


Conception en UML, Architecture n-tiers, par l'exemple

Utilisation de php 5, Mysql, Html, css, ...

Inspiré de UML2 par la pratique

M. Blay-Fornarino

Les codes sont disponibles sur le site web

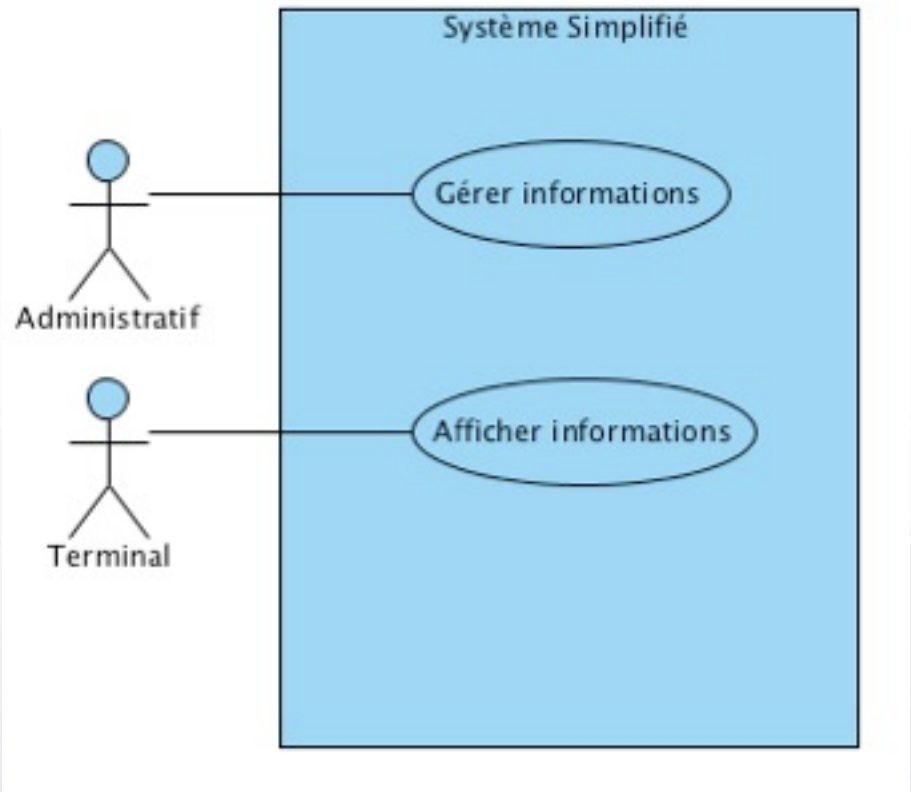
Bibliographie

- «Why MVC is not an application architecture» Stefan Priebsch, the PHP.cc ZendCon 2010
- Developing Web Applications with PHP, RAD for the World Wide Web,

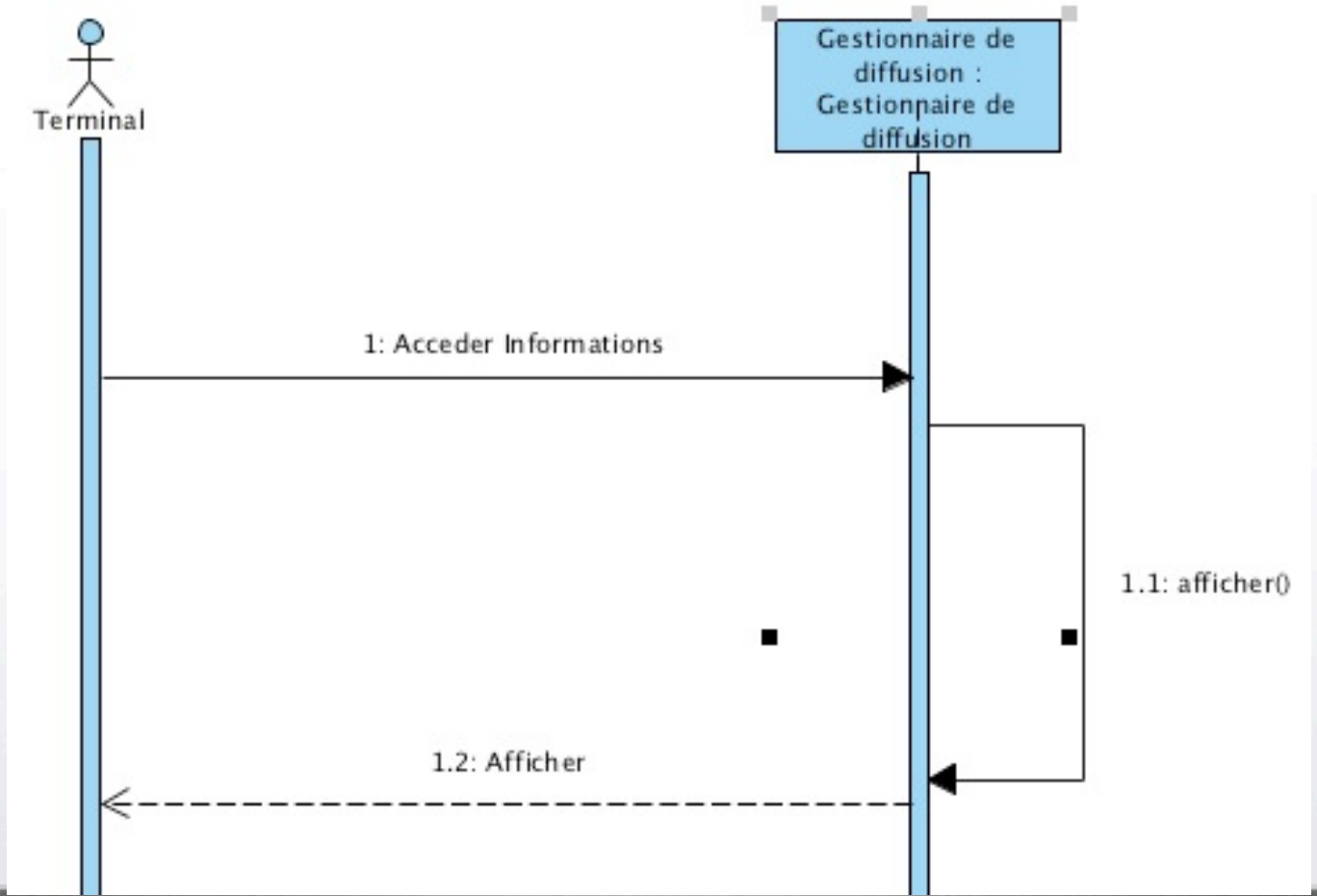

Approche

Analyse

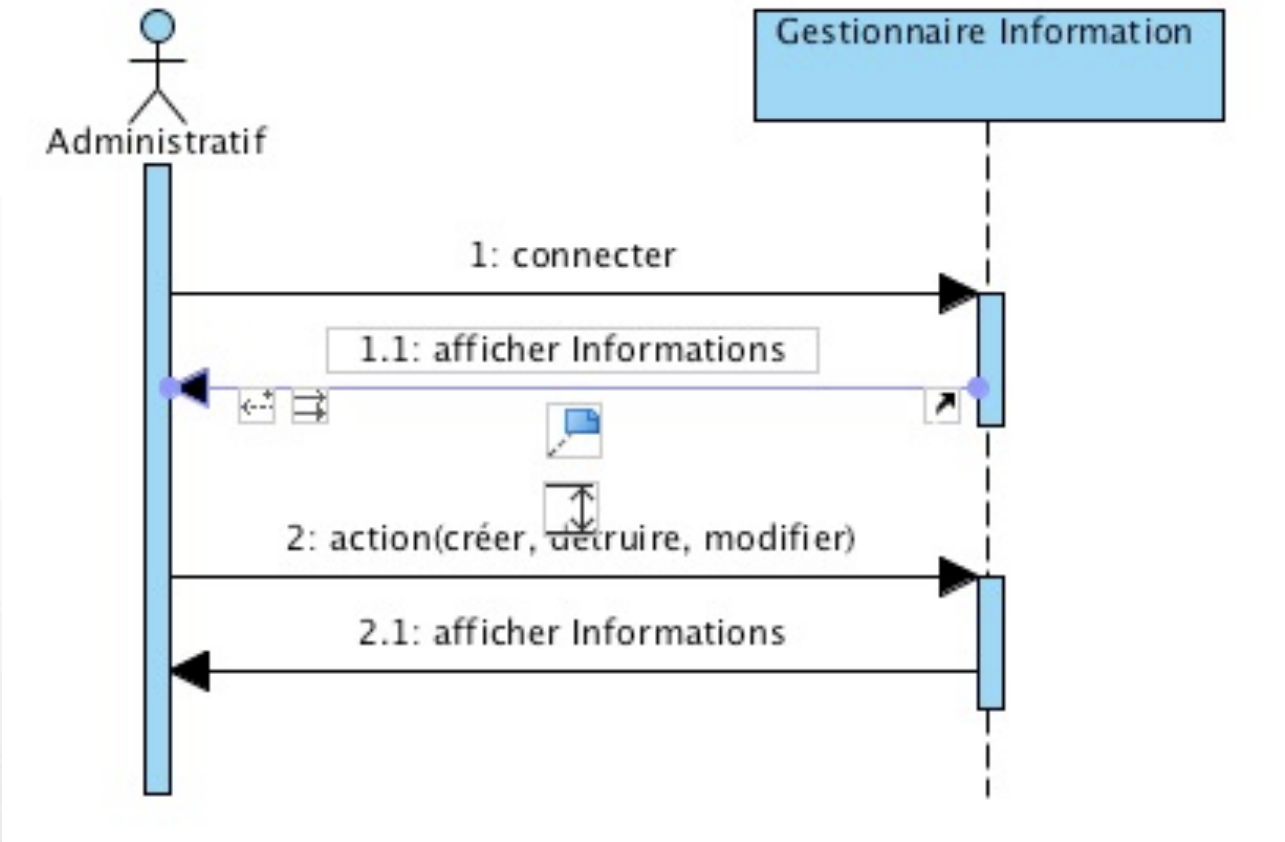
Diagramme de Use-cases

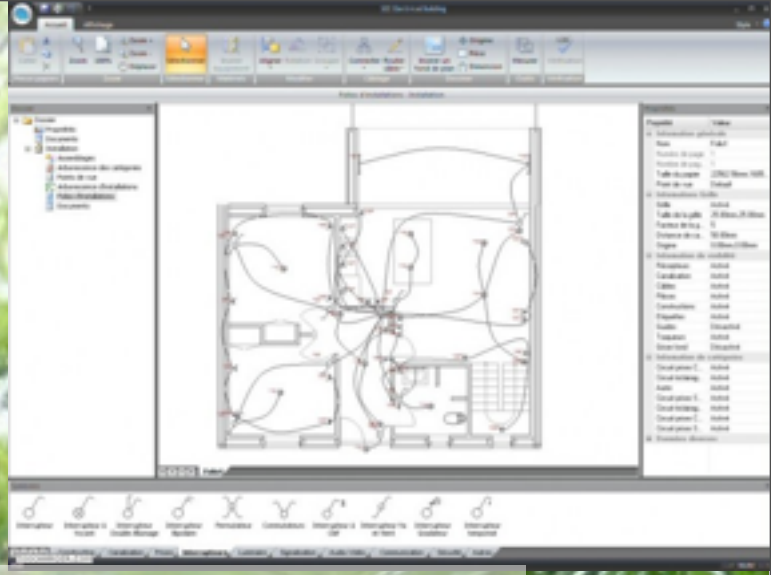


Afficher Informations : niveau Analyse



Gérer Informations : niveau Analyse (0)





Conception

Focus

-> Architecture

-> Classes

-> Données


Conception
Focus
-> Architecture
-> Données

Choix d'Architecture

Présentation

Logique applicative

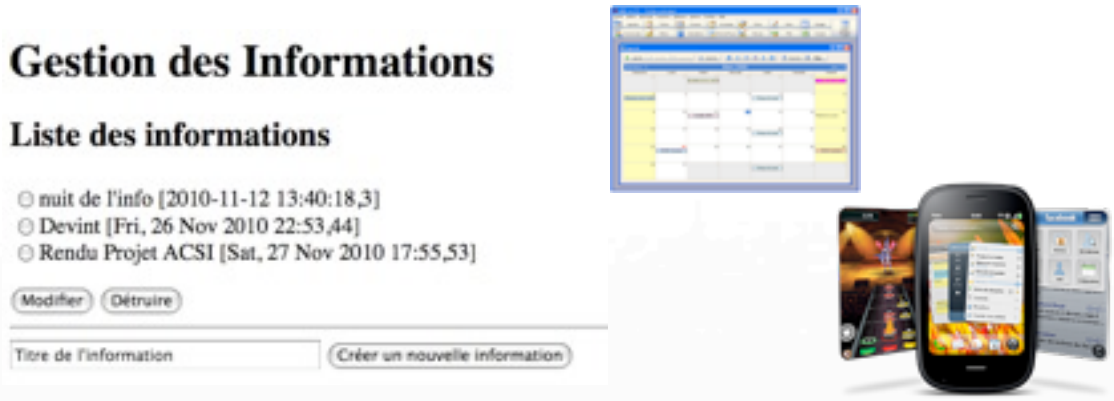
Gérer les informations

Stockage

```
CREATE TABLE `information` (
 `titre` varchar(20) NOT NULL,
 `date` varchar(22) NOT NULL,
`identifiant` int(11) NOT NULL auto_increment,
 PRIMARY KEY  (`identifiant`))
```

Choix d'Architecture

Présentation



Logique applicative

Gérer les informations

Stockage

```
CREATE TABLE `information` (
 `titre` varchar(20) NOT NULL,
 `date` varchar(22) NOT NULL,
`identifiant` int(11) NOT NULL auto_increment,
 PRIMARY KEY  (`identifiant`))
```

ORACLE

Choix d'Architecture

Présentation

clickOn()

IHM

créer Information

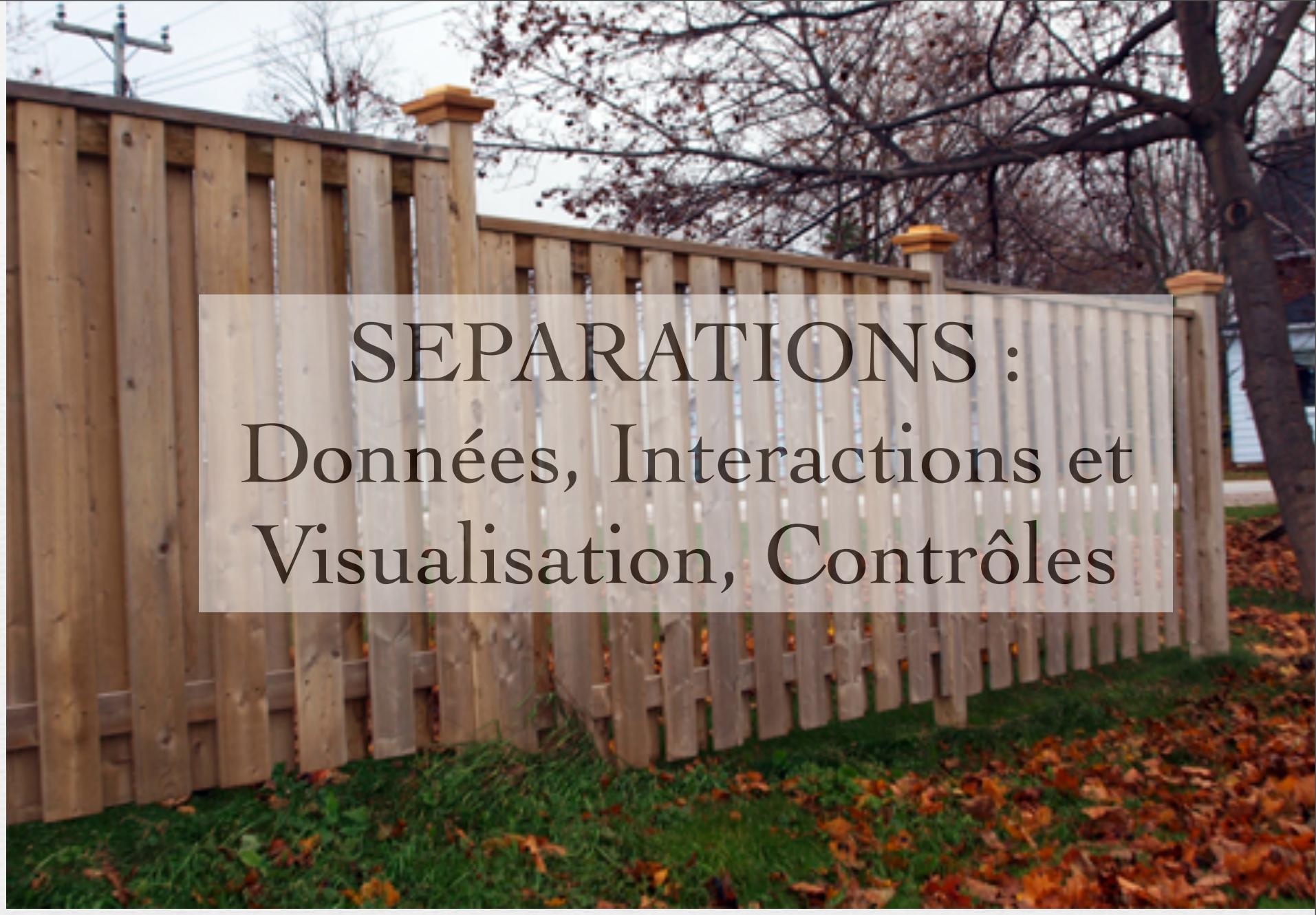
Contrôleur

Logique applicative

Gérer les informations : métier

Stockage

```
CREATE TABLE `information` (
 `titre` varchar(20) NOT NULL,
 `date` varchar(22) NOT NULL,
`identifiant` int(11) NOT NULL auto_increment,
 PRIMARY KEY  (`identifiant`))
```


SEPARATIONS :
Données, Interactions et
Visualisation, Contrôles

Modèle-Vue-Controleur (MVC)

Controller

contrôleur: chef d'orchestre

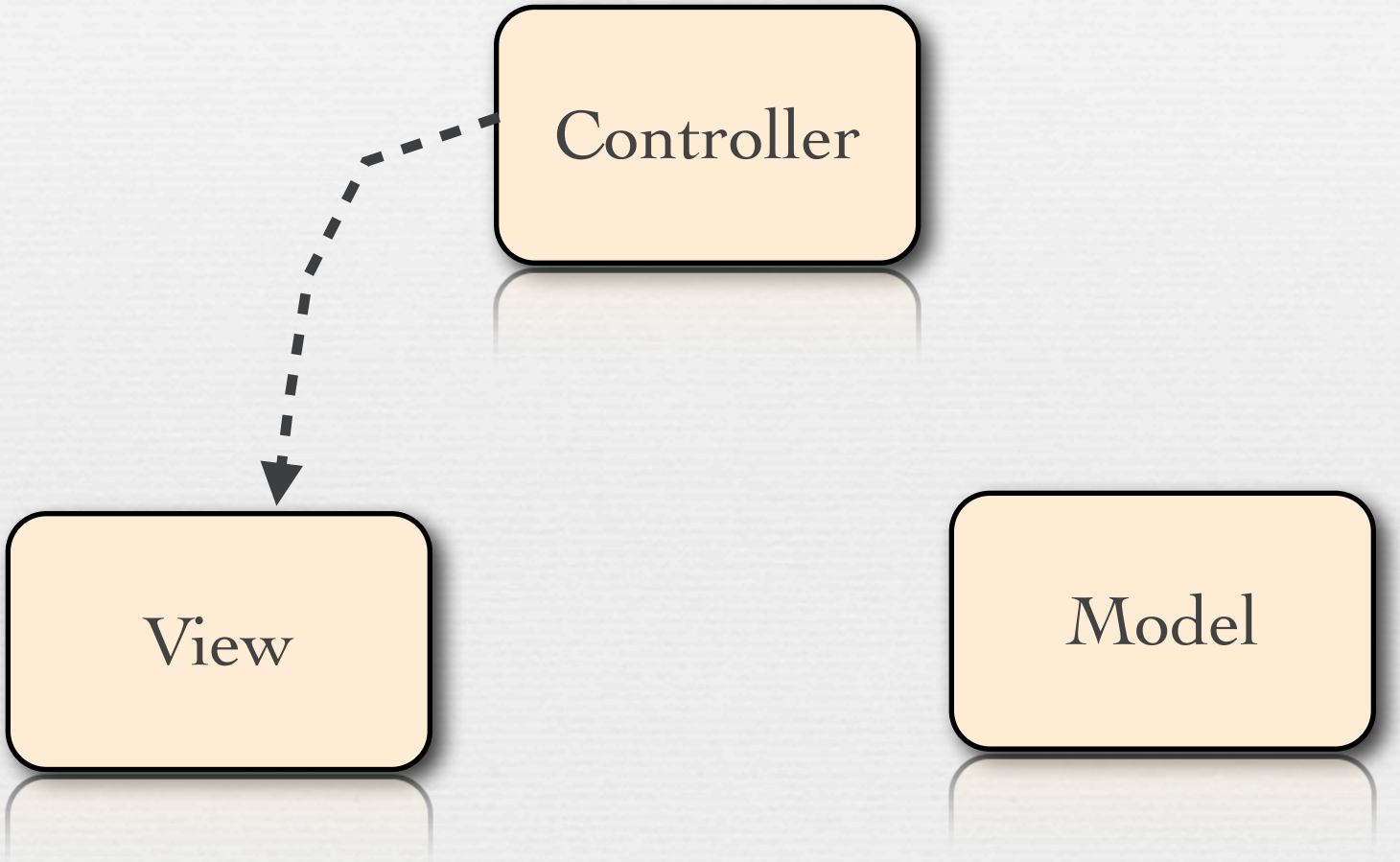
View

La vue: présentée à l'utilisateur

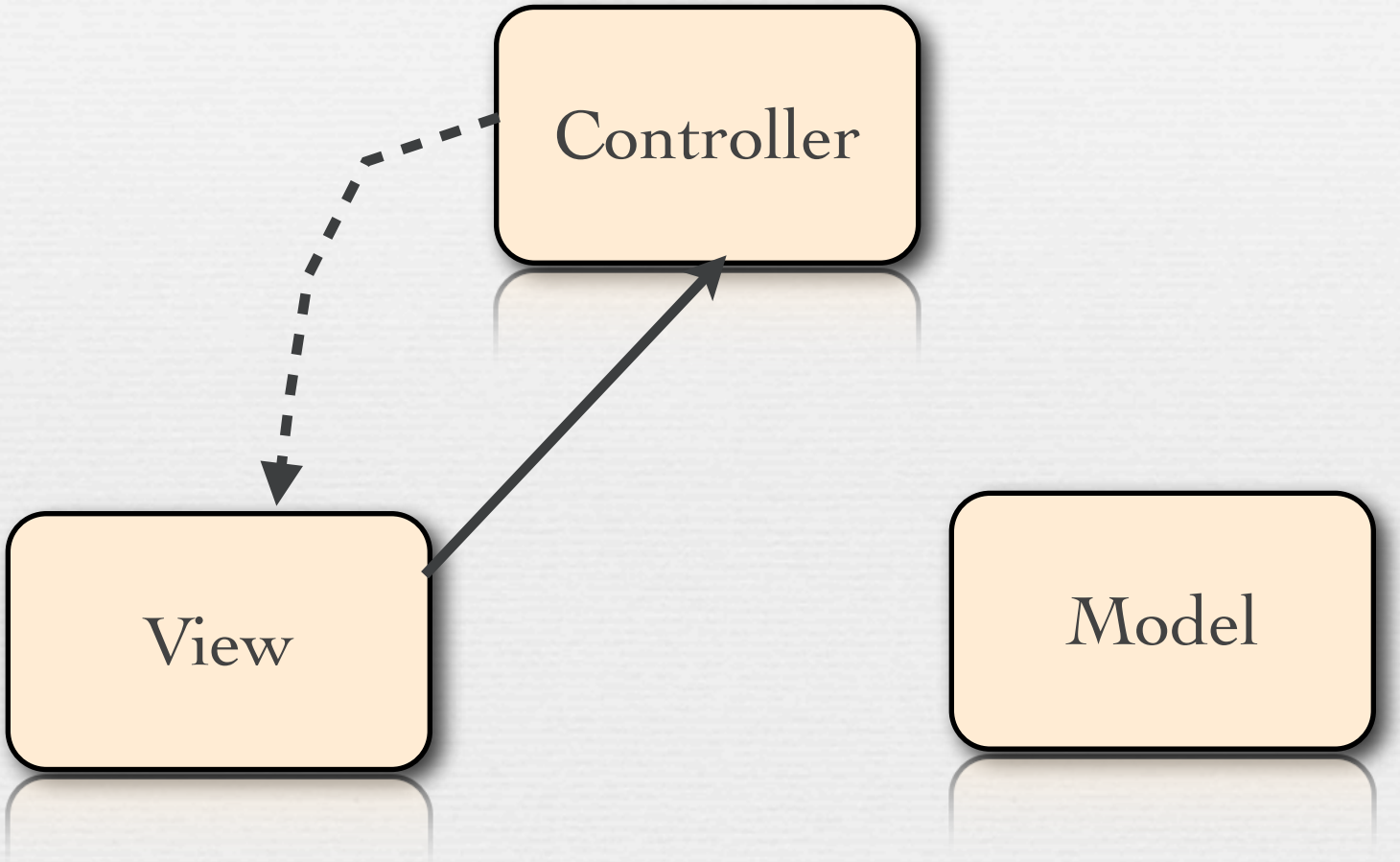
Model

Le modèle: les données indépendantes

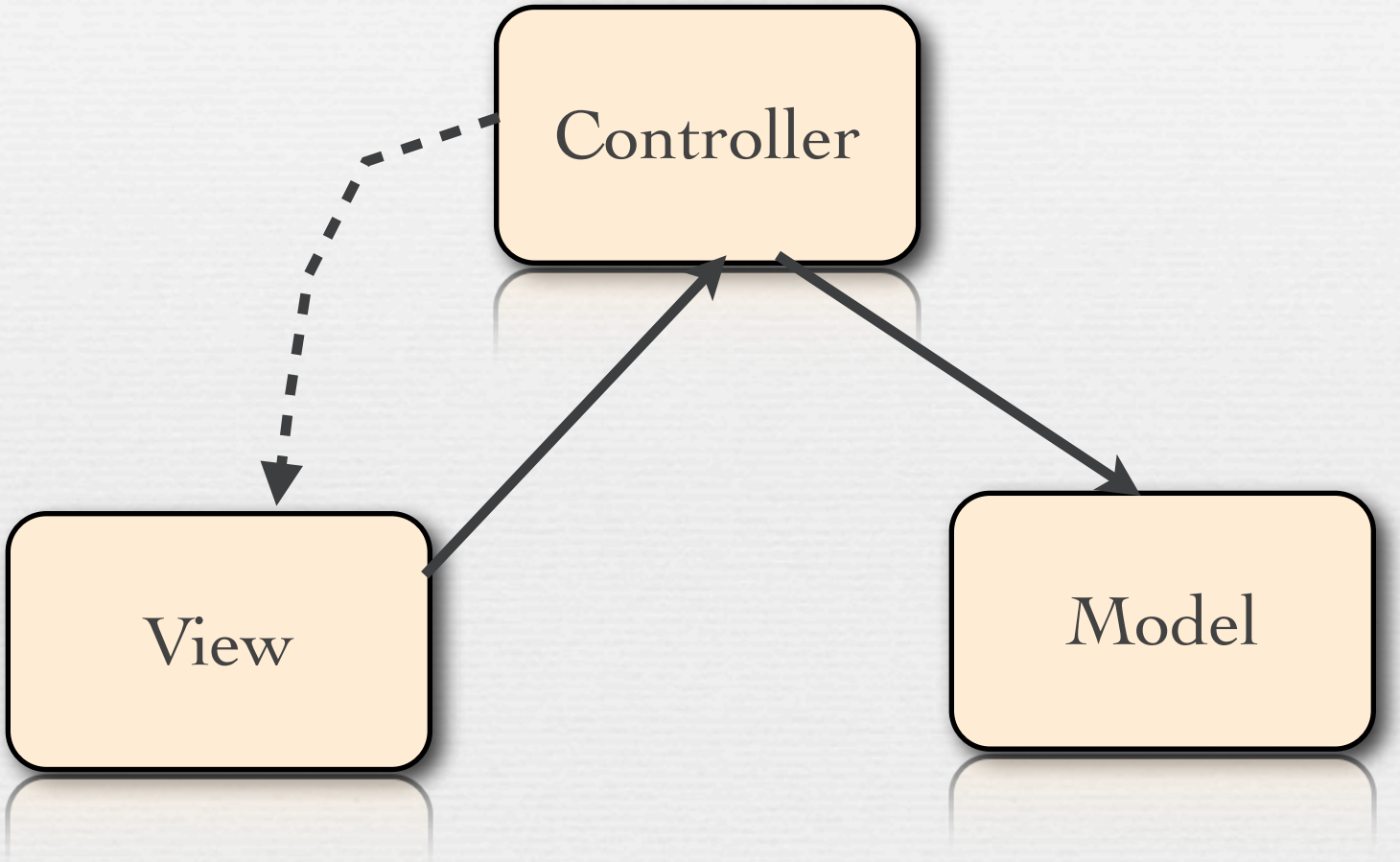
Controleur observe la vue



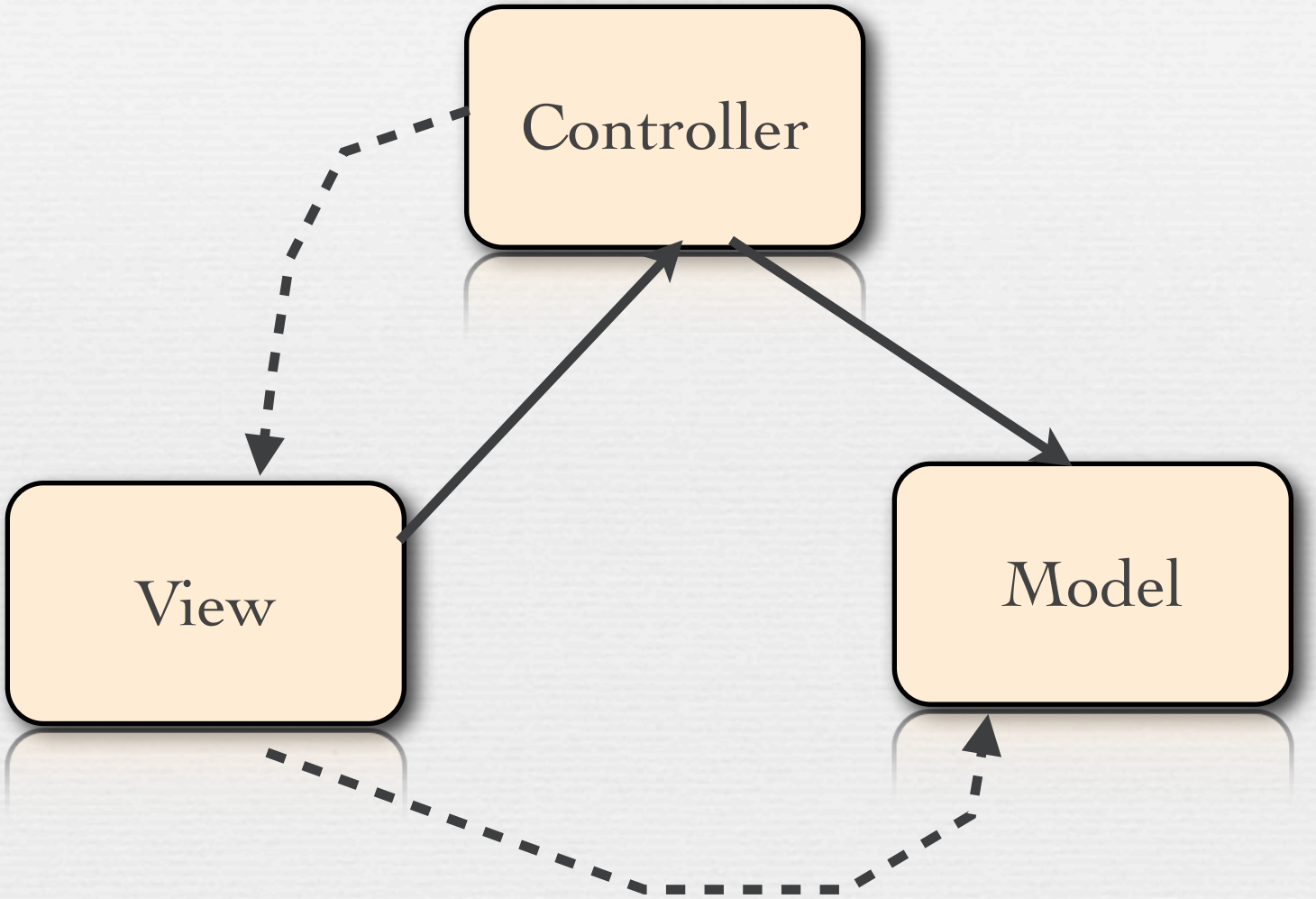
Contrôleur récupère les données de la vue



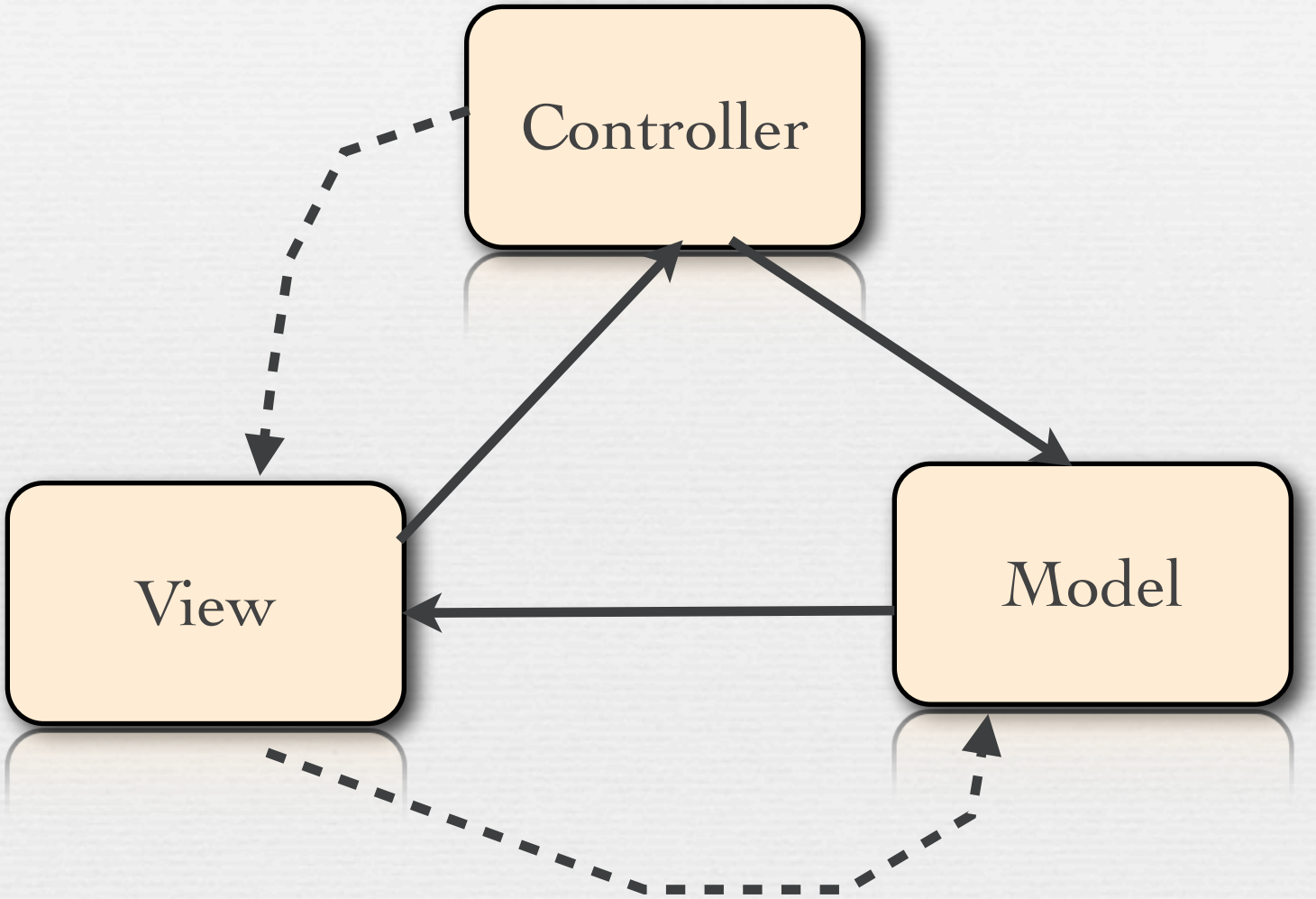
Contrôleur modifie le modèle



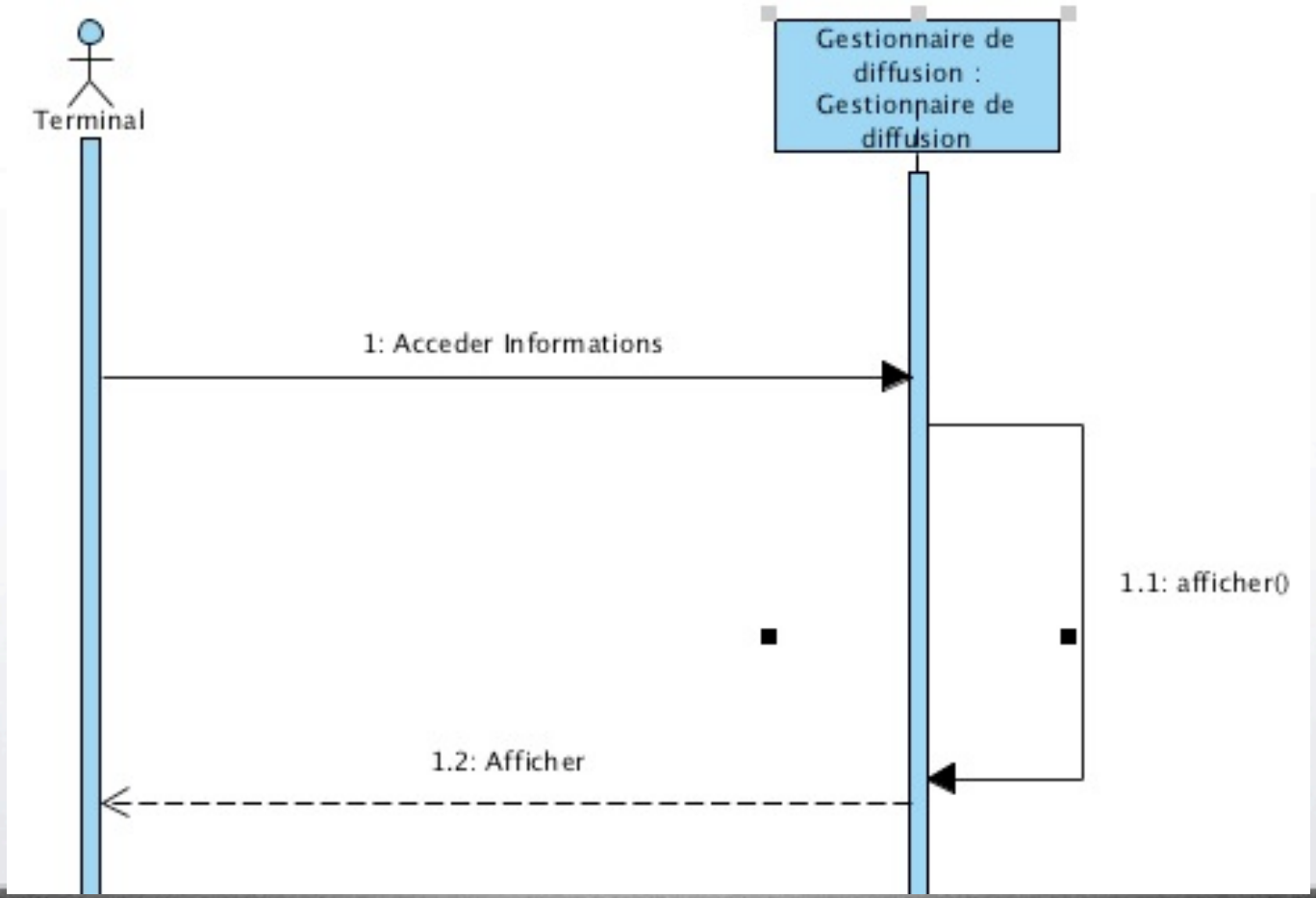
Vue observe le modèle



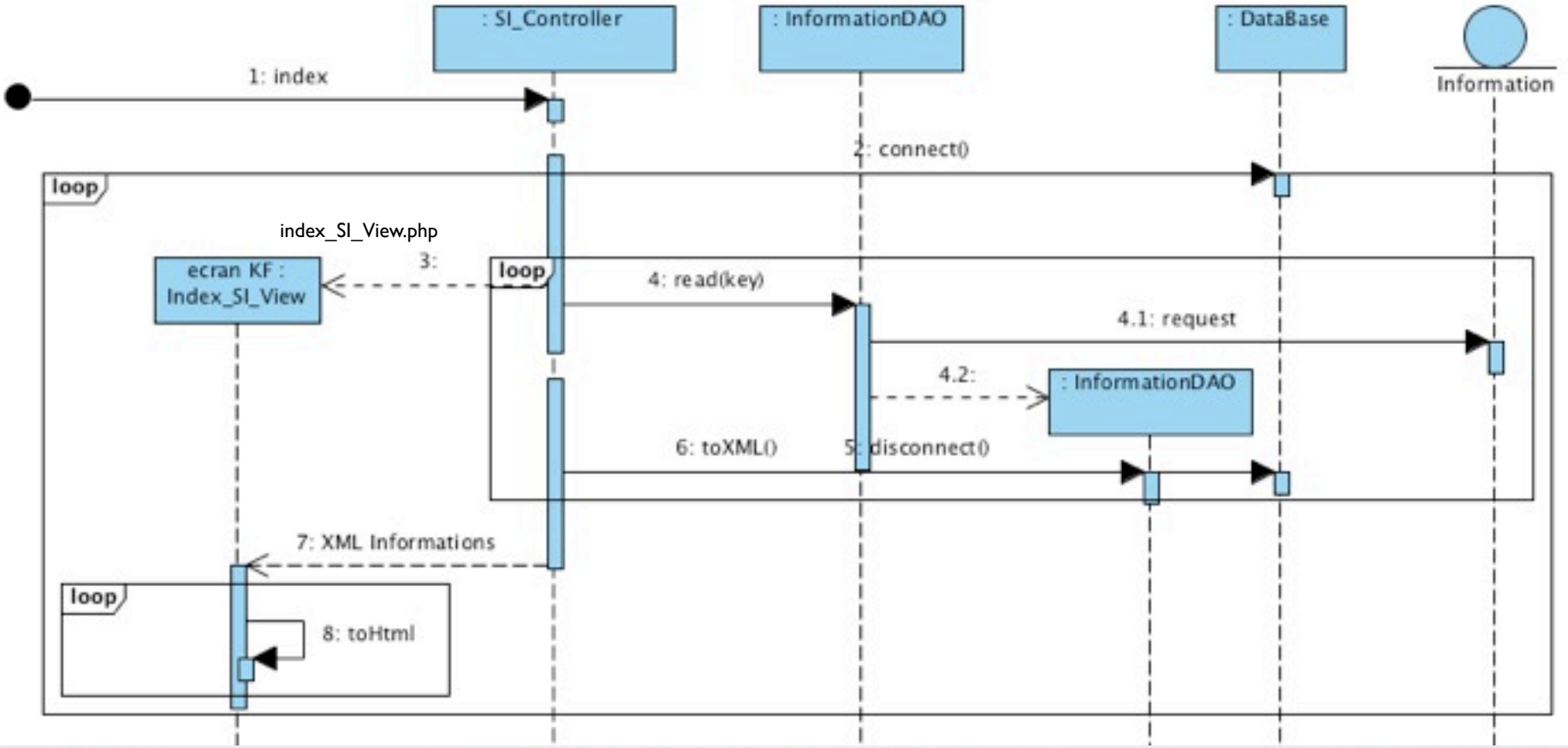
Vue récupère les données du modèle



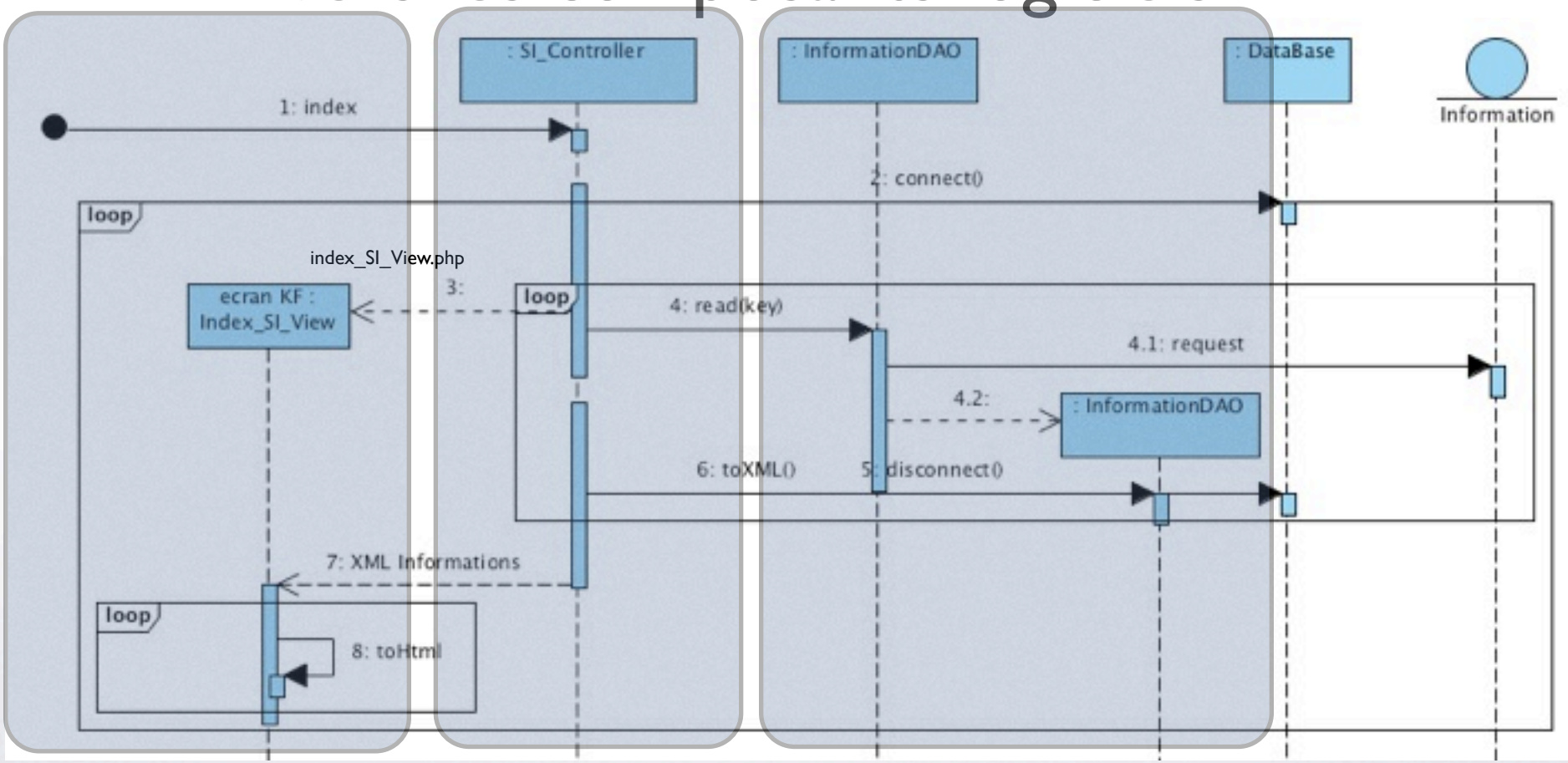
Afficher Informations : niveau Analyse



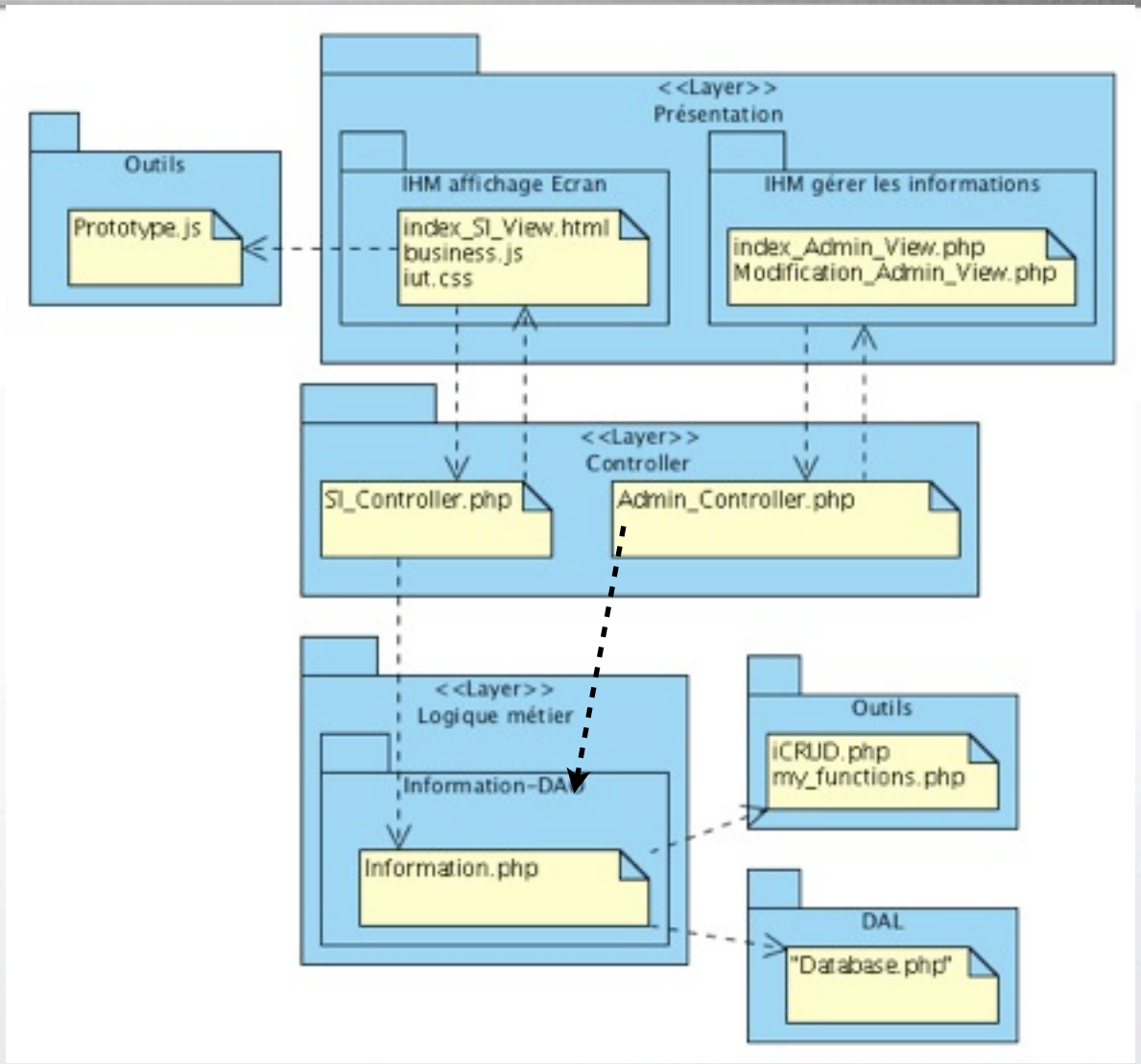
Afficher Informations : niveau Conception



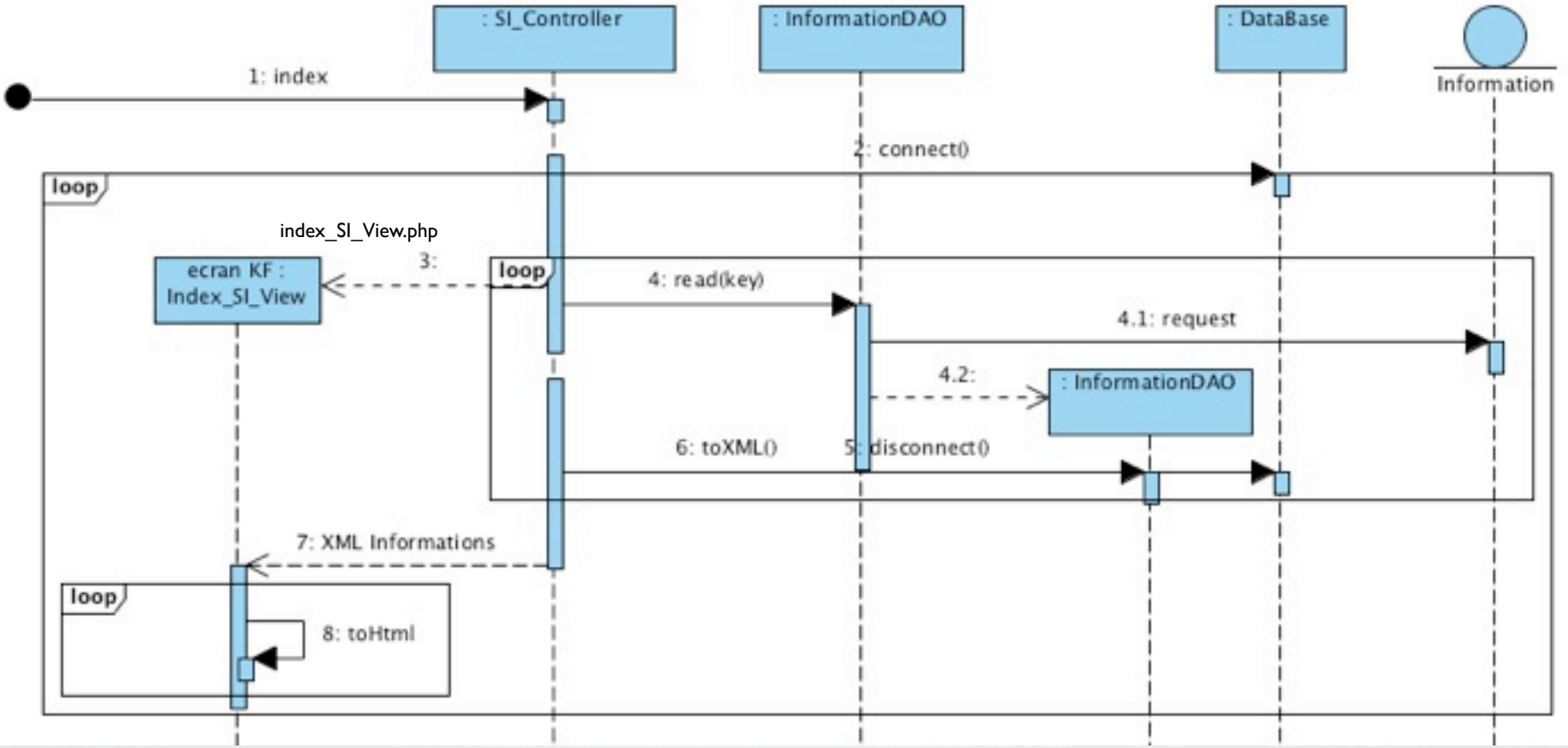
Vers les composants logiciels



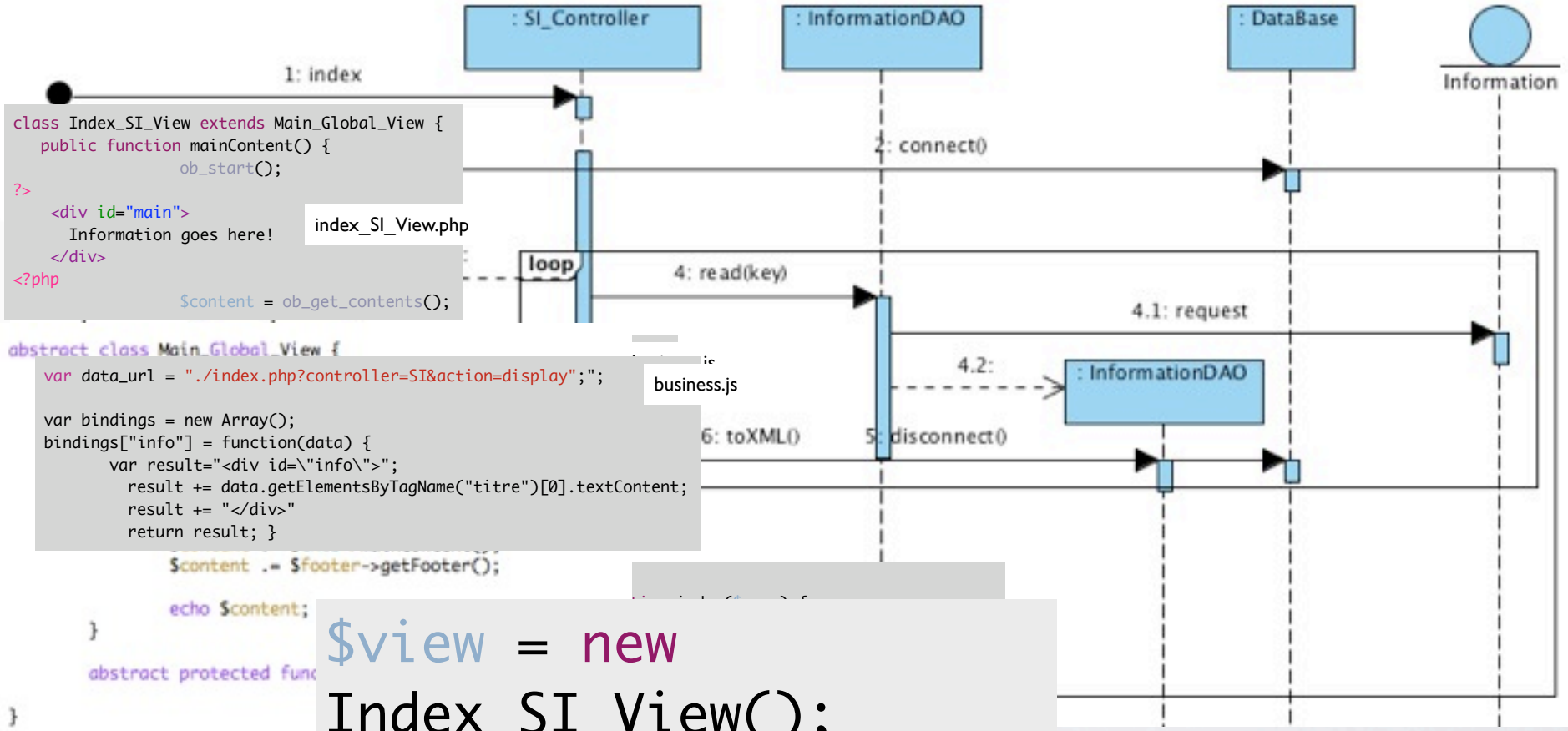
Architecture en couches & Fichiers



Afficher Informations : niveau Conception



Afficher Informations : niveau Conception




```
class Index_SI_View extends Main_Global_View {
   public function mainContent() {
                ob_start();
?>
    <div id="main">
      Information goes here!
    </div>
<?php
                $content = ob_get_contents();
```

index_SI_View.php

```
abstract class Main_Global_View {
```

```
var data_url = "./index.php?controller=SI&action=display";";

var bindings = new Array();
bindings["info"] = function(data) {
       var result="<div id=\"info\">";
         result += data.getElementsByTagName("titre")[0].textContent;
         result += "</div>"
         return result; }
```

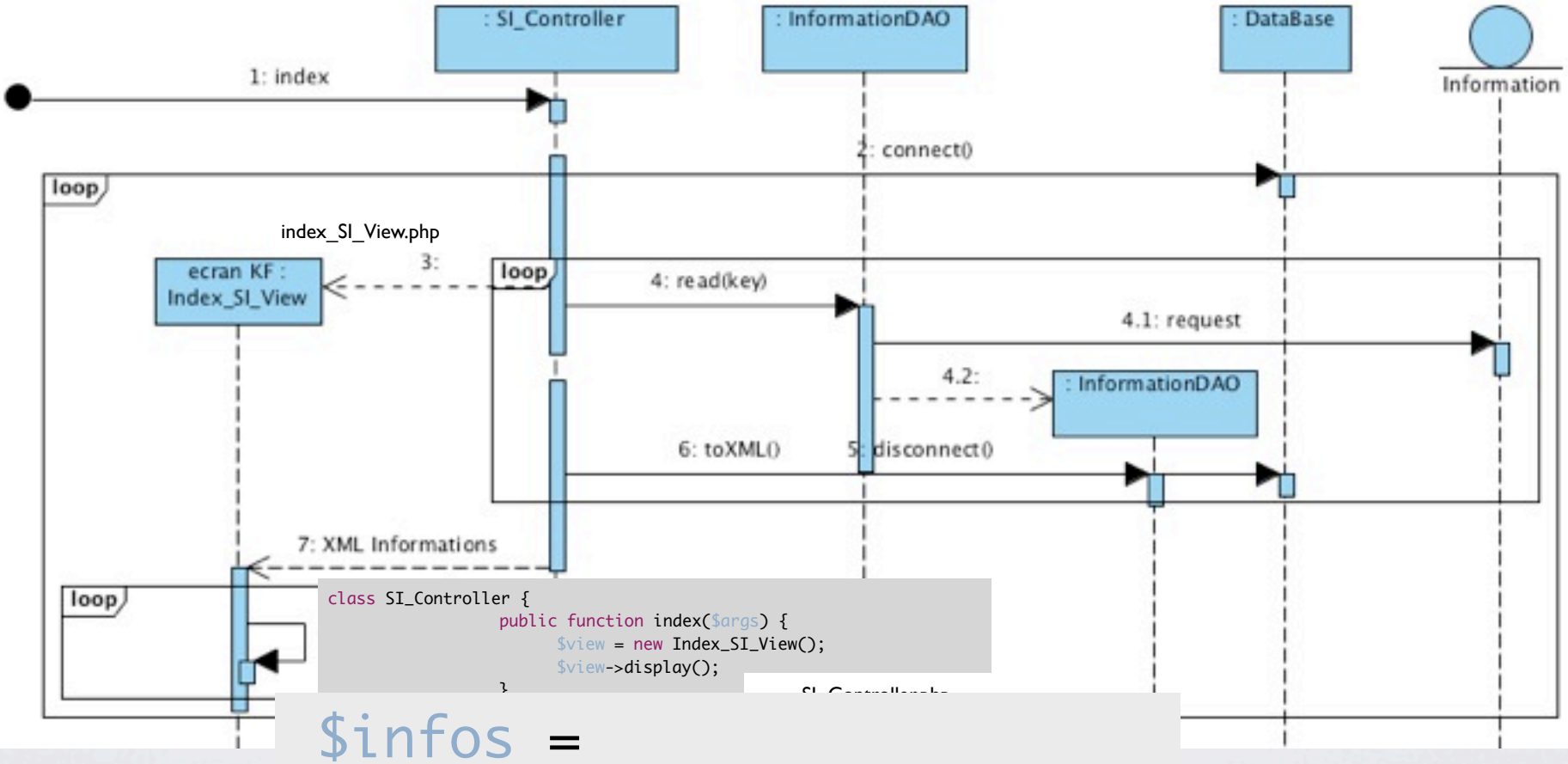
business.js

```
            $content .= $footer->getFooter();

            echo $content;
    }

    abstract protected func
}
```

```
$view = new Index_SI_View();
        $view->display
```

```
}
$res=$res.'</data>';

    echo $res;
}
```

Afficher Informations : niveau Conception



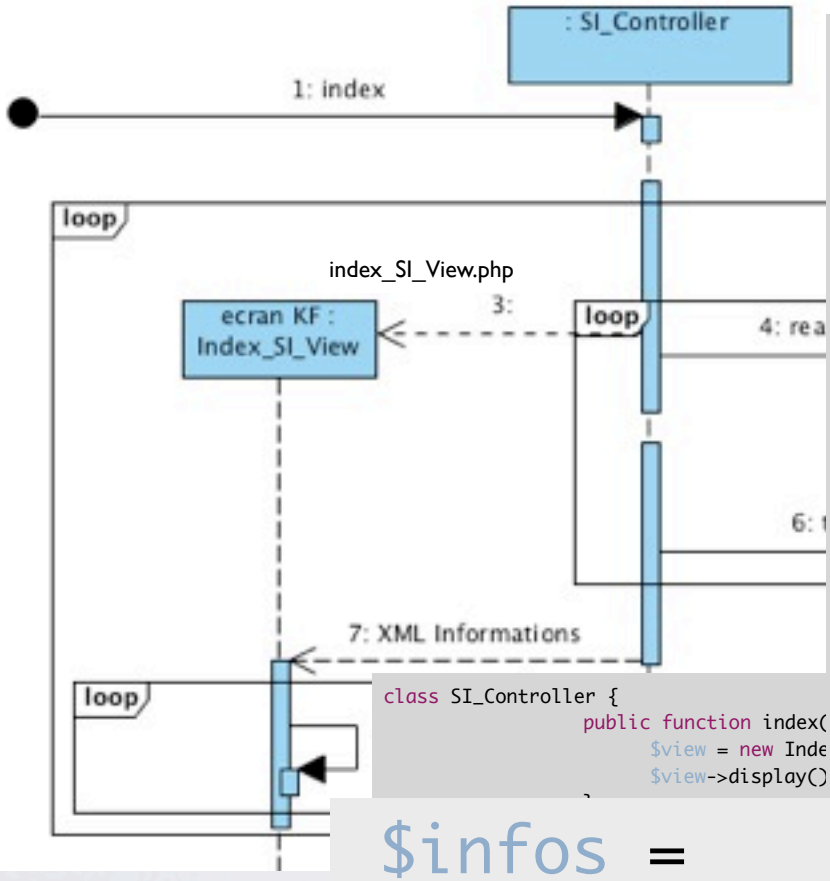

```
class SI_Controller {
                public function index($args) {
                      $view = new Index_SI_View();
                      $view->display();
                }
```

```
$infos =
Information::findAll();
```

```
                }
                $res=$res.'</data>';

                    echo $res;
                }
```

Information
<=>InformationDAO

Afficher Informations : niveau Conception



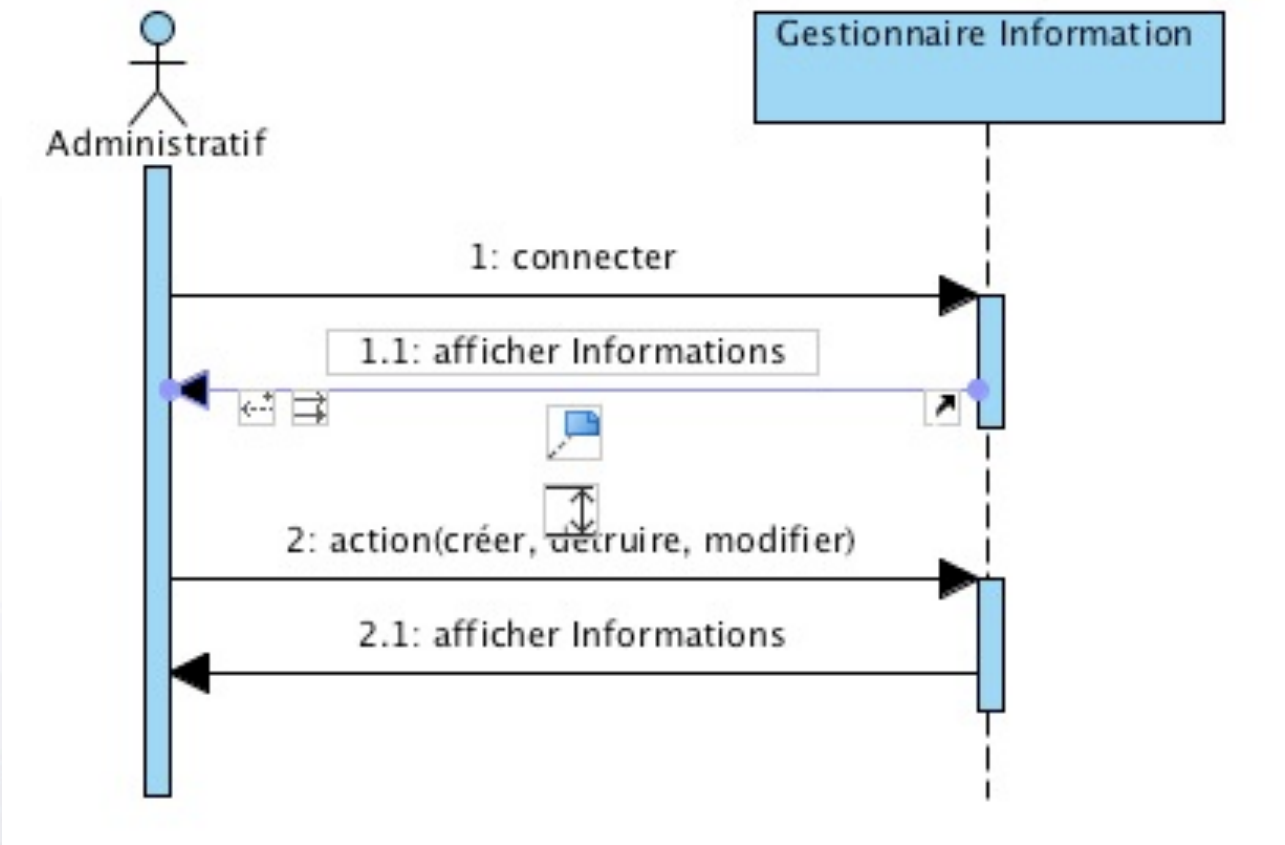
Information.php

```
class Information implements iCRUD
{
  private $_id;
  private $_titre;
  private $_date;
  public static function findAll() {
   $informations = array();
   Database::connect();
    $query = "SELECT * FROM information";
    $res = mysql_query($query);
    while($line = mysql_fetch_assoc($res))
{
        $titre = $line["titre"];
       $date = $line["date"];
       $key = $line["identifiant"];
                                    nformation($titre,
```

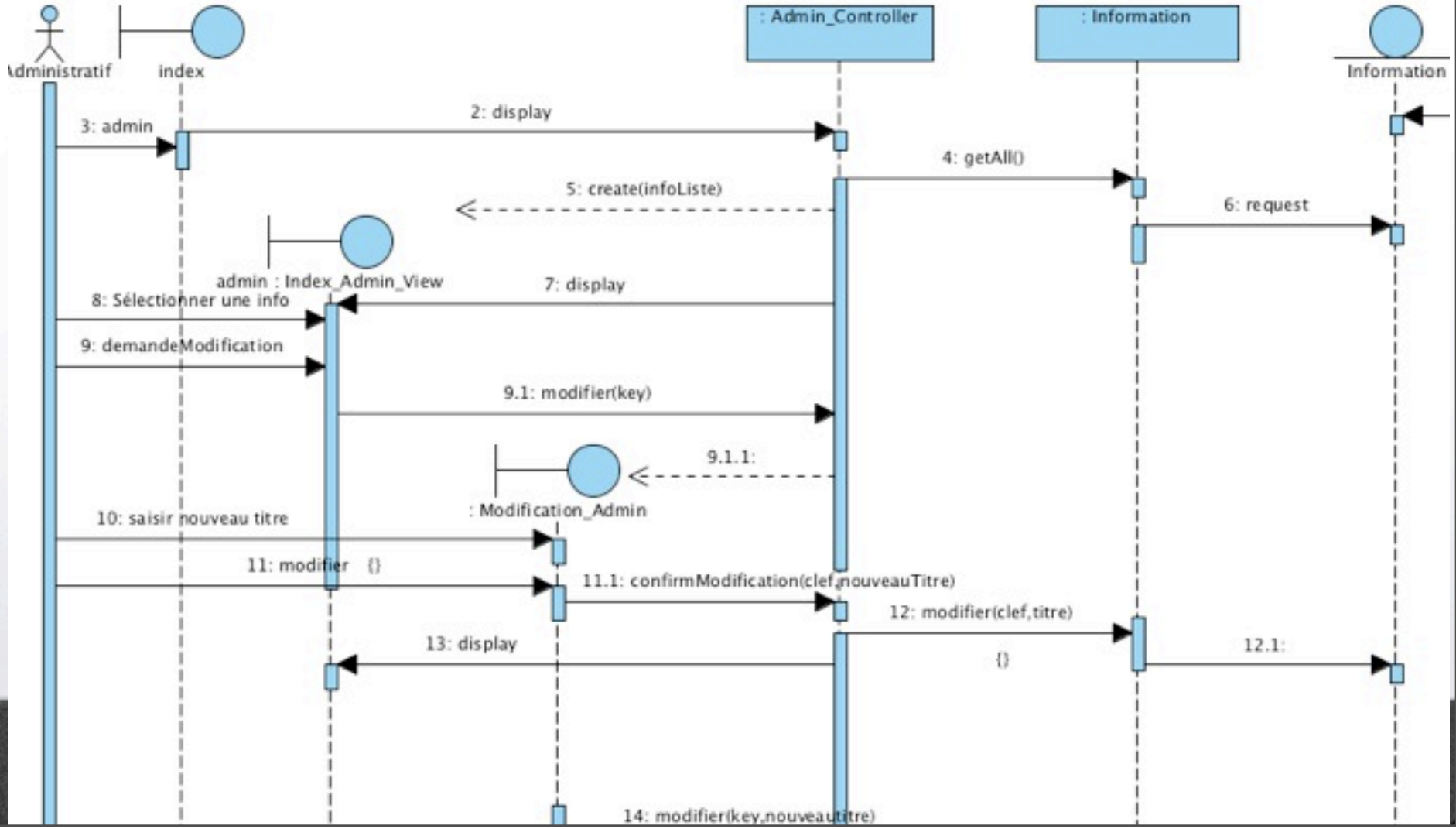
```
class SI_Controller {
                public function index(
                    $view = new Inde
                    $view->display()
                $res = $res.$tmpinformation->toXML() ;
            }
            $res=$res.'</data>';

                echo $res;
            }
```

```
$infos =
Information::findAll();
```

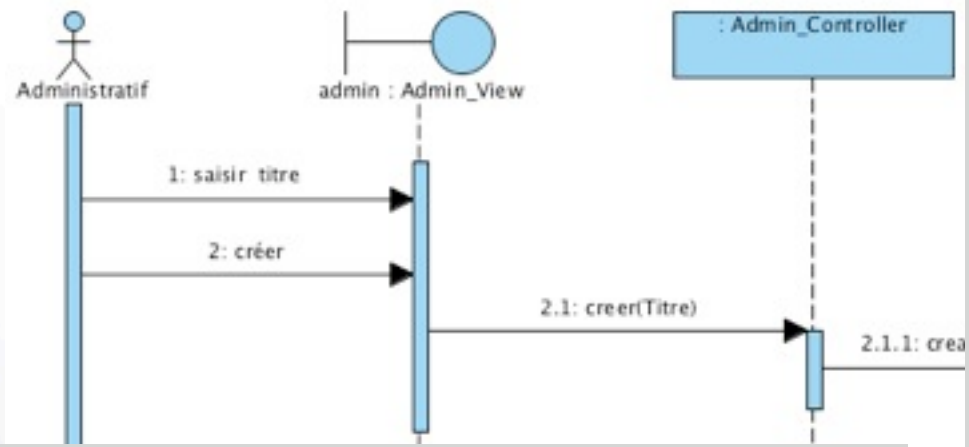
Gérer Informations : niveau Analyse (0)



modifier Information : niveau Conception



Créer Information : niveau Conception

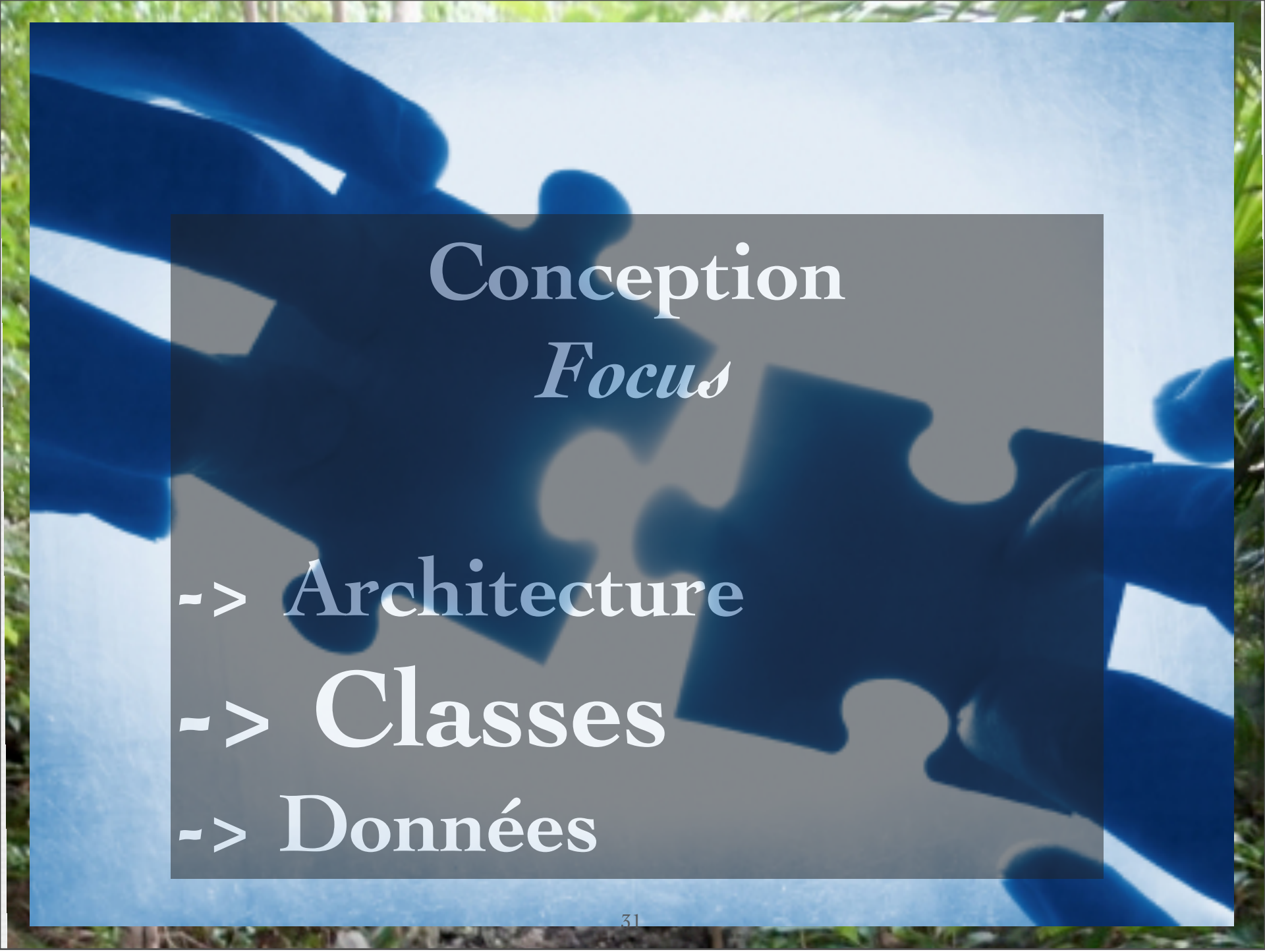


admin_View

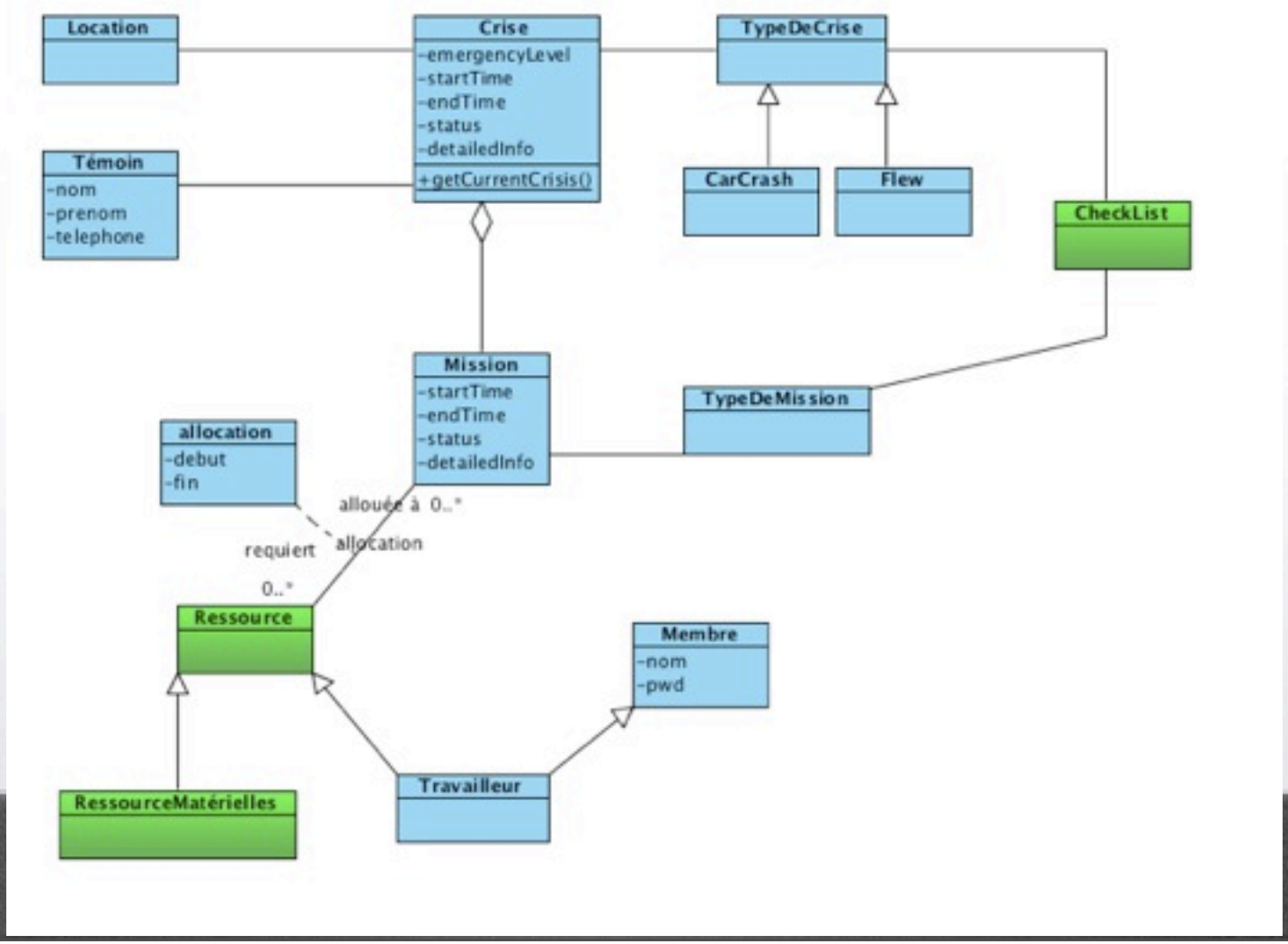
```
class Index_Admin_View extends Main_Global_View {

     private $infosListe;

    public function Index_Admin_View($args) {
                $this->infosListe = $args['infosListe'];
           }...
<h1>Gestion des Informations</h1>
  <h2> Liste des informations </h2>
  <form id="infoModifForm" name="infoModifForm" method="post" action="./">
  <p>
```

Admin_Controller.php

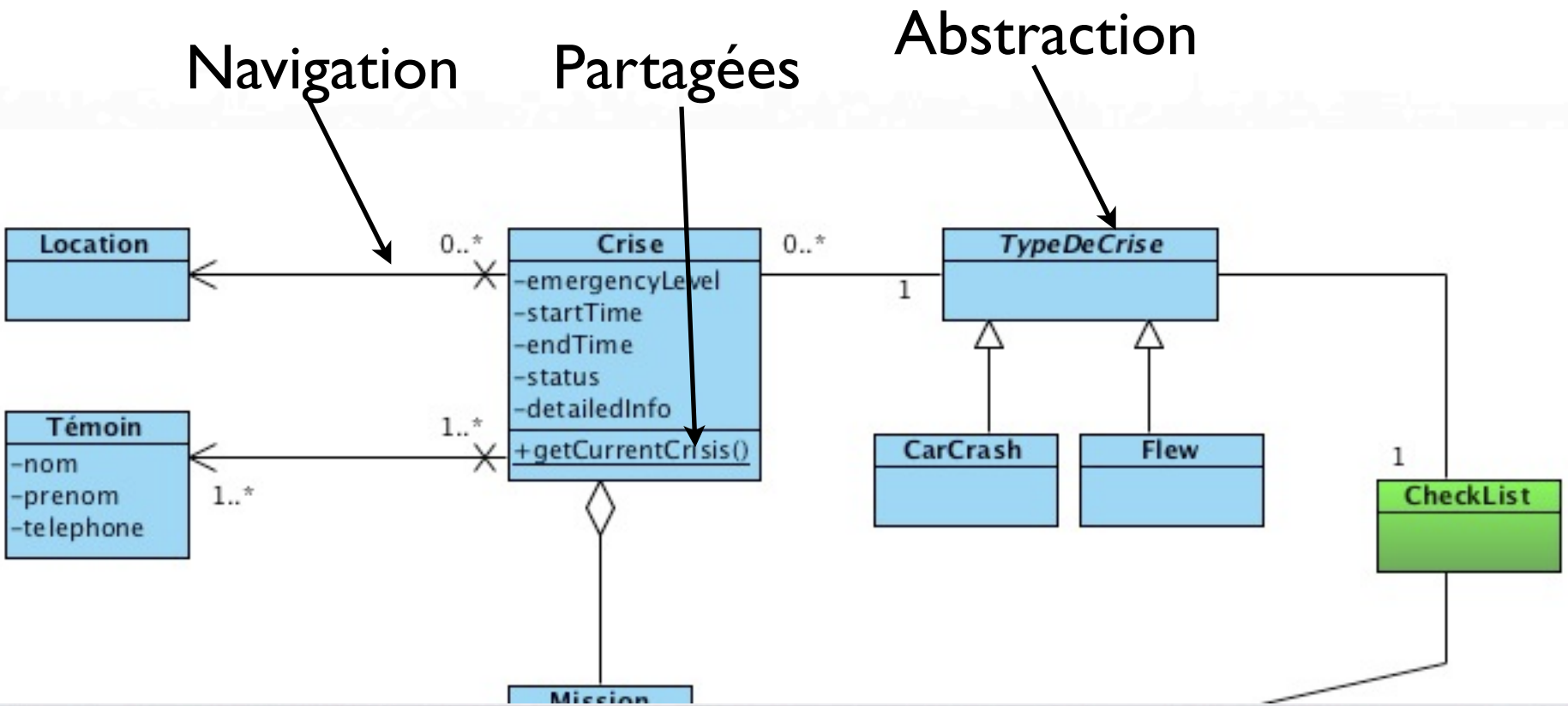
```
class Admin_Controller {
   public function index($args) {
                        $args['infosListe'] = Information::findAll();
                        $view = new Index_Admin_View($args);
                        $view->display();
                }

   public function confirmer_modifier($args) {
                        $key = $_POST["key"];
                        $newTitre = $_POST["NouveauTitre"];
                        $info = Information::read($key);
                        $info->setTitre($newTitre);
                        $info->update();

                        $args['infosListe'] = Information::findAll();

                        $view = new Index_Admin_View($args);
                        $view->display();
                }

   public function create($args) {
                        $titre = $_POST["Titre"];
                        $info = new Information($titre, $this->today());
                        $info->create();

                        $args['infosListe'] = Information::findAll();

                        $view = new Index_Admin_View($args);
                        $view->display();
                }
```


Conception
Focus
-> Architecture
-> Classes
-> Données

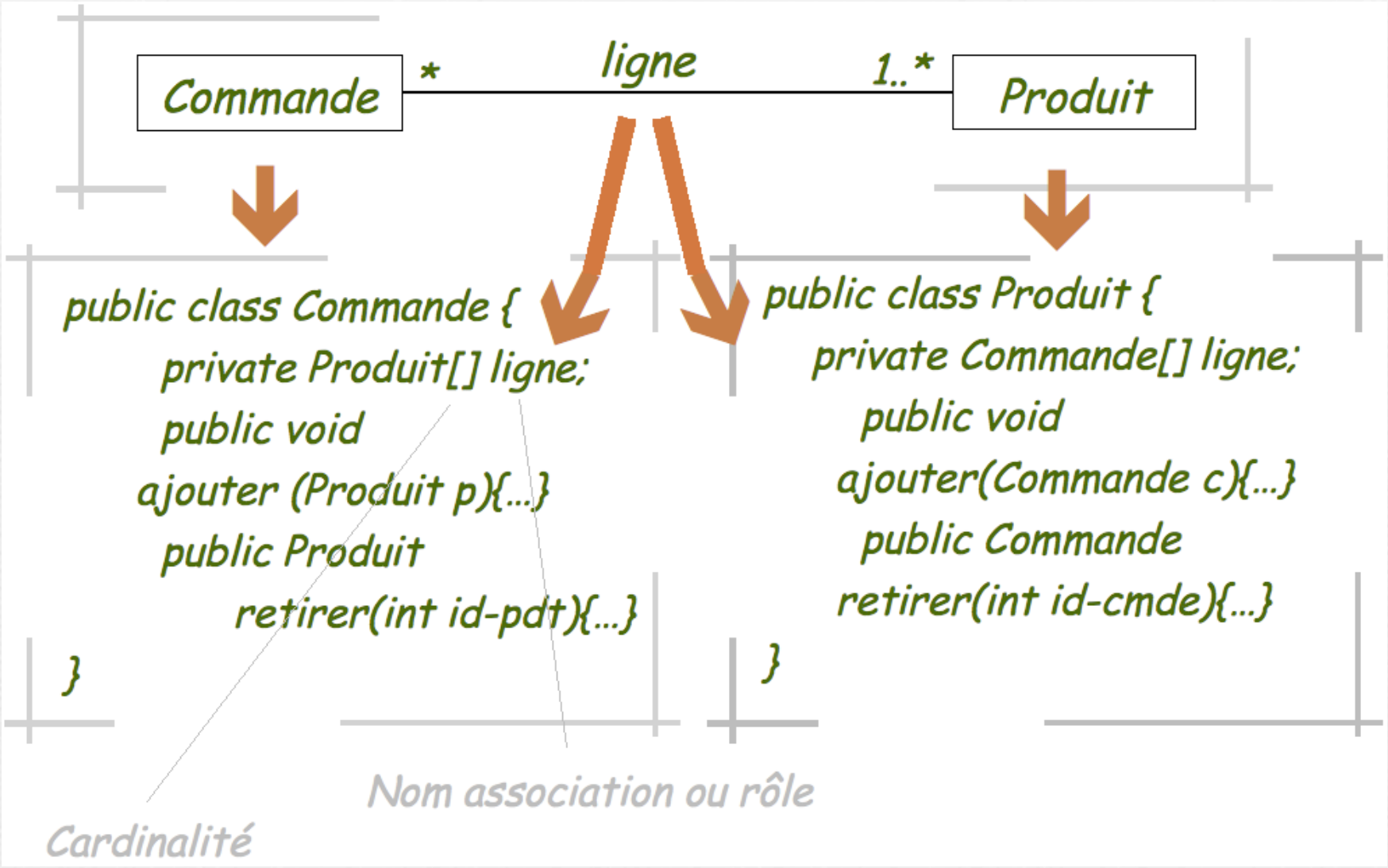
En cours d'analyse, Diagramme de Classes



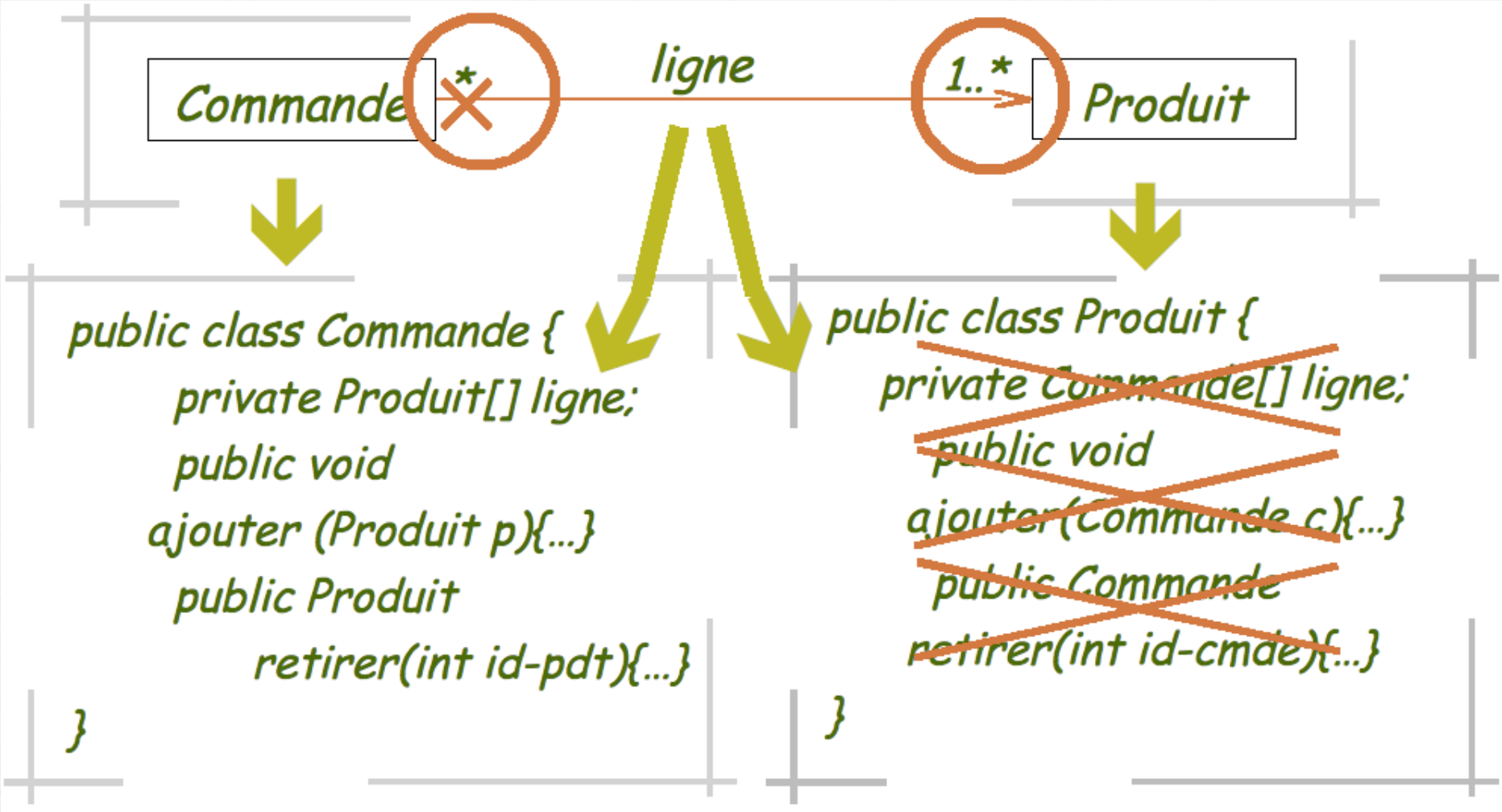
En conception, Diagramme de Classes



Association...



Association...



Association: De la conception à l'implémentation


```
public class Commande {
        private Produit[] ProduitsCommandés;
        public void
      ajouter (Produit p){…}
        public Produit
            retirer(int id-pdt){…}
}
```

Association: De la conception à l'implémentation


```
public class Commande {
        private Produit[] ProduitsCommandés;
        public void
        ajouter (Produit p){ }
```

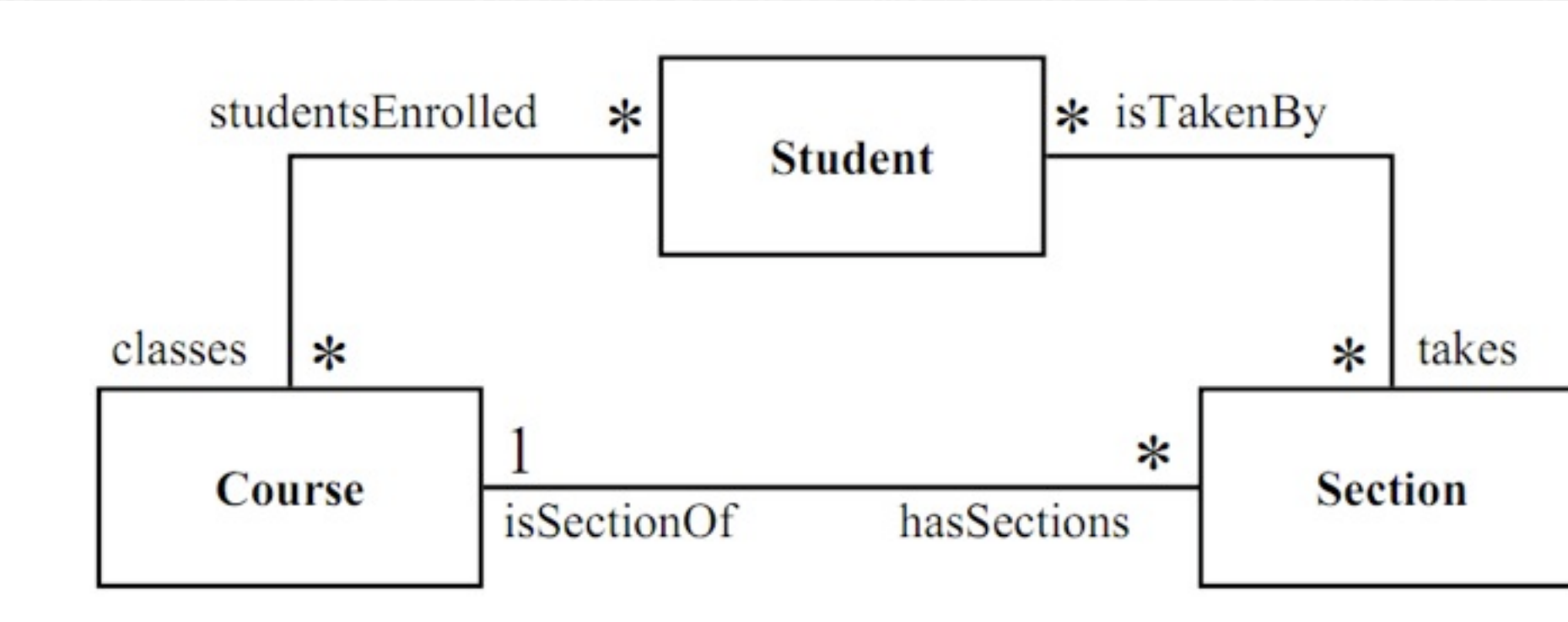
```
public Commande (Produit[] c) throws Exception {
     if (c.length != 0)
          lignes = c;
     else
          throw new Exception("Un produit au moins est requis");
}
```

Association: De la conception à l'implémentation

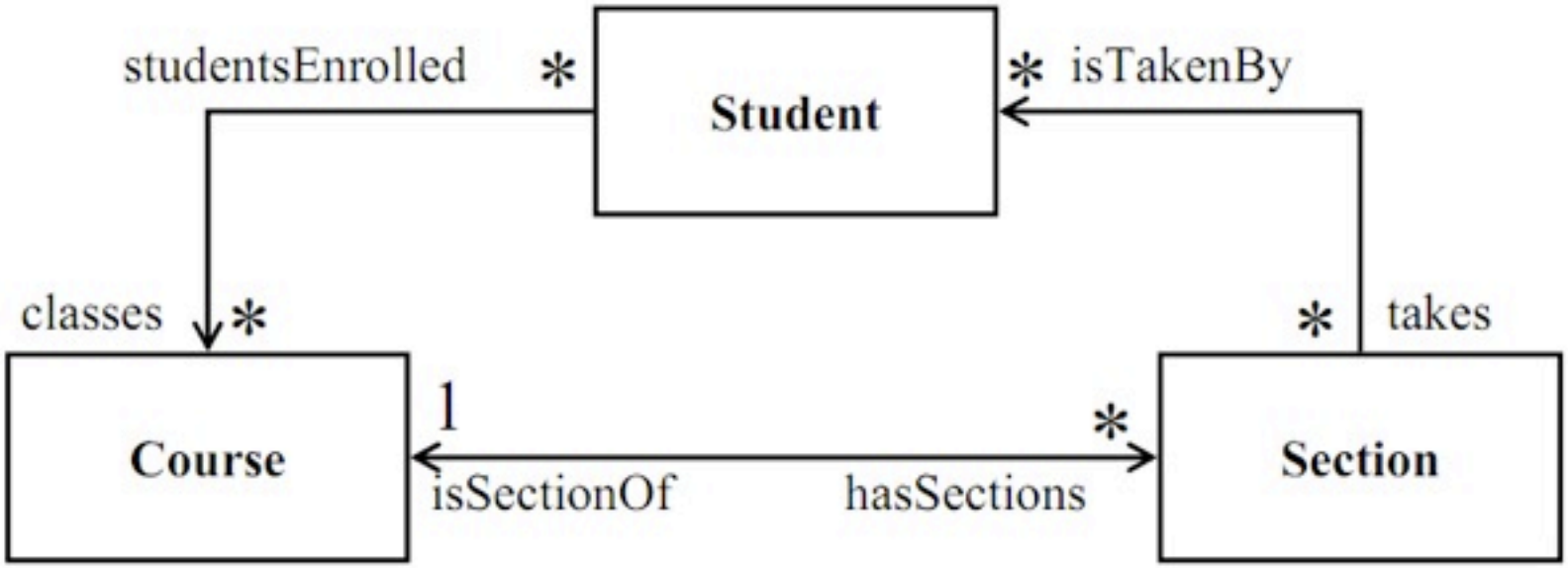

```
    public Commande (Produit[] c) throws Exception {
        if (c.length != 0)
            lignes = c;
        else
            throw new Exception("Un produit au moins est
requis");
    }
```

```
    public boolean oterProduit(Course c) {
        if (lignes.length==1)
            return false;
        ...
    }
```

Gestion des associations



Exemple de Raffinement



Ce n'est qu'un exemple, d'autres raffinements sont possibles...

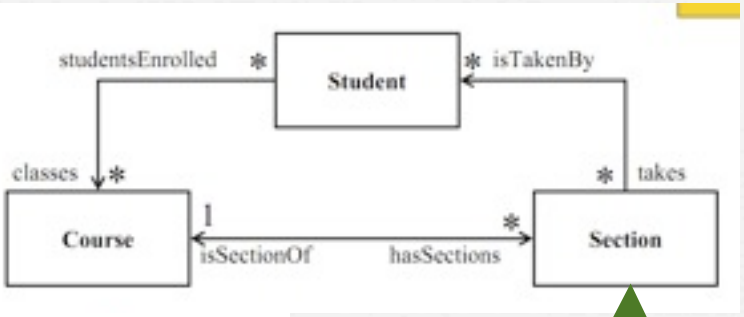
Principes d'implémentation

- Extrémité d'association 1
  - Rôle en Attribut avec type de l'extrémité
  - *Type* get*Role*()
- Extrémité d'association *
  - Rôle (pluriel) en collection
  - Type de l'extrémité en élément de collection
  - Collection get*Roles*()
  - // Collection<TypeExtrémité>//

Principes d'implémentation (Suite)

- *Fixer* une extrémité d'association 1
  - `void setRole(Type t)`
- *Fixer* une extrémité d'association *
  - `void setRoles(Collection c)`
  - `void addRole(TypeElement t)`
- *Fixer* une association navigable dans les 2 sens :
  - Définir les responsabilités : un des objets est responsable de la connexion/déconnexion (cf. exemple)

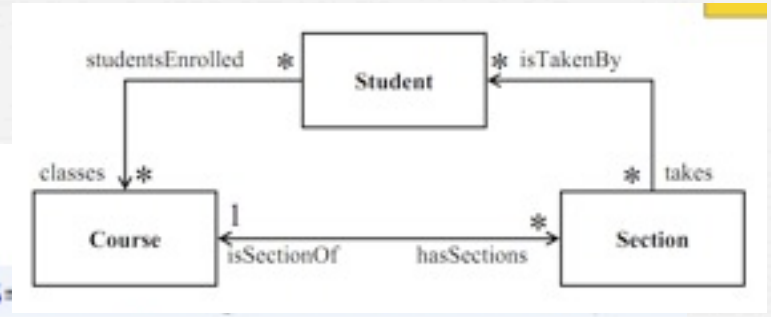
Implémentation




```
public class Section {
    private String name;
    private Course isSectionOf;
    private  Collection<Student> isTakenBy = new ArrayL
    public String toString() {
    public Section(String name, Course isSectionOf) {
        this.name = name;
        //ATTENTION ...
        setIsSectionOf(isSectionOf) ;
    }

    public void setIsSectionOf(Course c) {
        isSectionOf = c;
        c.addHasSections(this);
    }

    public Course getIsSectionOf() { return isSectionOf;}

    public Collection<Student> getIsTakenBy() {
        return isTakenBy;
    }

    public void addStudent(Student s) {
        isTakenBy.add(s);
        if (!(s.getClasses().contains(isSectionOf)) )
            s.addClass(isSectionOf);
    }
```

Prise de responsabilités

Implémentation



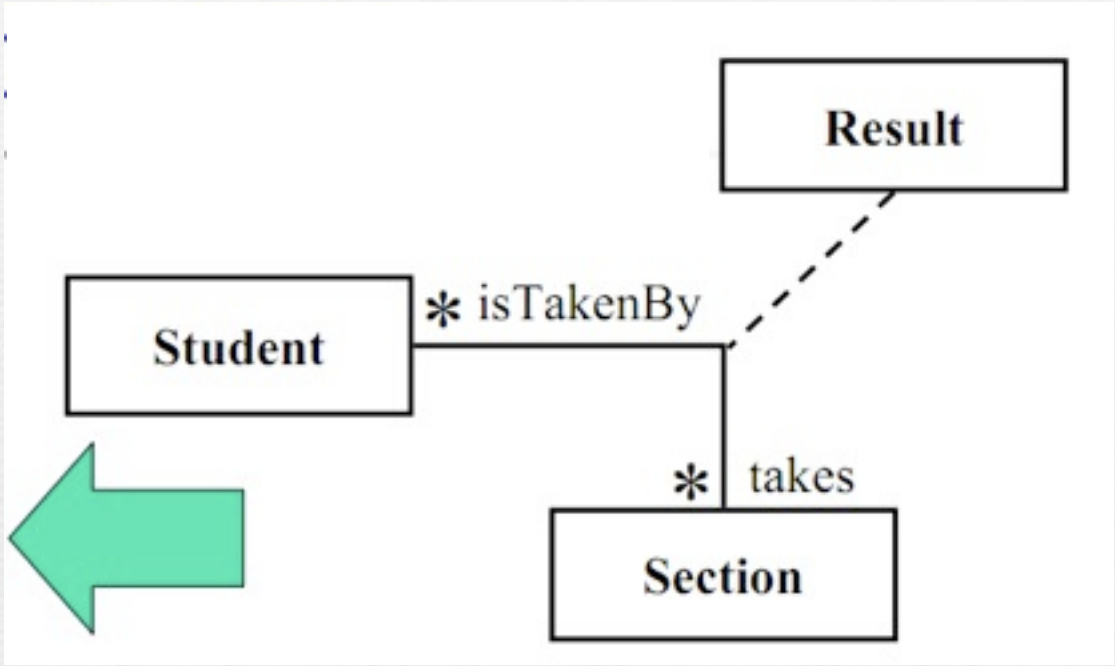

```
public class Student {
        private String name;
        private Collection<Course> classes;

        public Student(String name) {..
        public String toString() {..
        public Collection<Course> getClasses() {
                return classes;
        }
        protected void addClass(Course c){
            classes.add(c);
        }
```

```
public class Course {
    private String name;
    private Collection<Section>  hasSections = new ArrayList<Section>();

    public Course(String name) {..
    public String toString() {..
    public Collection<Section> getHasSections() {
        return hasSections;}
    protected void addHasSections(Section s){
            hasSections.add(s);
        }
```

Définition des responsabilités
Ne jamais appeler ***addHasSections*** ou ***addClass*** directement !

Implémentation

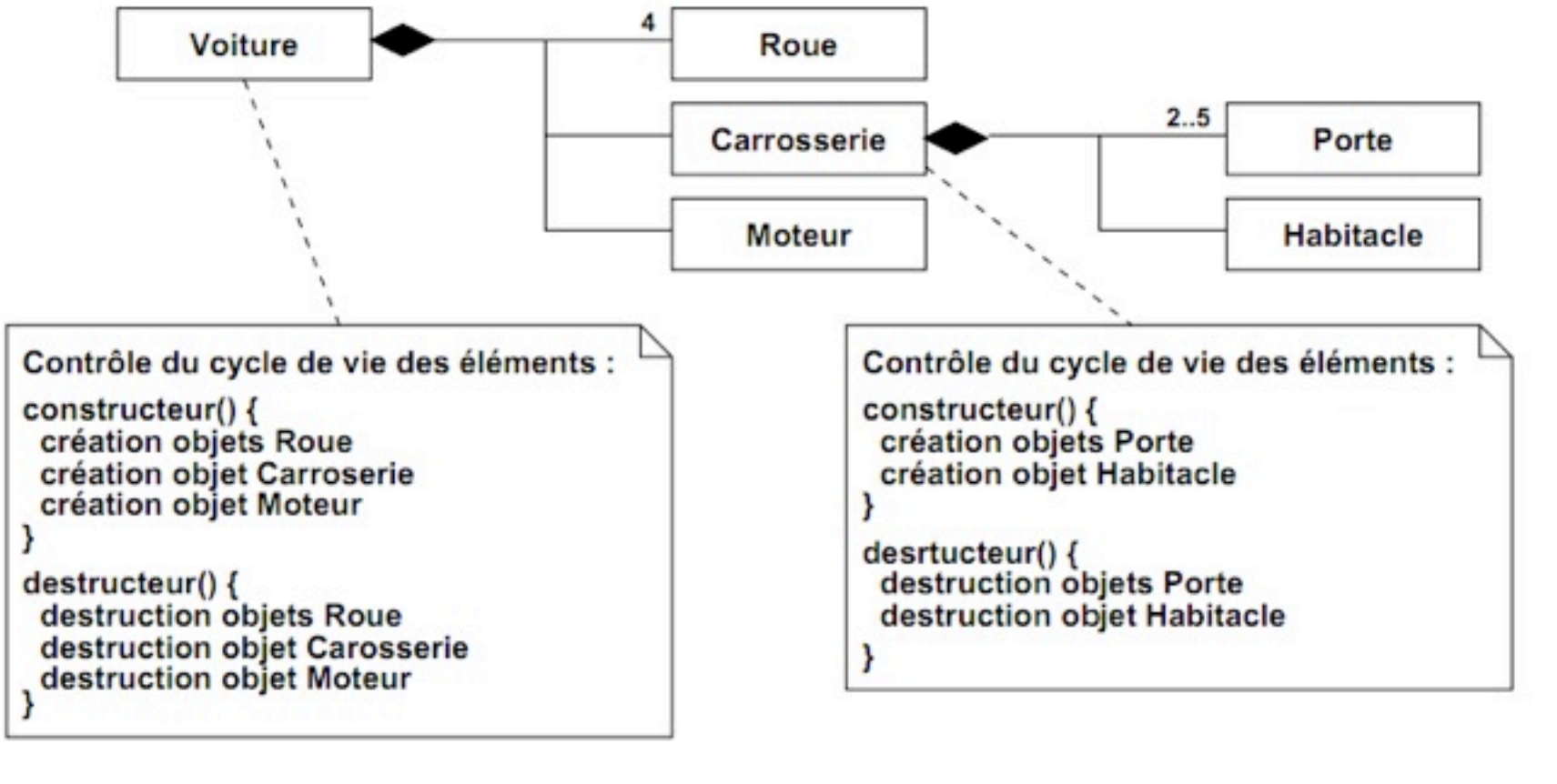
```
class Student {
   Result getResult(Section s)
  }
class Section {
  Result getResult(Student s)
 }
class Result {
  Student getStudent()
  Section getSection()
}
```

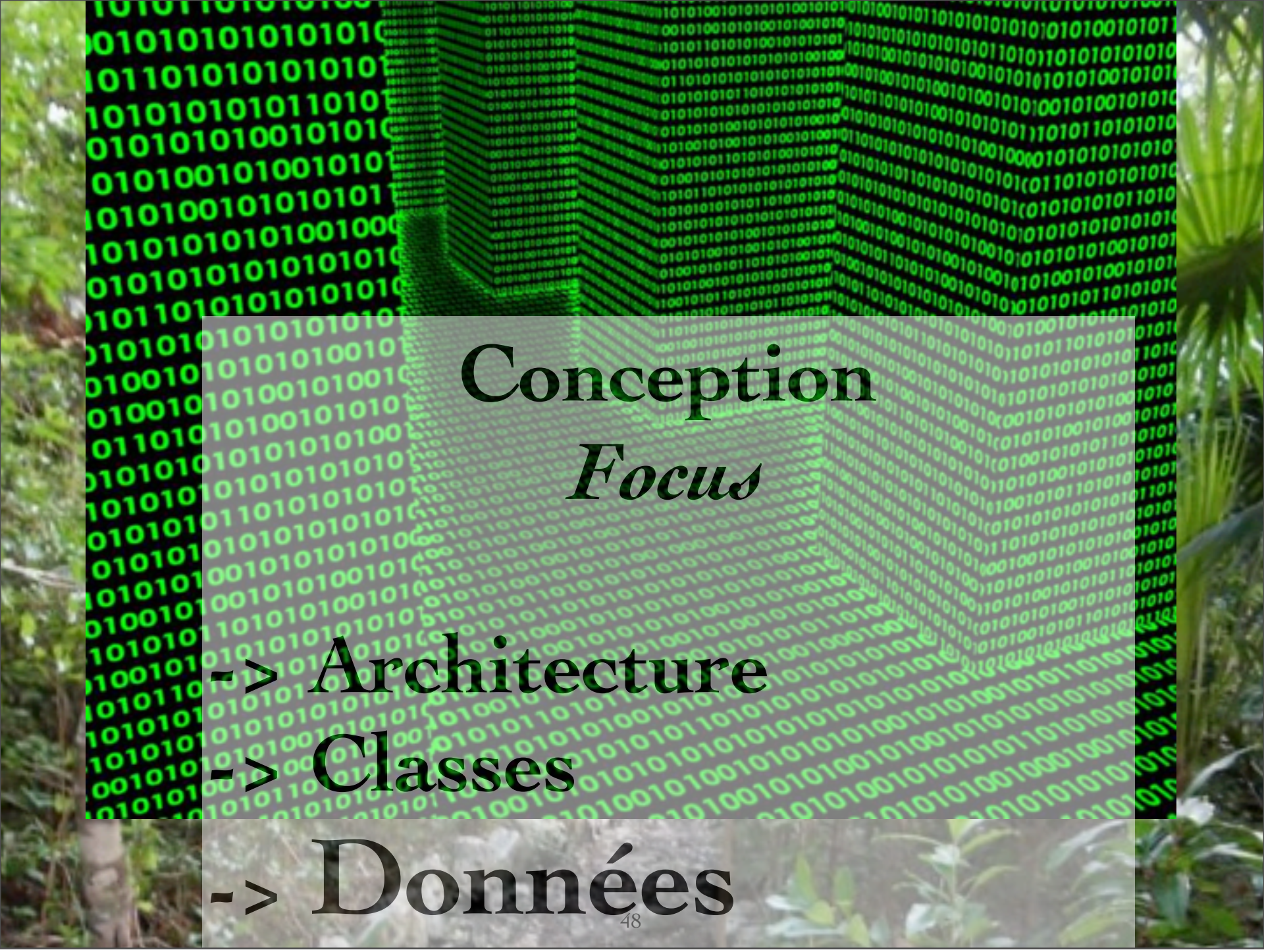


En résumé : Traduction des associations en attributs

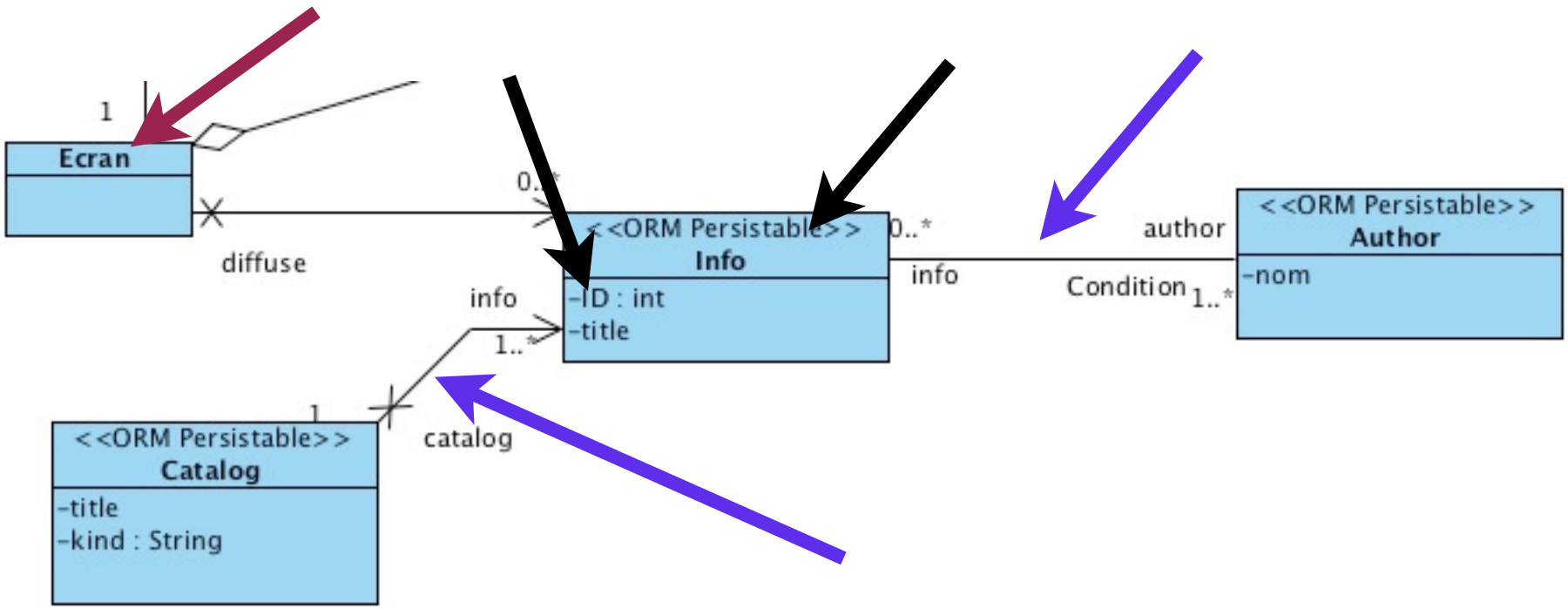
- Autant d'attributs que de classes auxquelles elle est reliée (navigable)
- Association unidirectionnelle = pas d'attribut du côté de la flèche
- Nom de l'attribut = nom du rôle ou forme nominale du nom de l'association
- Attribut du type référence sur un objet de la classe à l'autre extrémité de l'association
  - Référence notée « @ »
- Traduction des multiplicités
  - 1 =⇒ @Classe
  - ∗ =⇒ Collection @Classe
  - 0..N =⇒ Tableau[N] Classe
- Multiplicité avec tri = Collection ordonnée @Classe

Compositions

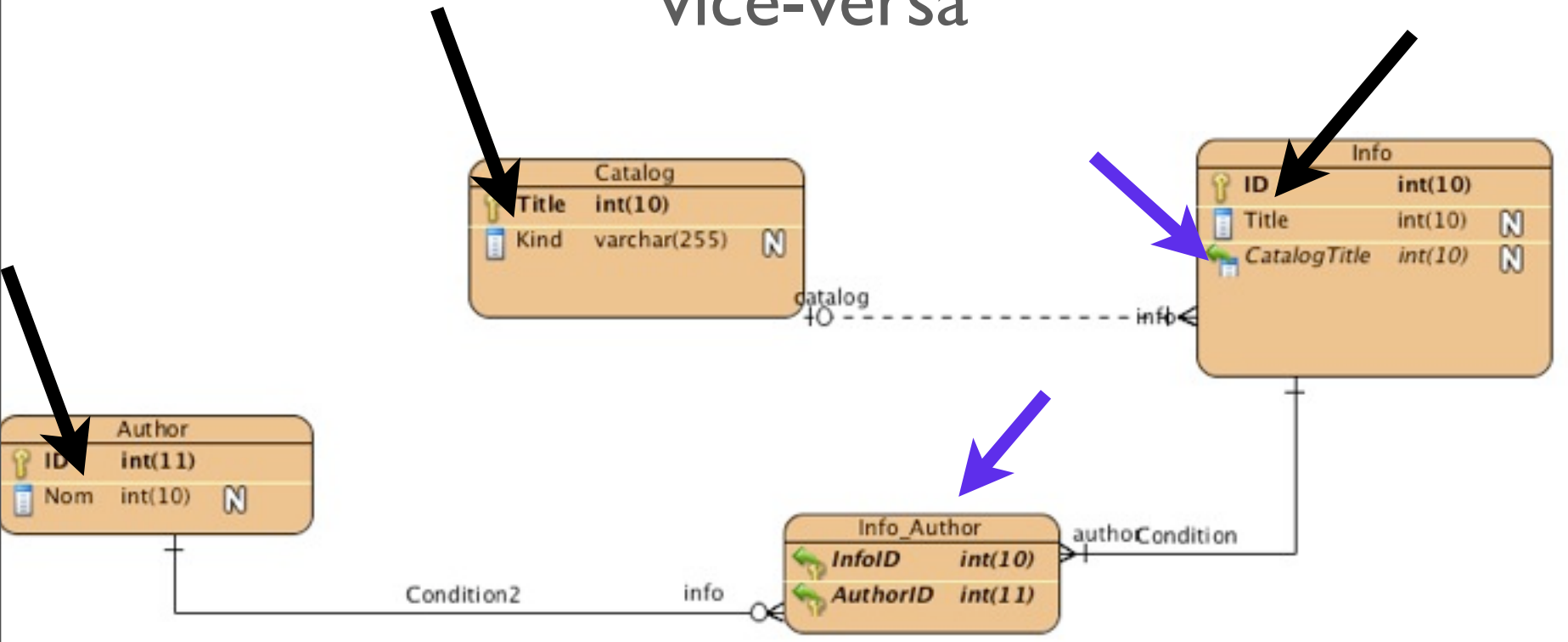


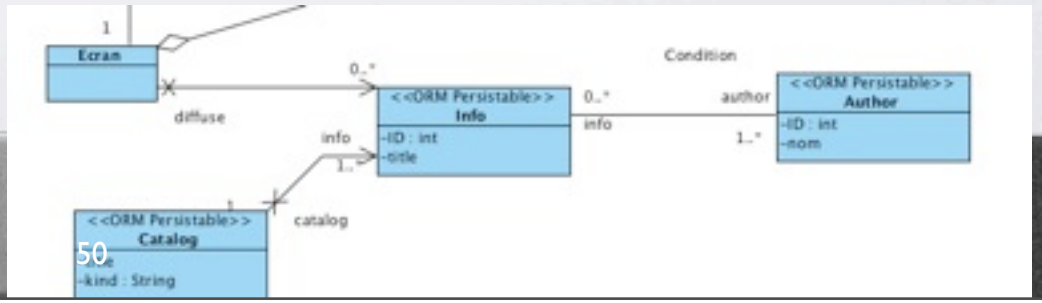

Conception
Focus
-> Architecture
-> Classes
-> Données

Des classes aux données

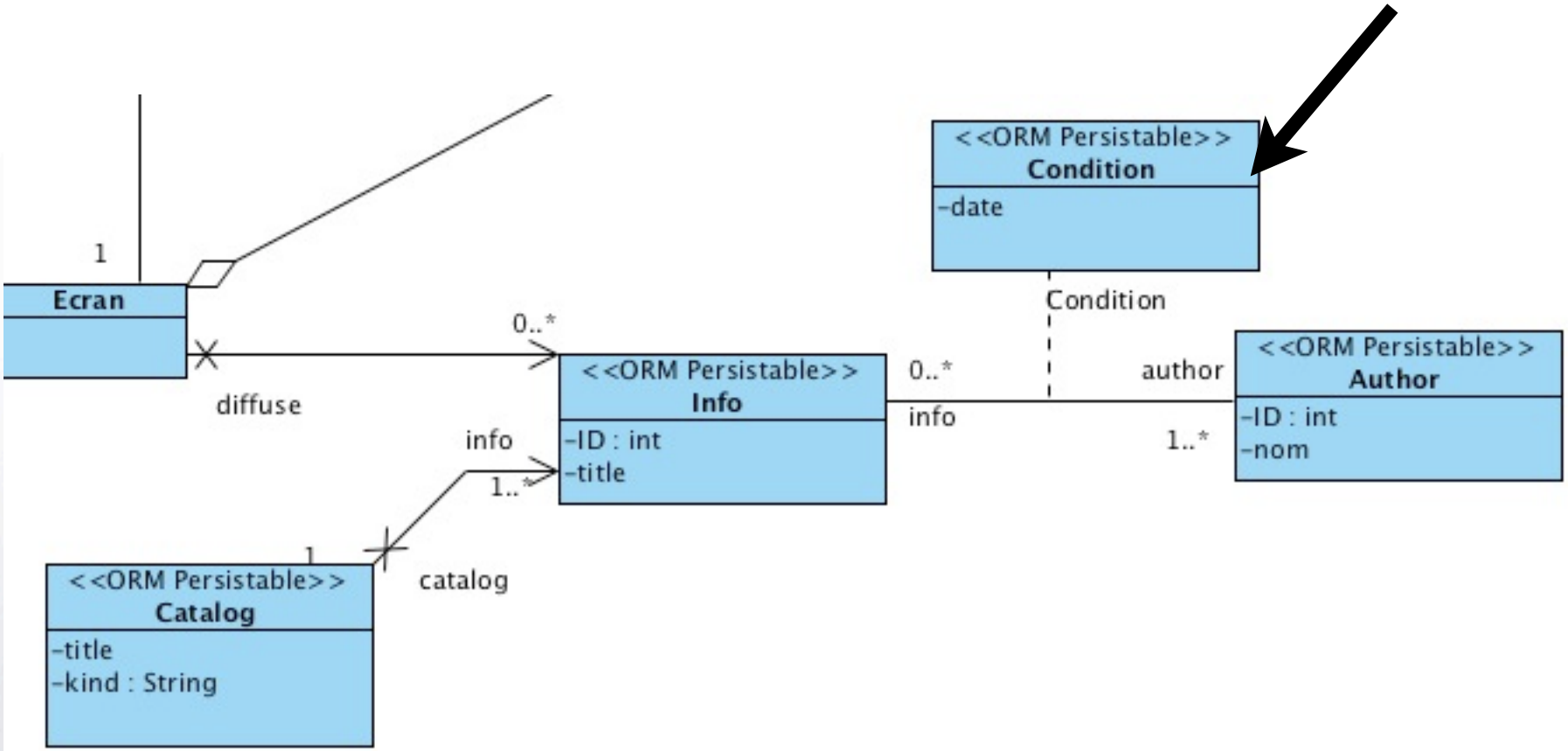


Des classes aux données (entité-relation) et vice-versa

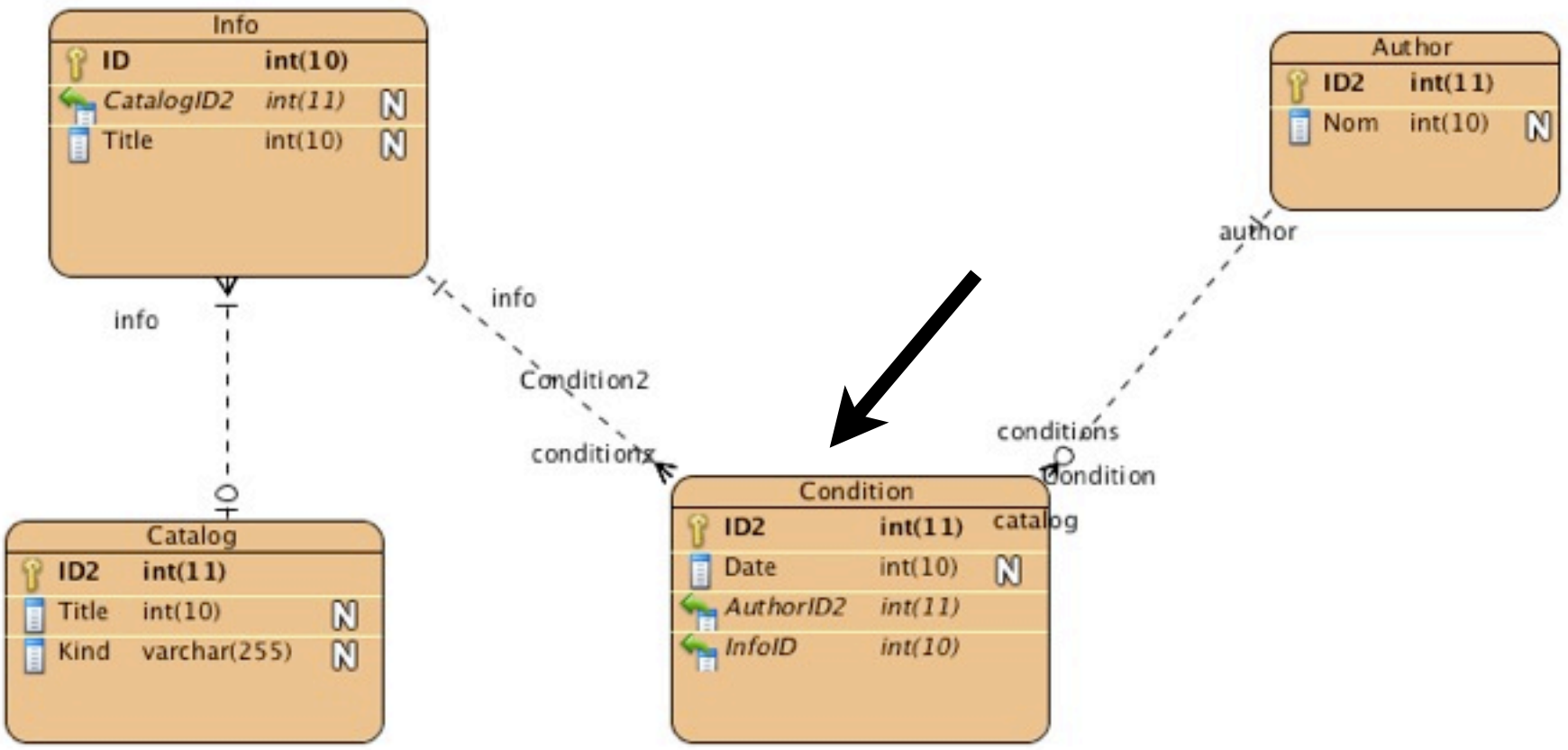




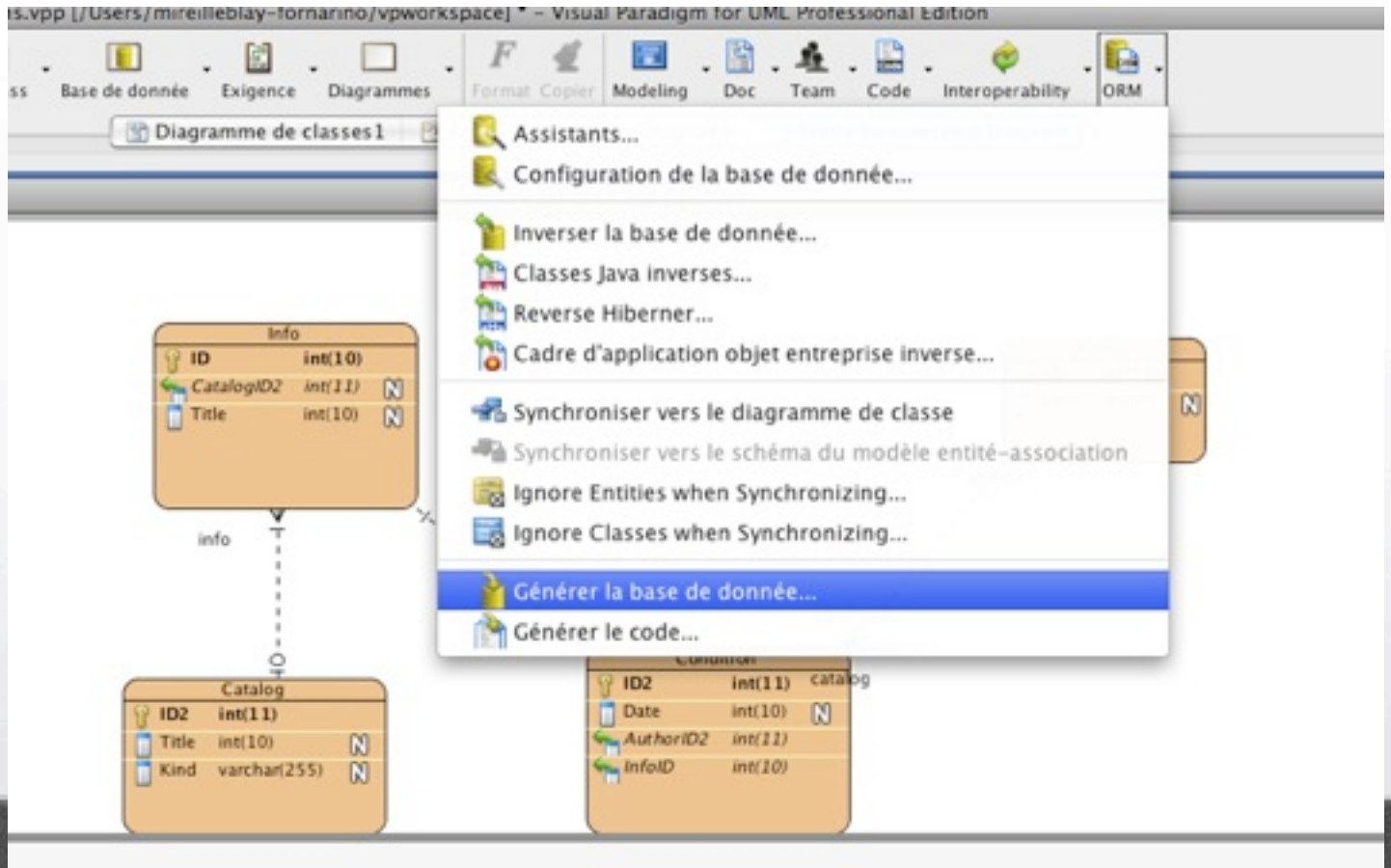
Des classes aux données



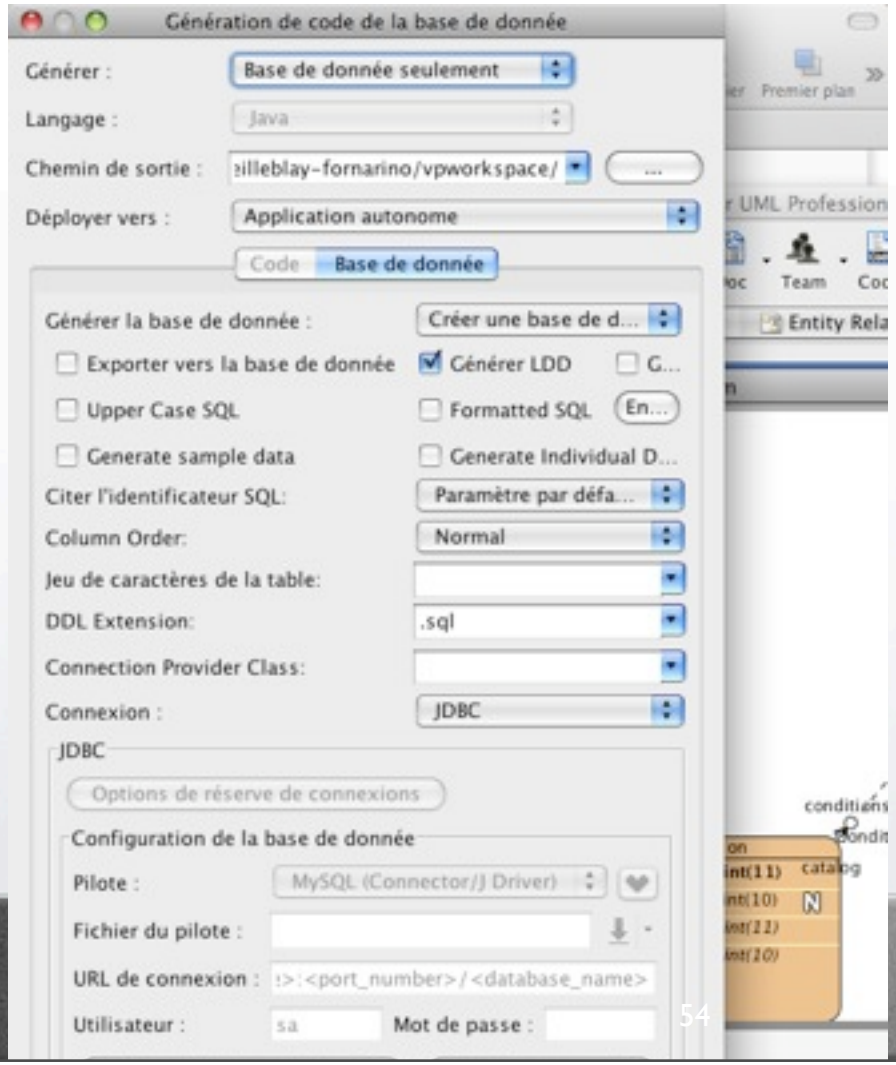
Des classes aux données



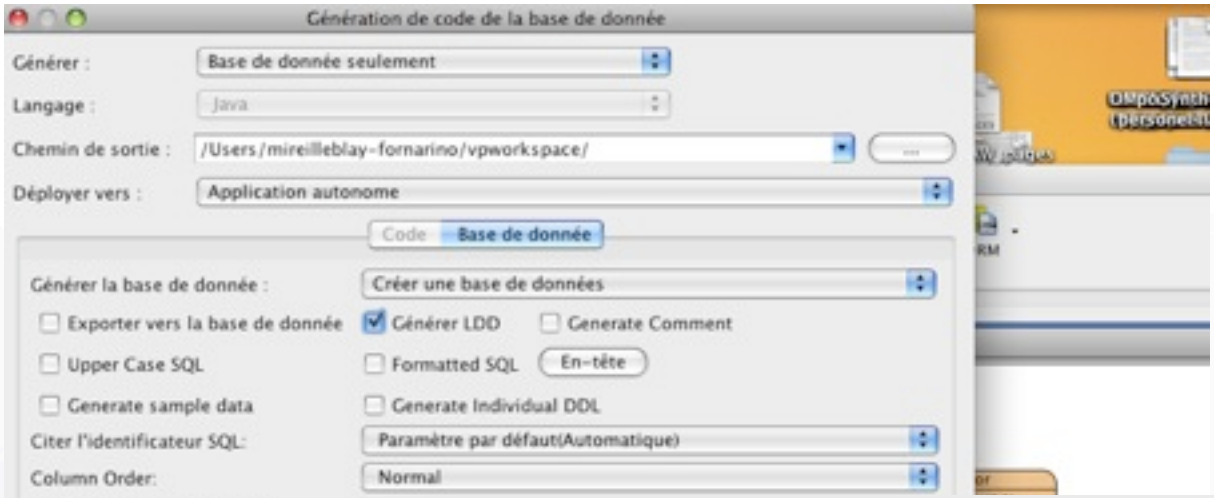
Des classes aux données



Des classes aux données



Des classes aux données




```
create table Info (ID int(10) not null auto_increment, CatalogID2 int(11), Title int(10), primary key (ID));
create table Author (ID2 int(11) not null auto_increment, Nom int(10), primary key (ID2));
create table Catalog (ID2 int(11) not null auto_increment, Title int(10), Kind varchar(255), primary key (ID2));
create table `Condition` (ID2 int(11) not null auto_increment, `Date` int(10), AuthorID2 int(11) not null, InfoID int(10) not null, primary key (ID2));
alter table `Condition` add index `Condition` (AuthorID2), add constraint `Condition` foreign key (AuthorID2) references Author (ID2);
alter table `Condition` add index Condition2 (InfoID), add constraint Condition2 foreign key (InfoID) references Info (ID);
alter table Info add index FKInfo468757 (CatalogID2), add constraint FKInfo468757 foreign key (CatalogID2) references Catalog (ID2);
```


ApprocheS :
Méthodologies au
prochain cours