisse);
cisse.setTeam(om);

//ajout d'un deuxième « Player »
Player barthez= new Player();
barthez.setName( "Barthez");
om.getPlayers().add(barthez);
barthez.setTeam(om);
…
…
```

```
public class Team {
        private int id;
        private String name;
        private Set<Player> players =
                        new HashSet<Player>();

        // ajouter les getters & setters
        …
        « Set<Player>  getPlayers(){
                return  this.players;
        }
}

public class Player {
                private int id;
                private String name;
                private Team team;
                // ajouter getters & setters
}
```

Exercice sur le Mapping Objet Relationnel avec JDBC

Modifier la méthode *persist(Team …)* pour ajouter l'insertion des données des joueurs (***players*)** dans la base de données, ceci en utilisant la méthode «***getPlayers()***» de la classe « ***Team*** »: La méthode «***getPlayers()***» retourne les joueurs (players) de chaque équipe (i.e., l'attribut «***Set<Player> players***» de la Classe ***Team***) .

Exemple d'utilisation de la méthode ***persist(Team …)*** pour enregistrer une équipe **qui contient des joueurs:**

```
Team om = new Team();
om.setName(...); ...
Player cisse = new Player();
cisse.setName( "CISSE");
om.getPlayers().add(cisse);
cisse.setTeam(om);
//ajout d'un deuxième « Player »
...
om.getPlayers().add(barthez);
barthez.setTeam(om);
persist(om) ;
```

Exercice sur le Mapping Objet Relationnel avec JDBC

Pour modifier la méthode persist(*Team* …), il faut ajouter (après l'étape 4 du code ci-dessous) l'insertion des données des **players** dans la base de données, ceci en utilisant la méthode «***getPlayers()***» de la classe «***Team***»

Code de la Méthode: persist(*Team* myTeam){ …

```
persist(Team  myTeam){
    try {
    // étape 1: récupération de la connexion
    con =DriverManager.getConnection("jdbc:hsqldb:test","sa","");
    // étape 2: le PreparedStatement
    PreparedStatement createTeamStmt;
    String s = "INSERT INTO TEAM VALUES (?, ?)";
    createTeamStmt = con.prepareStatement(s);
    createTeamStmt.setInt(1, myTeam.getId());
    createTeamStmt.setString(2, myTeam.getName());
    // étape 3: exécution de la requête
    createTeamStmt.executeUpdate();
    // étape 4: fermeture du statement
    createTeamStmt.close();

    … … …
}
```

Exercice sur le Mapping Objet Relationnel avec JDBC

Pour modifier la méthode persist(*Team* …), il faut ajouter (après l'étape 4 du code ci-dessous) l'insertion des données des **players** dans la base de données, ceci en utilisant la méthode «***getPlayers()***» de la classe «***Team***»

```
persist(Team  myTeam){
   try {
    // étape 1: récupération de la connexion
    con =DriverManager.getConnection("jdbc:hsqldb:test","sa","");
    // étape 2: le PreparedStatement
    PreparedStatement createTeamStmt, createPlayerStmt;
    String s = "INSERT INTO TEAM VALUES (?, ?)";
    createTeamStmt = con.prepareStatement(s);
    createTeamStmt.setInt(1, myTeam.getId());
    createTeamStmt.setString(2, myTeam.getName());
    createTeamStmt.executeUpdate(); // étape 3
    createTeamStmt.close(); // étape 4
    String s2 = "insert into Player (name, TEAM_ID) values (?, ?);
    createPlayerStmt = con.prepareStatement(s2);
       for (Iterator it=myTeam.getPlayers().iterator();it.hasNext();){
                      player = (Player) it.next();      … ???
                      … … … ???
                      createPlayerStmt.executeUpdate();
        }
   } … … …
```

Méthodologie d'association bidirectionnelle

Les lignes ***team.getPlayers().add(player)*** *et* ***player.setTeam(team)*** **doivent être regroupées** dans une méthode métier « de cohérence », en l'occurrence la méthode:

team.addPlayer(player)

- Celle-ci est à écrire une seule fois pour toute l'application,
- après quoi vous n'avez plus à vous soucier de la cohérence de vos instances.

- **Exercice** : Écrire la méthode ***addPlayer(Player p)*** de la classe ***Team***, qui gère les deux extrémités de l'association ?

```
public void addPlayer(Player p){
        this.getPlayers().add(p);
        p.setTeam(this);
}
```

Méthodologie d'association bidirectionnelle

Les lignes ***team.getPlayers().add(player)*** *et* ***player.setTeam(team)*** **doivent être regroupées** dans une méthode métier « de cohérence », en l'occurrence la méthode:

team.addPlayer(player)

Notre exemple de code fait désormais appel à « team.addPlayer(player) »:

```
Team om = new Team();
...
Player cisse = new Player();
cisse.setName( "CISSE");
om.addPlayer(cisse);
//ajout d'un deuxième « Player »
...
om..addPlayer(barthez);
```

Ajout de la table Coach

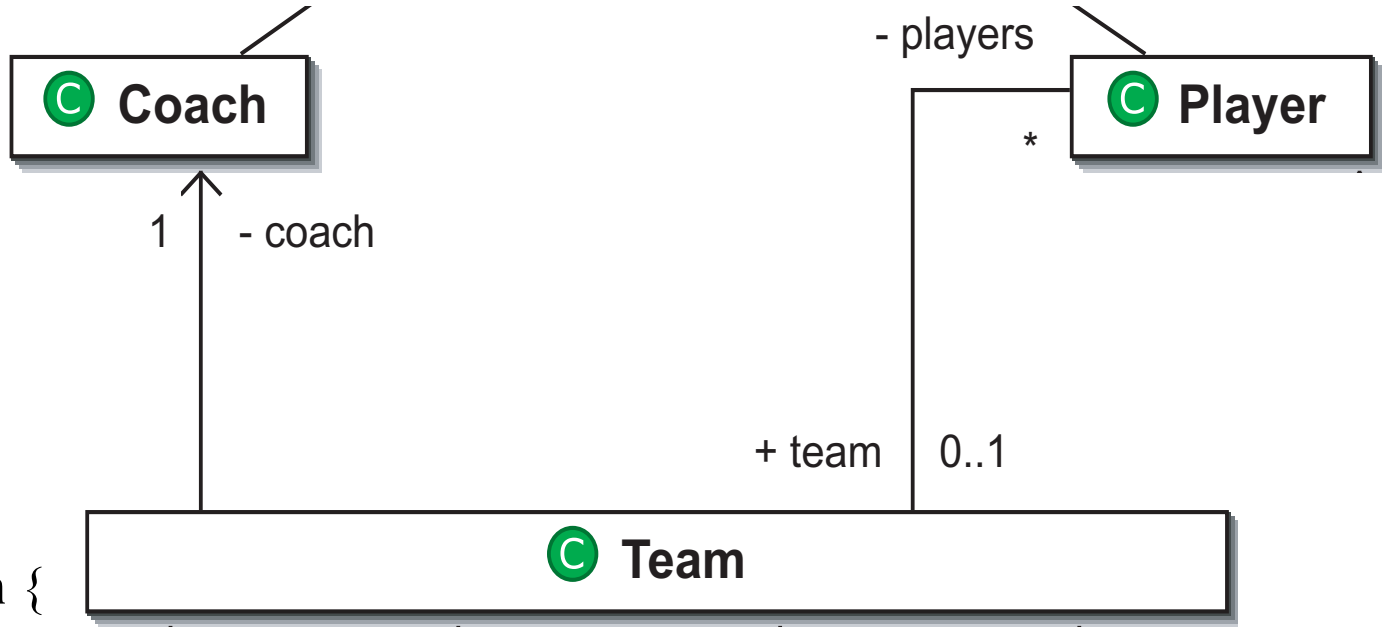
```
create table Team (
  TEAM_ID integer generated by default as identity (start with 1),
  name varchar(255),
  coach_id integer,
  primary key (TEAM_ID)
)
create table Player (
  id integer generated by default as identity (start with 1),
  name varchar(255),
  TEAM_ID integer,
  primary key (id)
)
create table Coach (
  id integer generated by default as identity (start with 1),
  name varchar(255),
  primary key (id)
)
```

Donner le diagramme de classe correspondant au mapping de ces trois Tables ?

(**foreign key(coach_id) references Coach(id)** est une contrainte **de la table Team**)

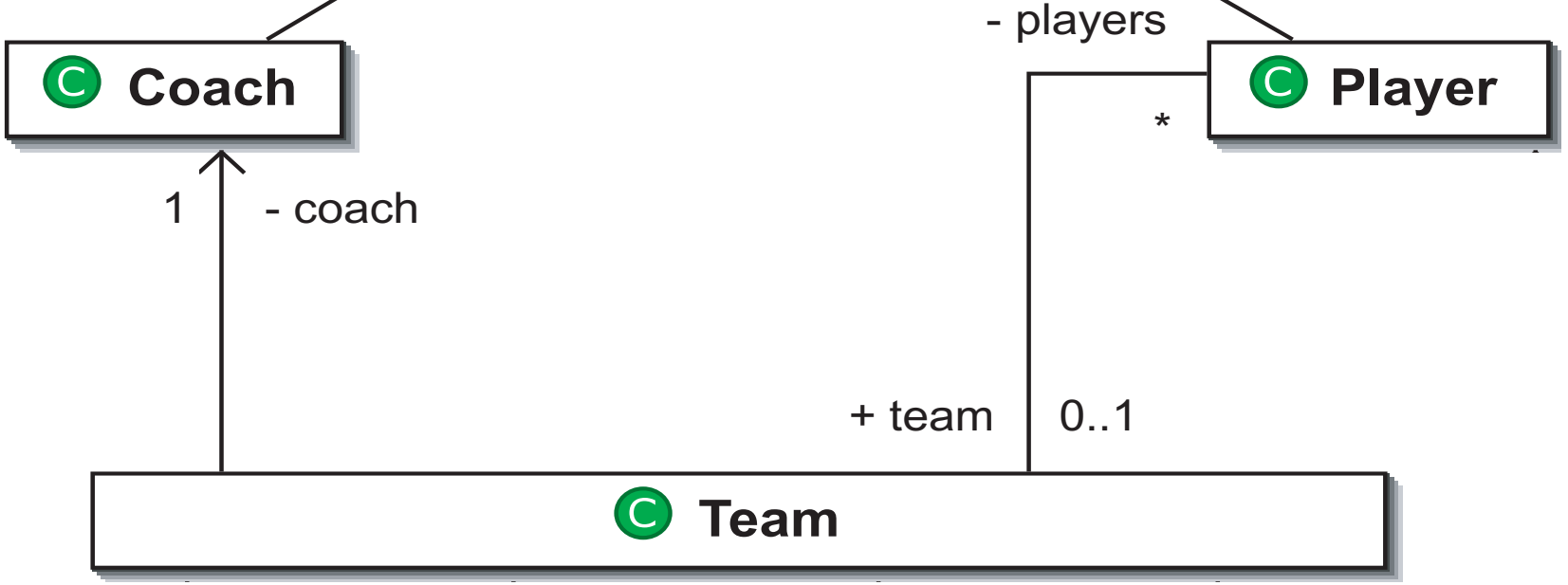
Alter table *Team* ***add constraint*** *FK_TEAM_COACH*
foreign key (coach_id) references (Coach)

Exemple d'une Association unidirectionnelle



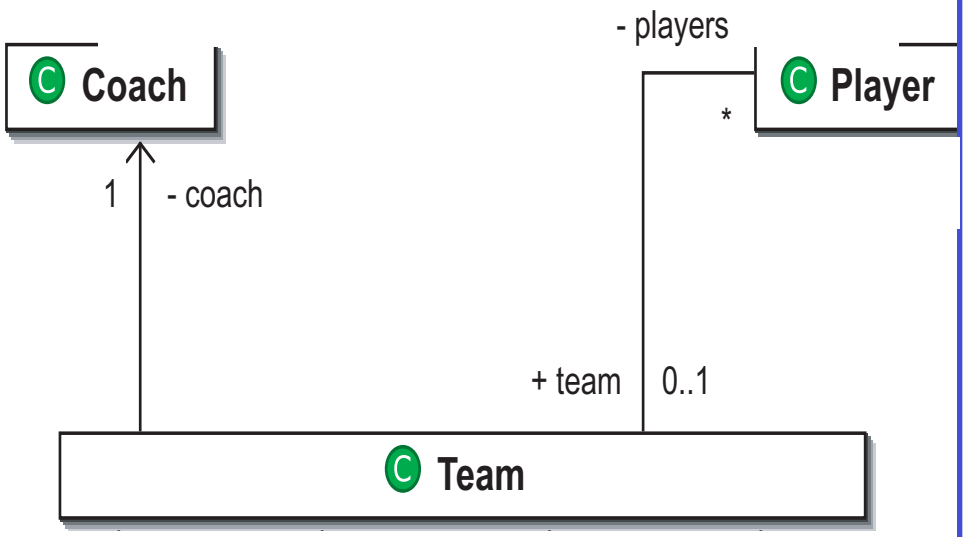

```
public class Team {
        private int id;
        // association unidirectionnelle, rien de particulier
        private Coach coach;
        …
}

public class Coach {
        private int id;
        private String name;
}
```




```
public class Team {
        private int id;
        // association unidirectionnelle, rien de particulier
        private Coach coach;
        …
        private Set<Player> players = new HashSet<Player>();
}

public class Coach {
        private int id;
        private String name;
}
```




```
public class Team {
        private int id;
        // association unidirectionnelle
        private Coach coach;
        private String name;
       …
        private Set<Player> players =
             new HashSet<Player>();
      // ajouter les getters & setters
}

public class Player {
             private int id;
             private String name;
             private Team team;
             // ajouter getters & setters
}

public class Coach {
             private int id;
              private String name;
             // ajouter getters & setters}
```

Récupération des Objets à partir de la Base de données

- **Exemple**: Pour récupérer tous les objets de type « Team » de la base, on peut écrire une méthode « getAllTeams() » qui retourne une liste d'instances de la classe « Team »:

```
List<Team>  getAllTeams(){
        ... ???
        String queryTeams = "select t.TEAM_ID, t.name as tname, c.id, c.name as cname
                            from Team t left outer join Coach c on t.coach_id=c.id" ;

        ... ???
        String queryPlayers = "select id, name  from Player where TEAM_ID=?";
        ... ???

}
```

Récupération des Objets à partir de la Base de données

Exemple: Pour récupérer tous les objets de type « Team » de la base, on peut écrire une méthode « getAllTeams() » qui retourne une liste d'instances de la classe «Team»:

```
List<Team>  getAllTeams(){
        String queryTeams = "select t.TEAM_ID, t.name as tname, c.id as cid, c.name as cname
                            from Team t left outer join Coach c on t.coach_id=c.id" ;
        ResultSet resultSet = connection.createStatement(queryTeams).executeQuery();
        List<Team> teams = new ArrayList<Team>();
        while(resultSet.next()) {
            team = new Team();
            team.setId(resultSet.getInt(1));          team.setName(resultSet. getString("tname"));
            coach= new Coach();         coach.setId(resultSet.getInt("cid"));
            coach.setName(resultSet.getString("cname"));
            team.setCoach(coach);
             ... ???
             String queryPlayers = "select id, name  from Player where TEAM_ID=?";
             ... ???
             teams.add(team);
         }
    return  teams;
}
```

Supprimer un objet de la base de données

- Enlever un objet signifie l'extraire **définitivement** de la base de données.
- La méthode ***remove(Team …)*** permet d'effectuer cette opération.

- **Exercice**: écrire la méthode ***remove(Team …)***

(**Attention**: Ne pas oublier de mettre à jour la table ***Player***)

Supprimer un objet de la base de données

- **Exercice**: écrire la méthode remove(Team …)

(**Attention**: Ne pas oublier de mettre à jour la table ***Player***)

```
remove(Team team){
    … ???

     String q1= update Player set Team_ID=null where Team_ID=?

     … ???

     String q2= delete from Team where Team_ID=?

    … ???

}
```

Exemple de DAO: *PlayerDao*

```
package persistence;

public interface PlayerDao {
    Player  find(Integer id);
    List<Player> findAll();
    Integer  persist(Player p);
    void   update(Player p);
    void  remove(Player p);
}
```

***Exercice**: écrire le code de la méthode « **find**(Integer **id**) » qui retourne l'objet de type Player identifié par le paramètre **id**.*

PlayerDao

```
public class PlayerDaoImpl implements PlayerDao {
    private static final String SELECT_ALL = "SELECT * FROM PLAYER ";
    private static final String SELECT_BY_ID = SELECT_ALL + "WHERE id = ? ";
    private Connection connection;
    public PlayerDaoImpl(Connection connection) { this.connection = connection;}
    public Player find(Integer id) {
        Player p = null;  PreparedStatement ps = null;   ResultSet rs = null;
        try {
            ps = this.connection.prepareStatement(SELECT_BY_ID);
            ps.setInteger(1, id);
            rs = ps.executeQuery();
            while (rs.next()) {
                String name = rs.getString("name");
                Integer team_id = rs.getInteger("id");
                … //il faut ensuite récupérer le Team du player
                …  p = new Player(id, name, team);
                return p;}
        }
```