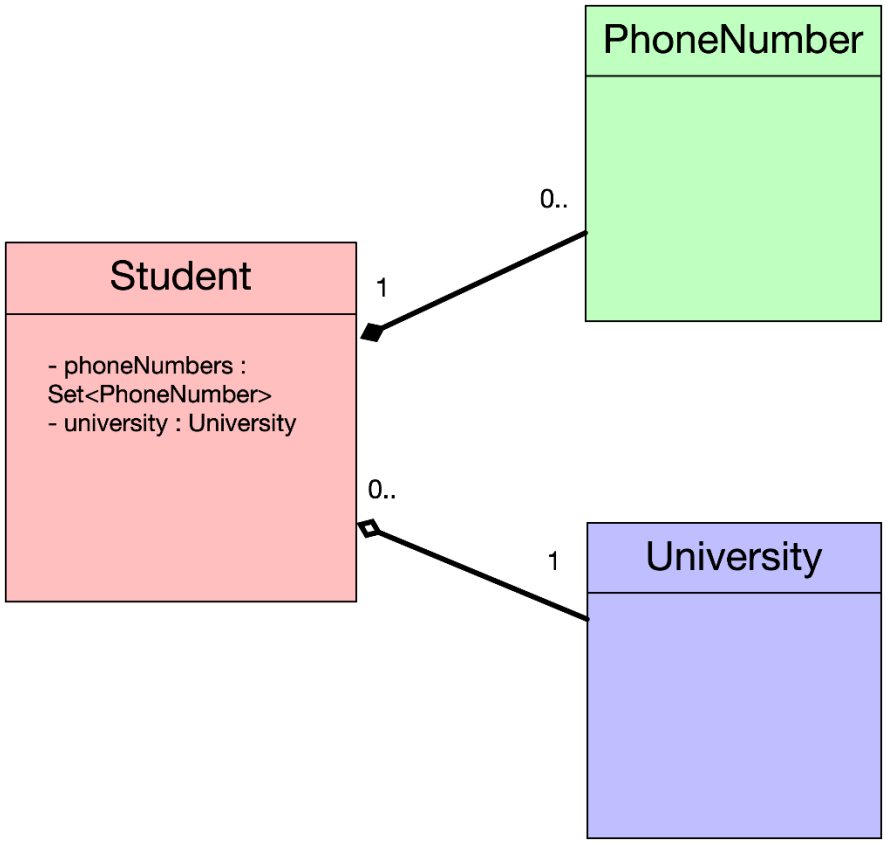
Mapping Hibernate

Cours n°4

Architecture Java EE - ENSG - M2 TSI

Relations entre les classes



Problématique des relations entre tables (clés étrangères)

- Comment faire en sorte que deux classes liés dans un programme objet soient correctement mappées dans la base ?
  - trouver le type de mapping ?
  - composition ou agrégation ?
  - uni ou bi directionnel ?

- Annotations possibles :
  - `@OneToMany`
  - `@ManyToOne`
  - `@ManyToMany`
  - `@OneToOne`

- Plus de détails ici : https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#associations

Exemple : une classe Etudiant

```
public class Student {
	private long id;
	private String firstName;
	private String lastName;

	private Address address;
	private Set<PhoneNumber> phoneNumber;
	private University university;
	// setters and getters
}
```

Uni-directionnel ou bi-directionnel

- Uni-directionnel : un objet A contient des objets d'un autre classe B, mais celle-ci ne contiennent pas de référence sur les objets de A.

- Bi-directionnel : dans le cas contraire

Exemple bi-directionnel:

```
public class PhoneNumber {
    private int id;
    private String phoneNumber;
    private Student student;
    ...
}
```

Embarquer un objet dans une entité
@Embedded

```
@Entity
@Table(name = "Students")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private long id;

    @Column(name ="first_name")
    private String firstName;

    @Column(name ="last_Name")
    private String lastName;

    @Embedded
    @AttributeOverrides({
    @AttributeOverride(name = "streetNumber",column = @Column(name = "street_number")),
    @AttributeOverride(name = "street",column = @Column(name = "street")),
    @AttributeOverride(name = "zipCode",column = @Column(name = "zip_code")),
    @AttributeOverride(name = "city",column = @Column(name = "city"))   })
    private Address address;
```

- @Embedded permet d'embarquer un objet dans une entité
  - La classe adresse devra être annoté @Embeddable
  - Une seule table sera créé avec les noms indiqués dans @AttributeOvverrides

@OneToMany

```
@OneToMany
private Set<PhoneNumber>
phonenumbers;
```

- Avec cette annotation Hibernate va créer une table de jointure
- Ce qui n'est pas la meilleure solution ici un numéro de telephone n'appartient qu'à un seul étudiant !

@JoinColumn

- Pour ne pas créer cette table, on utilise l'annotation @Joincolumn
  - creation d'une clé étrangère dans la table PhoneNumber avec la clé primaire de Student

```
@OneToMany
@JoinColumn(name = "Student_Id")
private Set<PhoneNumber>
phonenumbers;
```

Cascades

```
@OneToMany(cascade = CascadeType.ALL)
private Set<PhoneNumber> phonenumbers;
```

- Il existe plusieurs types de cascades selon le type de l'opération concernée CascadeType.REMOVE, CascadeType.MERGE, ...
- CascadeType.ALL concerne toutes opérations.
- Dans notre cas, on veut aussi que la destruction d'un Student en BD supprime aussi ses PhoneNumber.

Dans la documentation: @OneToMany

Java SQL

```java
@Entity(name = "Person")
public static class Person {

        @Id
        private Long id;

        private String name;

        @OneToMany(mappedBy = "author")
        private List<Book> books = new ArrayList<>();

        //Getters and setters are omitted for brevity

}

@Entity(name = "Book")
public static class Book {

        @Id
        private Long id;

        private String title;

        @NaturalId
        private String isbn;

        @ManyToOne
        private Person author;
```

```sql
create table Book (
    id bigint not null,
    isbn varchar(255),
    title varchar(255),
    author_id bigint,
    primary key (id)
)

create table Person (
    id bigint not null,
    name varchar(255),
    primary key (id)
)

alter table Book
    add constraint UK_u31e1frmjp9mxf8k8tmp990i unique (isbn)

alter table Book
    add constraint FKrxrgiajod1le3gii8whx2doie
    foreign key (author_id)
    references Person
```

@ManyToOne

```
@ManyToOne
@JoinColumn(name = "university_id")
private University university;
```

- @ManyToOne permet de relier un classe qui est liée par une relation hiérarchique
  - La relation peut être unidirectionnelle ou bidirectionnelle
  - Il n'y aura pas de cascade ici : on ne veut pas supprimer l'université si on supprimer un Student !

@OneToOne et @ManyToMany

- Le mapping @OneToOne réalise une colonne de jointure dans la table de la classe.

- Le @ManyToMany fait toujours une table de jointure.

Mapping bi-directionnel (1/3)
Exemple

```
public class Student {

public class Student {
    private long id;
    private String firstName;
    private String lastName;
    private Set<PhoneNumber> phoneNumbers;
    ...
}

public class PhoneNumber {
    private long id;
    private String phoneNumber;
    private Student owner;
    ...
}
```

Mapping bi-directionnel (2/3)
Exemple annoté

```
@Entity
public class Student {
    @Id
    @GeneratedValue
    private long id;
    private String firstName;
    private String lastName;
    @OneToMany(cascade = CascadeType.ALL,mappedBy="owner")
    private Set<PhoneNumber> phoneNumbers;
}
```

```
@Entity
public class PhoneNumber {
    @Id
    @GeneratedValue
    private long id;
    private String phoneNumber;

     //chargement LAZY (paresseux) (EAGER par défaut)
    @ManyToOne(fetch = FetchType.LAZY)
    private Student owner;
}
```

Mapping bi-directionnel (3/3)

- Un seul des cotés est responsable de la relation.
  - C'est toujours le @ManyToOne
  - Le coté @OneToMany qui n'est pas responsable doit faire référence au responsable avec un mappedBy.

- Pour une @ManyToMany, il faudra choisir un coté comme responsable (non traité ici)

Fichier de configuration Hibernate

- Il faut ajouter les « mapping » dans `hibernate.cfg.xml`

```
<mapping class ="tsi.ensg.jee.hibernate.Student"/>
<mapping class ="tsi.ensg.jee.hibernate.University"/>
<mapping class ="tsi.ensg.jee.hibernate.PhoneNumber"/>
```

- Chaque classe indiquée devra elle-même être annotée selon les besoins

Héritage avec Hibernate

http://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#entity-inheritance

Héritage

- Les bases de données relationnelles ne fournissent pas de support pour l'héritage.
  - ce n'est pas tout à fait vrai pour PostgreSQL

- Hibernate fournit plusieurs stratégies pour résoudre le problème de la traduction de l'héritage d'un programme orienté-objet vers une base de données relationnelle

Les stratégies

- **MappedSuperclass**
  - L'héritage n'est visible que dans le programme objet
  - La base de données contient une table pour chaque sous-classe qui contient tous les attributs

```
@MappedSuperclass
```

- **Single table**
  - Toute la hiérarchie est incluse dans une seule table.

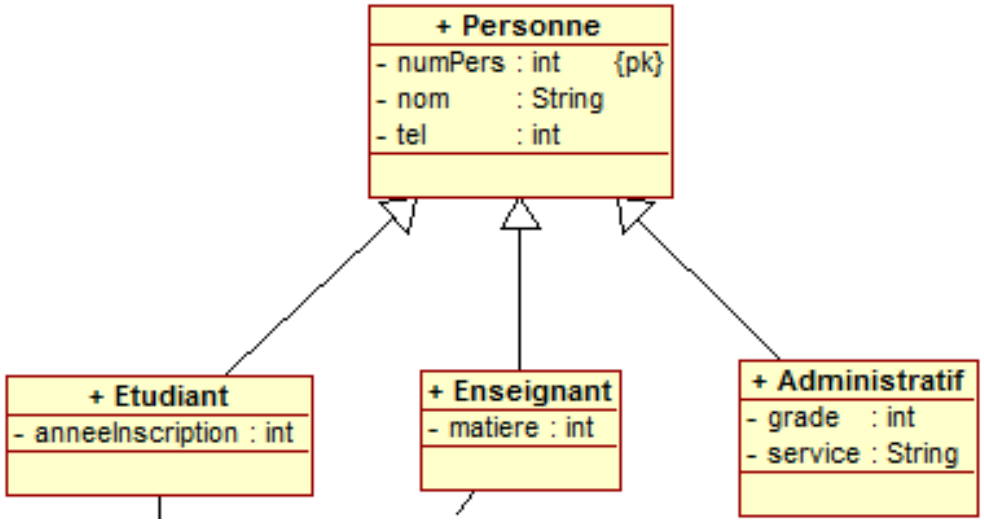
```
@Entity(name = "nom_table")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
```

- **Joined table**
  - Toutes les classes et les sous-classes ont leur propre table dans la base de données. Des clés étrangères sont introduites.

```
@Entity(name = "nom_table")
@Inheritance(strategy = InheritanceType.JOINED)
```

- **Table per class**
  - Toutes les classes et les sous-classes ont leur propre table qui contient toutes les attributs hérités.
  - `@Entity(name = "Account")`
  - `@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)`

Exemples : stratégies de traduction de l'héritage



Joined Table

Création d'une relation par classe dérivée
Solution la plus courante pour limiter le nombre de champ pouvant prendre la valeur NULL

```
PERSONNE(numPers, nom, tel)
ETUDIANT(#numPers, anneeInscription)
ENSEIGNANT(#numPers, matiere)
ADMINISTRATIF(#numPers, grade, service)
```

MappedSuperClass

Duplication de tous les champs dans les classes dérivées.
Possible s'il n'existe pas d'association qui pointe vers la classe principale.

```
ETUDIANT(numPers, anneeInscription, nom, tel)
ENSEIGNANT(numPers, matiere, nom, tel)
ADMINISTRATIF(numPers, grade, service, nom, tel)
```

Exemple de code Java -> SQL

```java
@MappedSuperclass
public static class Account {

        @Id
        private Long id;

        private String owner;

        private BigDecimal balance;

        private BigDecimal interestRate;

        //Getters and setters are omitted for brevity

}

@Entity(name = "DebitAccount")
public static class DebitAccount extends Account {

        private BigDecimal overdraftFee;

        //Getters and setters are omitted for brevity

}

@Entity(name = "CreditAccount")
public static class CreditAccount extends Account {

        private BigDecimal creditLimit;

        //Getters and setters are omitted for brevity

}
```

```sql
CREATE TABLE DebitAccount (
    id BIGINT NOT NULL ,
    balance NUMERIC(19, 2) ,
    interestRate NUMERIC(19, 2)
    owner VARCHAR(255) ,
    overdraftFee NUMERIC(19, 2)
    PRIMARY KEY ( id )
)

CREATE TABLE CreditAccount (
    id BIGINT NOT NULL ,
    balance NUMERIC(19, 2) ,
    interestRate NUMERIC(19, 2)
    owner VARCHAR(255) ,
    creditLimit NUMERIC(19, 2) ,
    PRIMARY KEY ( id )
)
```

Liens

Documentation Hibernate

- http://docs.jboss.org/hibernate/orm/5.4/quickstart/html_single/

- http://hibernate.org/orm/documentation/5.4/