



Chapitre 9

Architecture des applications

Structurer les applications en couches en vue de faciliter leur intégration dans la structure de l'entreprise, leur réutilisation et leur mise à jour

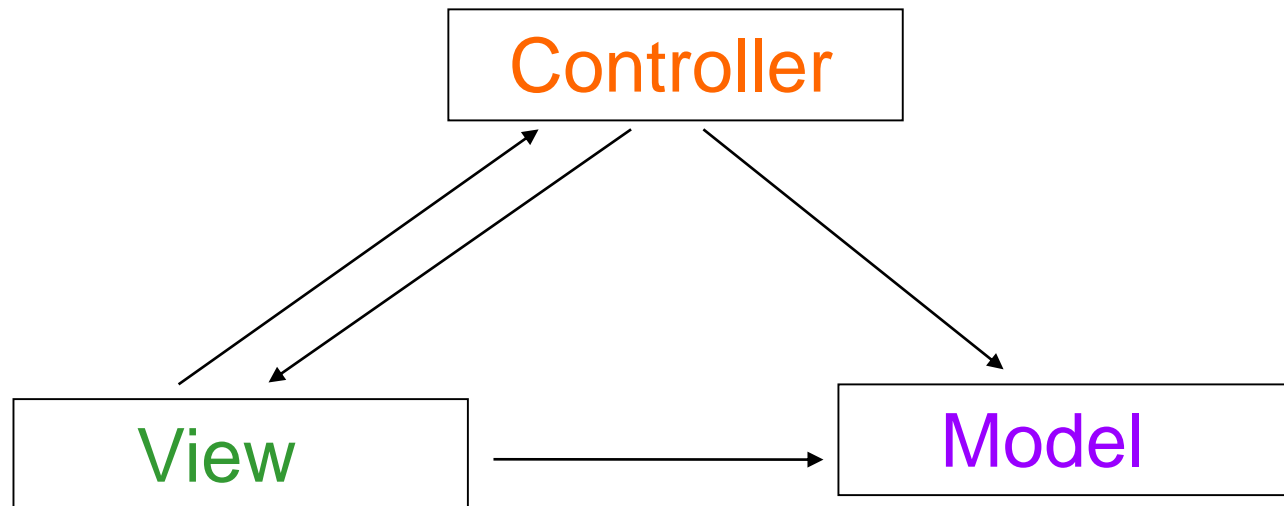
Architecture des applications

1. Model – View – Controller (MVC)

Model - View - Controller

MVC Pattern

Découplage de l'accès aux données
et de la présentation des données



Présentation des données
Interactions avec l'utilisateur

Accès aux données

Model - View - Controller

Model

- Gère les informations
 - Les données de l'application
- Consiste en des **données** de l'application et des **règles métier**
- Fournit le stockage permanent des données
- Notifie le observateurs quand l'information change

Model - View - Controller

View

- Reçoit les entrées de l'utilisateur
- Affiche les données
 - La vue traduit les données sous une forme présentable pour l'utilisateur
 - Demande à la couche Model les informations nécessaires pour générer une représentation à afficher à l'utilisateur
 - C'est une représentation du model
 - ⇒ Il peut y avoir plusieurs vues associées à un même model !

Model - View - Controller

Controller

- Transfère l'information entre la vue et le modèle
- Gère toutes les requêtes venant de la vue (user interface)
 - Forwarde la requête au gestionnaire approprié
- Est responsable de l'accès au modèle et de son rendu à l'utilisateur
- Peut envoyer des commandes
 - A la vue : pour changer la représentation du modèle
 - Au modèle : pour mettre à jour son état

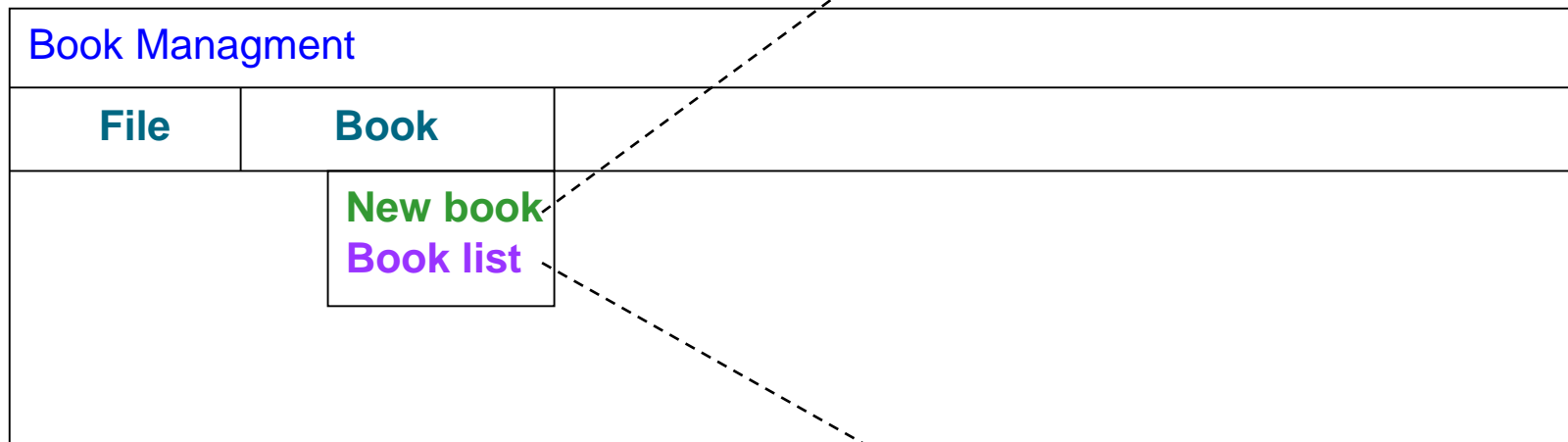
+ Sécurité

Architecture des applications

Ex: Gestion bibliothèque :

- Ajout d'un nouveau livre
- Listing de tous les livres enregistrés

Panneau d'affichage du formulaire
d'encodage d'un nouveau livre :
NewBookPanel



Panneau d'affichage de la liste de tous
les livres de la BD (dans une JTable) :
AllBooksPanel

Model -View - Controller

View

MainJFrame

NewBookPanel

AllBooksPanel

Controller

ApplicationController

Model

Architecture des applications

- 1. Model – View – Controller**
- 2. Modèle 3 couches (3-tiers Model)**

Surtout au niveau **physique** :
si l'application est distribuée sur des
machines différentes



Couche présentation de
l'application
*Ex: pages Web exécutées
chez le client*

Couche métier: traitement
de l'information
*Ex: sur serveur dans
l'entreprise*

**Couche accès et
stockage des données**
*Ex: serveur de BD à
distance*

Couche business

La **couche métier** est très développée en entreprise.

Elle comprend toute **l'intelligence**, le noyau de l'entreprise

⇒ Boîte à outils de méthodes contenant l'intelligence métier

⇒ **Librairie de fonctions métier** réutilisables :

Intelligence artificielle, algorithmes des tâches métier, calculs complexes, statistiques ...

Ex : calcul de fiches de paie, de factures avec réductions éventuelles, planification et optimisation des trajets de livreurs, organisation de rencontres sportives, d'horaires ...

User Interface

MainJFrame

NewBookPanel

AllBooksPanel

Business Logic

BookManager

(Couche peu développée dans cet exemple: pourrait contenir des calculs élaborés (statistiques, intelligence artificielle...))

Book

Data Access

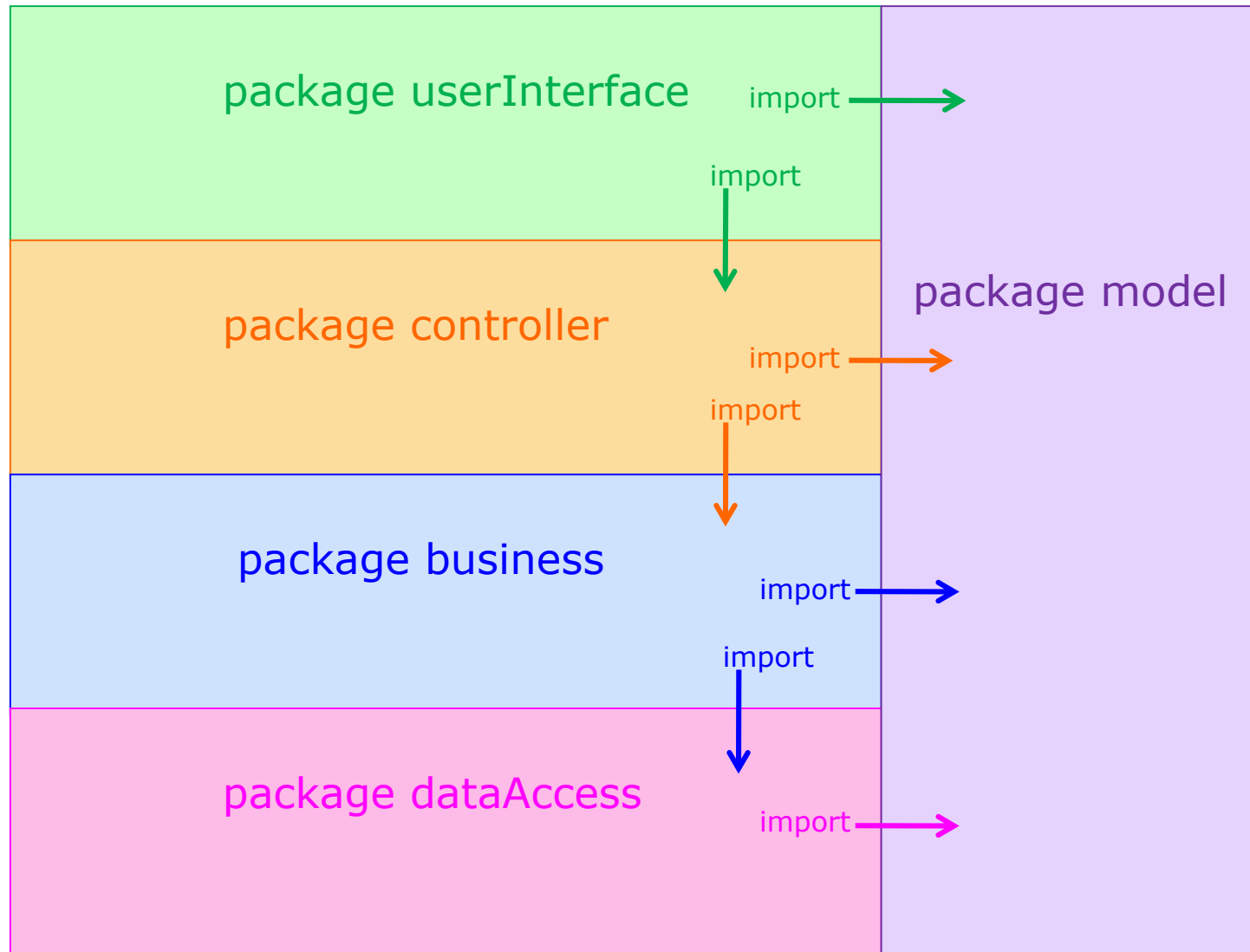
BookDBAccess

Conversion java ↔ SQL

Architecture des applications

- 1. Model – View – Controller**
- 2. Modèle 3 couches (3-tiers Model)**
- 3. Découpe en couches**

Structure des packages



Découpe en couches

User Interface

MainJFrame

NewBookPanel

AllBooksModel

AllBooksPanel

Controller

ApplicationController

Business Logic

BookManager

(Couche peu développée dans cet exemple)

Data Access

BookDBAccess

Conversion java ↔ SQL

Model

Book

Découpe en couches

Si choix de l'option de menu **New book** :

① Dans gestion event de l'option de menu New book :
new **newBookPanel()**

View

MainJFrame

NewBookPanel

② Dans gestion event du bouton insérer :
créer un objet Book book
book = new Book(...)

Book

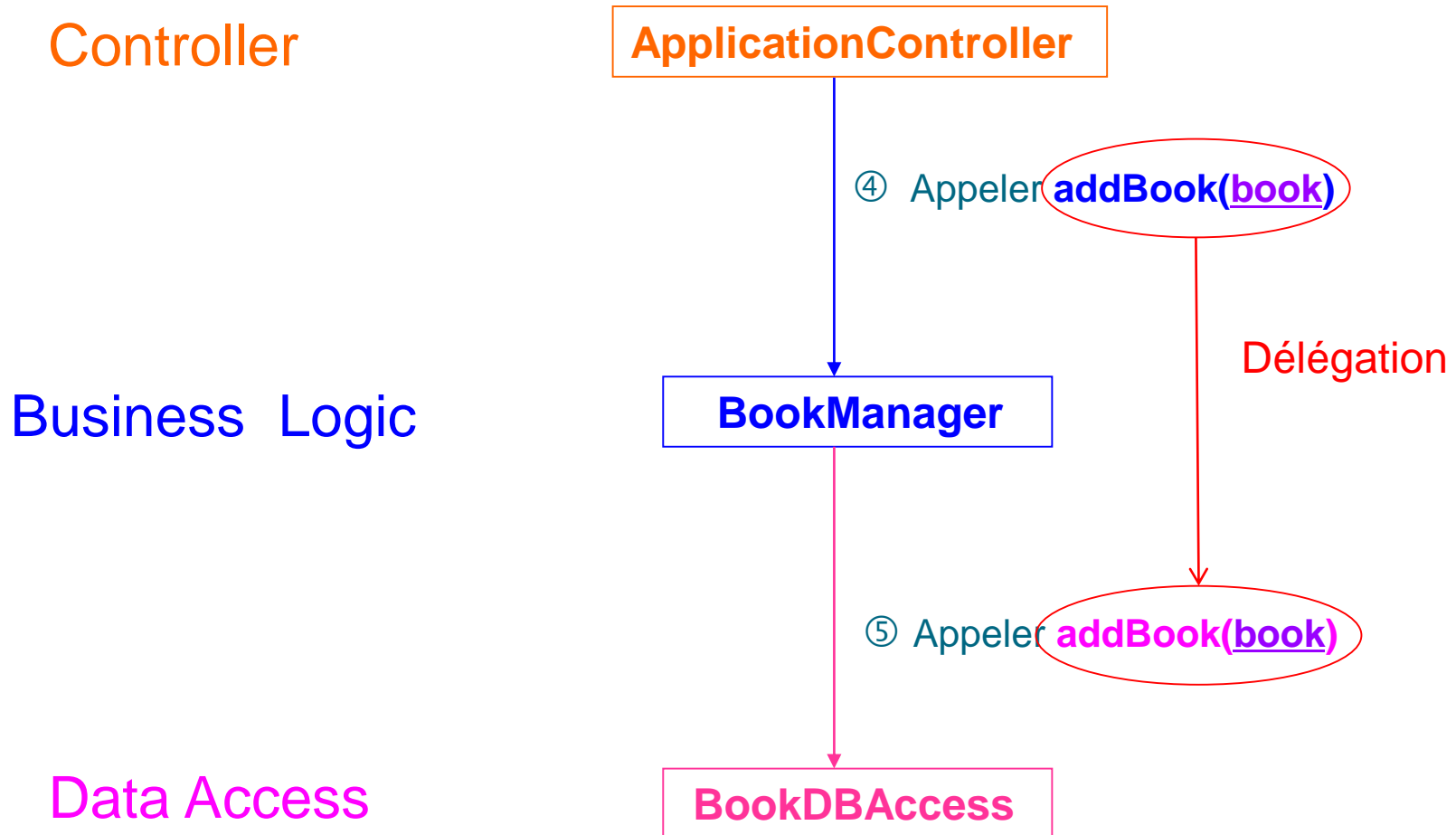
③ Appeler **addBook** (**book**)

Controller

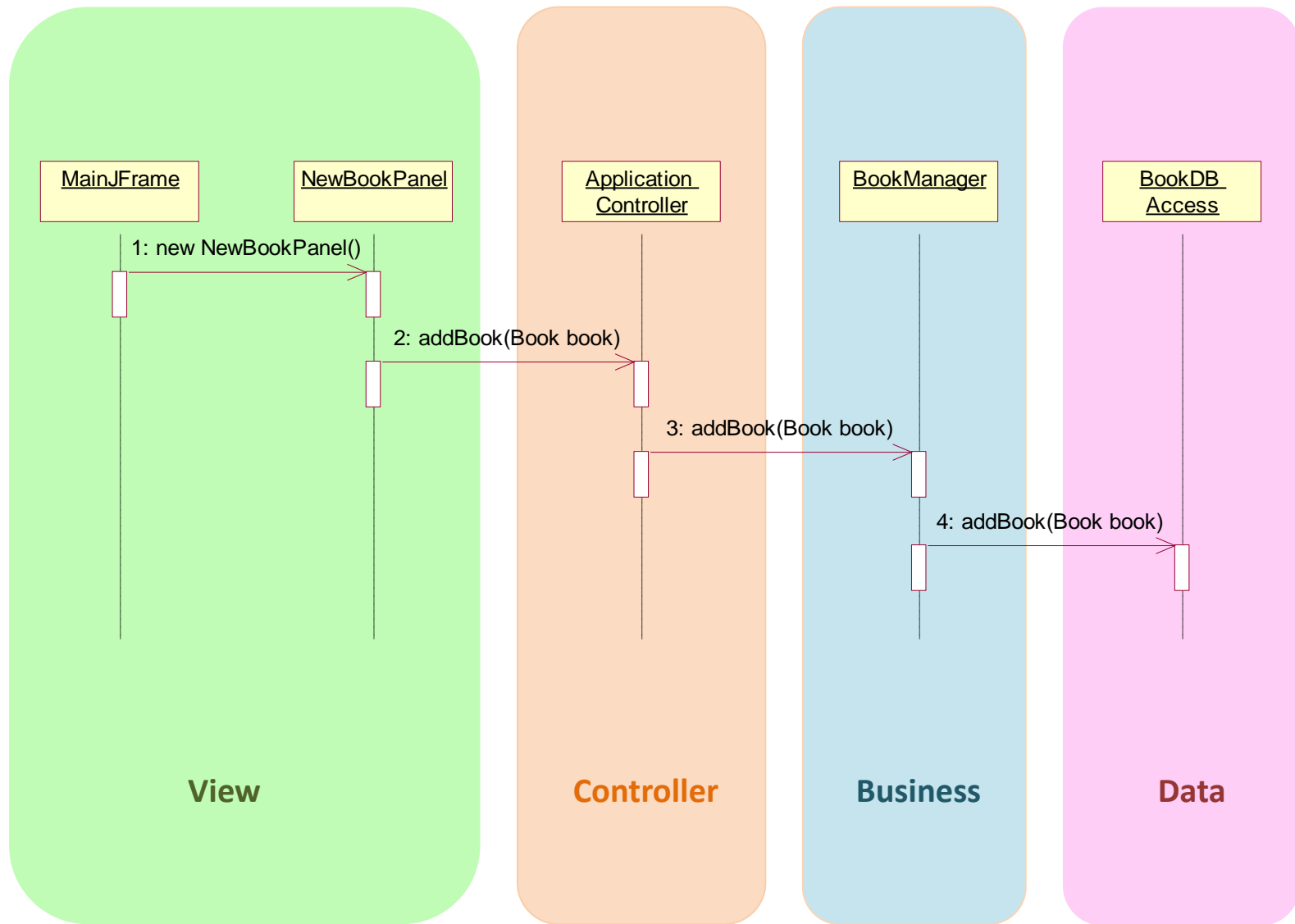
ApplicationController

Découpe en couches

Si choix de l'option de menu **New book** (suite) :

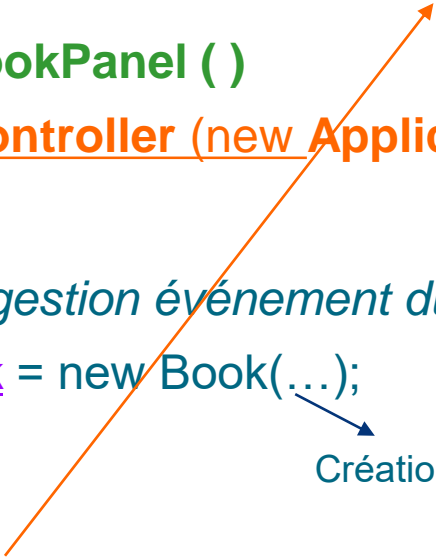


Découpe en couches



Découpe en couches

```
public class NewBookPanel extends JPanel {  
    private ApplicationController controller ;  
    public NewBookPanel ( )  
        { setController (new ApplicationController()) ; }  
    ...  
    // Dans la gestion événement du bouton insérer :  
    Book book = new Book(...);  
    try  
        { controller.addBook (book); }  
    catch ( AddBookException exception) —————→ Cf point 6  
        { ... }
```



Création du livre à partir des données introduites par l'utilisateur

Découpe en couches

```
public class ApplicationController {  
  
    private BookManager manager ;  
  
    public ApplicationController ()  
        { setManager (new BookManager()); }  
  
    public void addBook (Book book) throws AddBookException  
    { manager.addBook(book); }  
}
```

Délégation

Découpe en couches

```
public class BookManager {  
    private BookDBAccess dao ;
```

```
    public BookManager ( )  
        { setDao (new BookDBAccess()) ; }
```

```
    public void addBook (Book book) throws AddBookException  
    {  
        ... ← Tests et traitements éventuels sur le livre avant insertion dans la BD  
              (ex: recherche sur internet des infos du livre à partir de l'ISBN)  
  
        dao addBook (book);  
    }  
}
```

Délégation

Découpe en couches

```
public class BookDBAccess {
```

```
    public void addBook (Book book) throws AddBookException  
    {
```

```
        "insert into book values (...)"
```

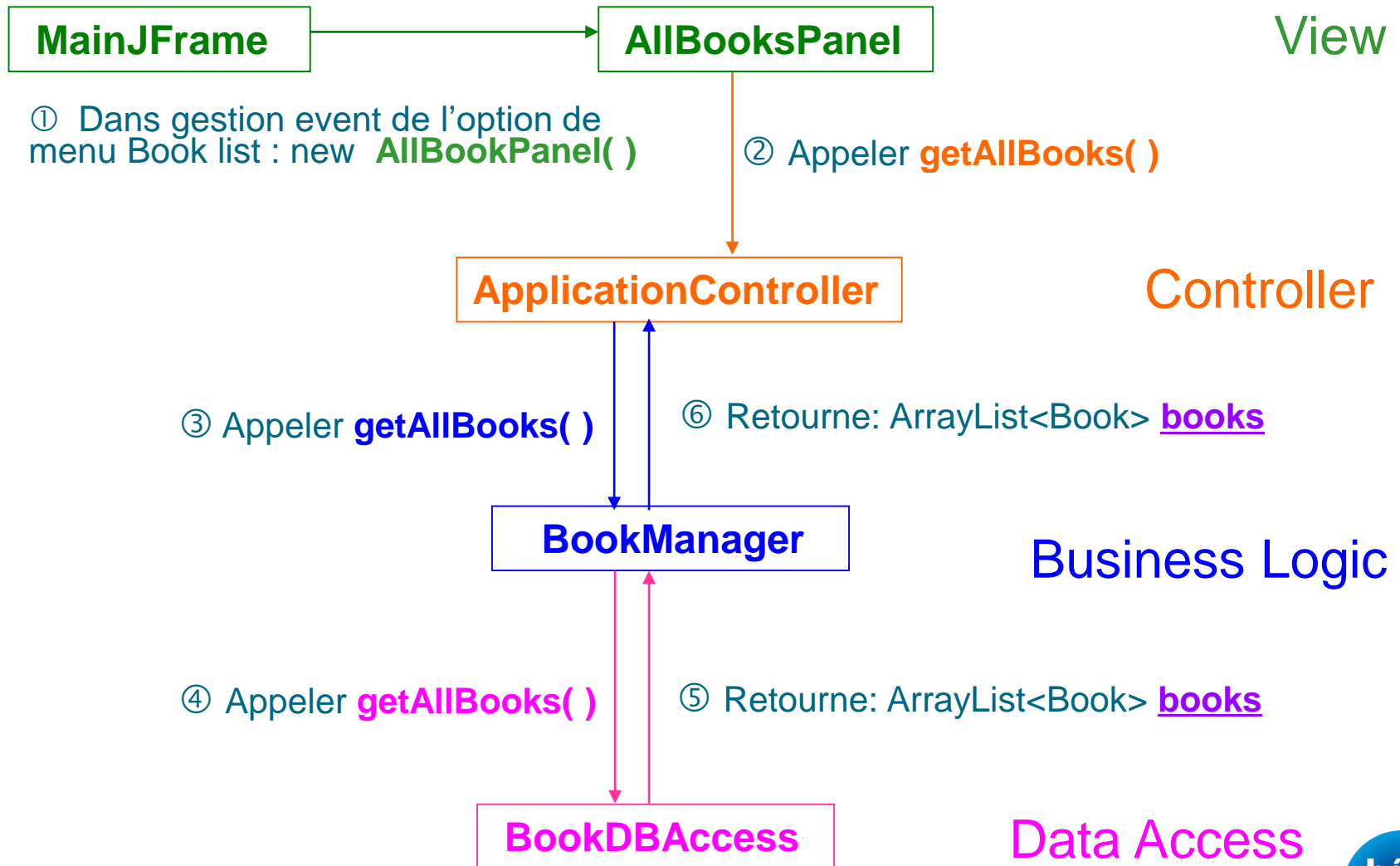
```
        ...
```

→ **Essayer** d'accéder à la base de données et d'exécuter l'instruction SQL

```
    }
```

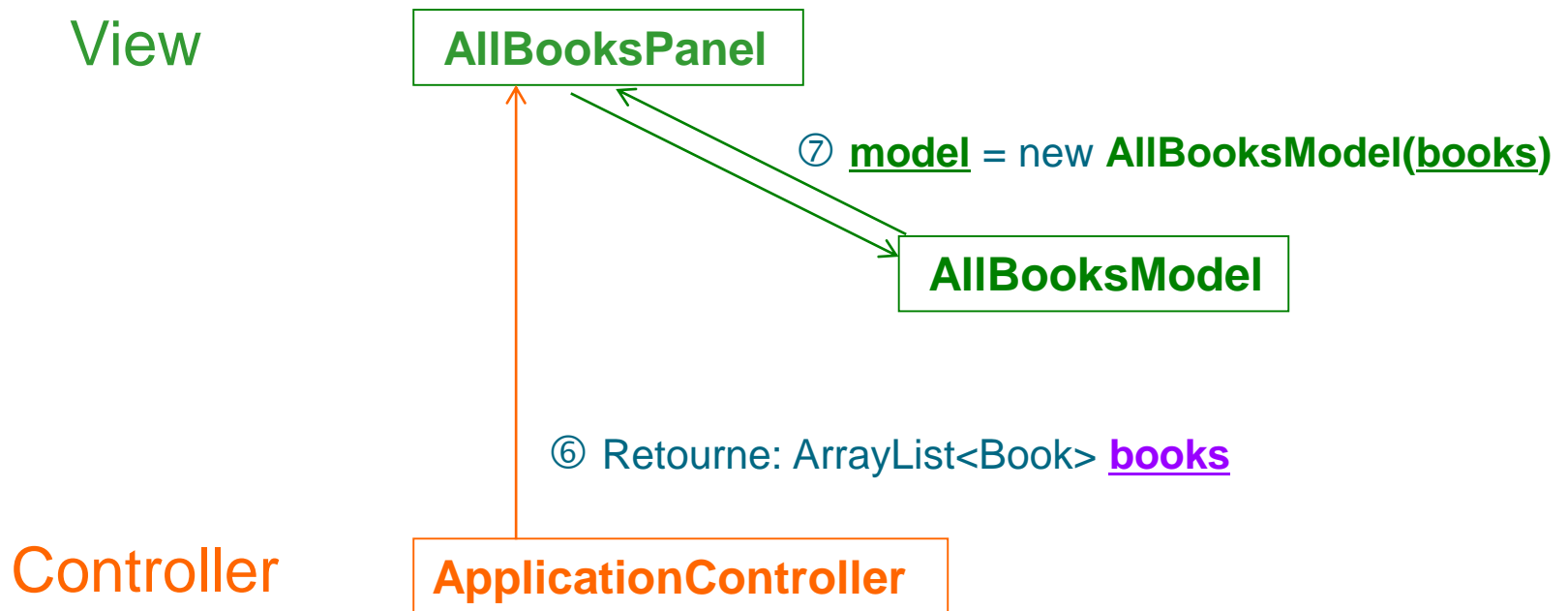
```
}
```

Si choix de l'option de menu **Book list** :

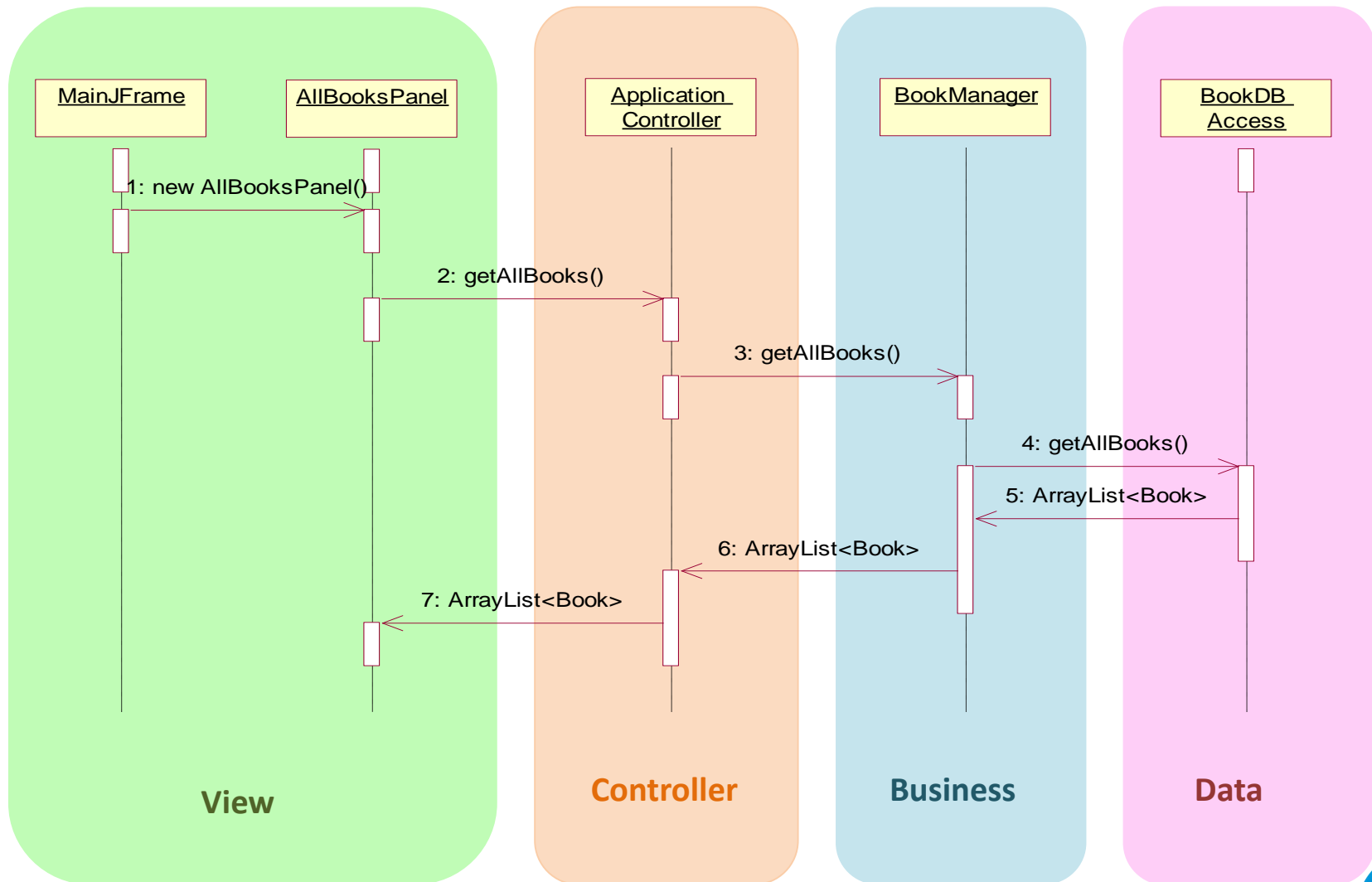


Découpe en couches

Si choix de l'option de menu **Book list (suite)** :



Découpe en couches



Découpe en couches

```
public class AllBooksPanel extends JPanel
{ private ApplicationController controller ;

  public AllBooksPanel ( )
  { setController (new ApplicationController ());
    try
    { ArrayList<Book> books = controller.getAllBooks( );
      ...           —————→ Afficher les livres à l'écran
    }
    catch ( AllBooksException ex)   —————→ Cf point 6
    { ... }
  }
}
```

Découpe en couches

```
public class ApplicationController {  
    private BookManager manager ;  
  
    public ApplicationController ()  
    { setManager(new BookManager()); }  
  
    public ArrayList<Book> getAllBooks( ) throws AllBooksException  
    {  
        return manager getAllBooks( );  
    }  
}
```

Délégation

Découpe en couches

```
public class BookManager {  
    private BookDBAccess dao ;
```

```
    public BookManager ( )  
        { setDao (new BookDBAccess()); }  
}
```

```
    public ArrayList <Book> getAllBooks( ) throws AllBooksException
```

```
    { ArrayList <Book> bookList = dao.getAllBooks( ) ;
```

... \longrightarrow *Traitements éventuels sur la liste de livres*

```
        return bookList ;
```

```
    }
```

```
}
```

Délégation

Découpe en couches

```
public class BookDBAccess {
```

```
    public ArrayList <Book> getAllBooks( ) throws AllBooksException
```

```
    { "select * from book";
```

```
        ArrayList <Book> allBooks = new ArrayList < >( );
```

```
        ...
```



Essayer d'accéder à la base de données et d'exécuter l'instruction SQL.
Boucler sur toutes les lignes de la table book ramenées.
Créer les objets de type Book correspondants et les ajouter à la liste.

```
        return allBooks;
```

```
    }
```

```
}
```

Architecture des applications

1. Model – View – Controller
2. Modèle 3 couches (3-tiers Model)
3. Découpe en couches
4. Data Access Object Pattern

Data Access Object Pattern

Objectif du pattern **Data Access Object** (DAO)

Séparer la persistance des données de l'accès logique aux données

⇒ Indépendance du mécanisme de persistance

Data Access Object Pattern

1. Encapsuler les accès aux données \Rightarrow les placer dans une interface
= Interface public du DAO
2. Créer des classes qui implémentent ces interfaces
= Implémentations du DAO
 - \hookrightarrow connaissent la source de données à laquelle se connecter
(ex: BD, XML, Web Service, ...)
 - \hookrightarrow spécifiques à une source de données

Data Access Object Pattern

Le DAO joue le rôle d'**intermédiaire** entre l'application (business) et le système de persistance des données.

Le DAO **transfère des objets** entre la couche business et le stockage des données.

Le DAO **fournit des opérations** sur les données sans exposer les détails du stockage des données.

Data Access Object Pattern

Avantages

La logique business peut varier indépendamment de la persistance des données :

il suffit d'utiliser la **même interface**.

La couche persistance peut varier :

il suffit que **l'interface soit correctement implémentée**

⇒ **Réduit le couplage** entre la logique business et la logique persistance

Data Access Object Pattern

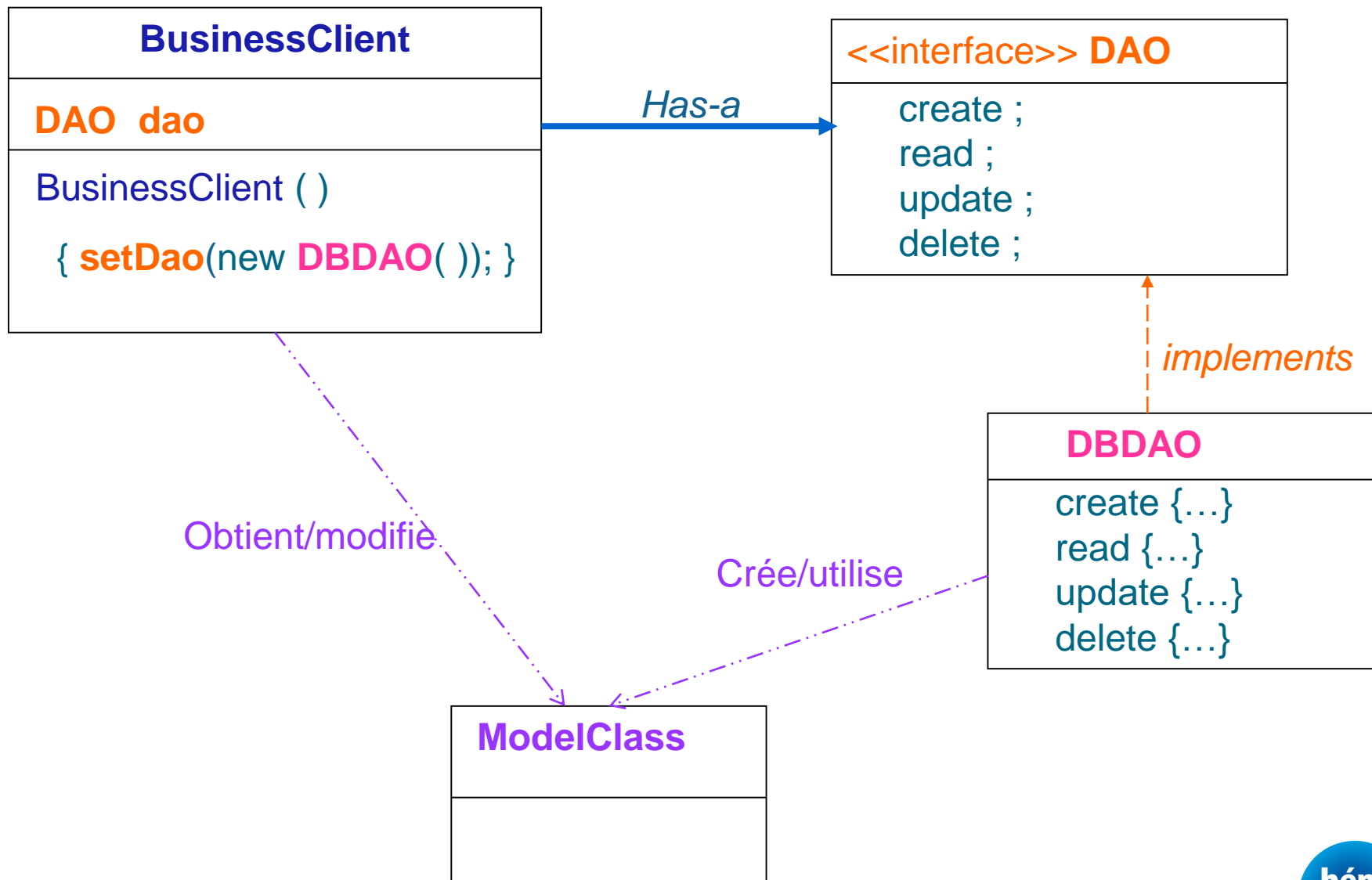
Via l'encapsulation du code des opérations **CRUD**

Create

Read

Uppdate

Delete



Découpe en couches

User Interface

MainJFrame

NewBookPanel

AllBooksModel

AllBooksPanel

Model

Controller

ApplicationController

Book

Business Logic

BookManager

(Couche peu développée dans cet exemple)

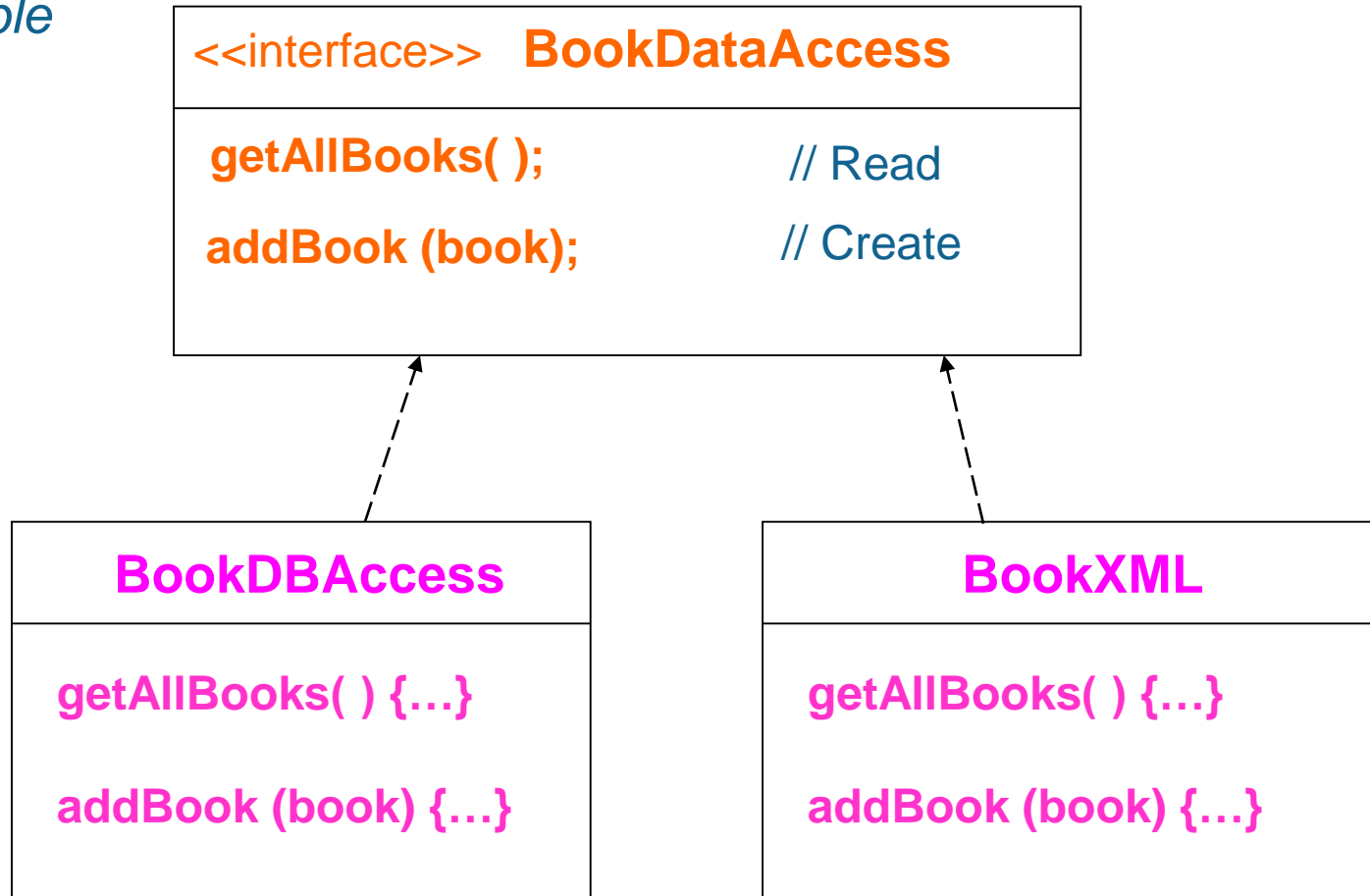
Data Access

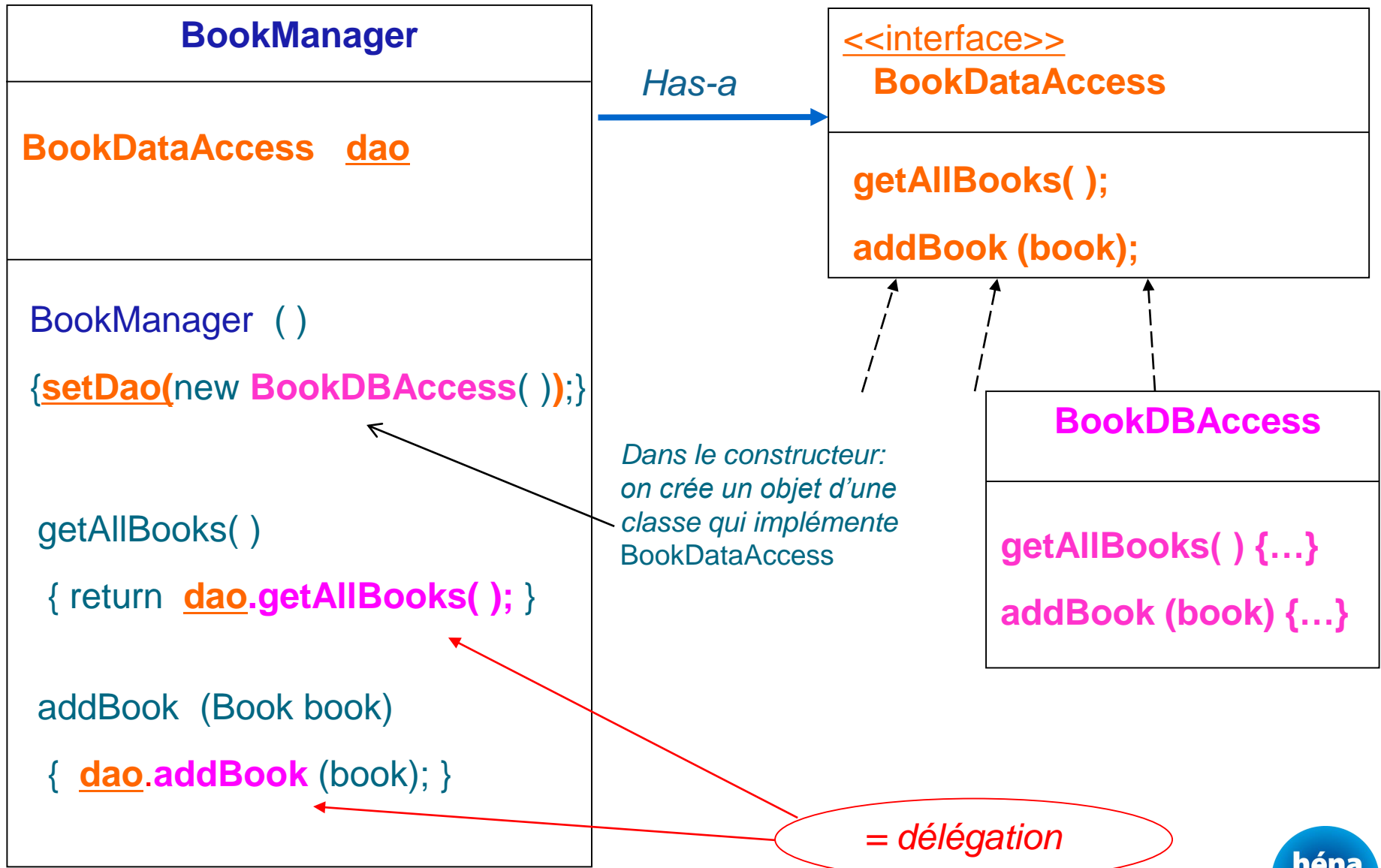
BookDBAccess

Conversion java ↔ SQL

Data Access Object Pattern

Exemple





Architecture des applications

- 1. Model – View – Controller**
- 2. Modèle 3 couches (3-tiers Model)**
- 3. Découpe en couches**
- 4. Data Access Object Pattern**
- 5. Avantages de la découpe en couches**

Avantages de la découpe en couches

Réutilisation des couches dans une autre application

Couche Business

Bibliothèque de fonctions métiers

Couche Data Access

Méthodes d'accès en lecture et écriture aux données persistées
(CRUD)

Couche Model


*NB. Prévoir des variables d'instance de type référence
et non de type primitif pour pouvoir stocker des valeurs inconnues*

Avantages de la découpe en couches

Exemple

```
public class Book
{ private String isbn;
  private Integer pagesNb;
  private String title;
  private GregorianCalendar editionDate;
  ...
}
```

private ~~int~~ pagesNb;



Prévoir des variables d'instance de type référence
et non de type primitif
⇒ afin de gérer les attributs/colonnes facultatifs

Avantages de la découpe en couches

Remplacement d'une couche compète par une autre

Ex: Remplacement de la **couche vue** :
 Composants Swing ⇒ Pages Web

Remplacement de la **couche accès aux données** :
 Base de données relationnelle ⇒ Fichiers XML

Objectif : **Découplage des couches**

Si on remplace une couche par une autre

⇒ **modifier le moins de lignes de code** possible!

Avantages de la découpe en couches

Exemple

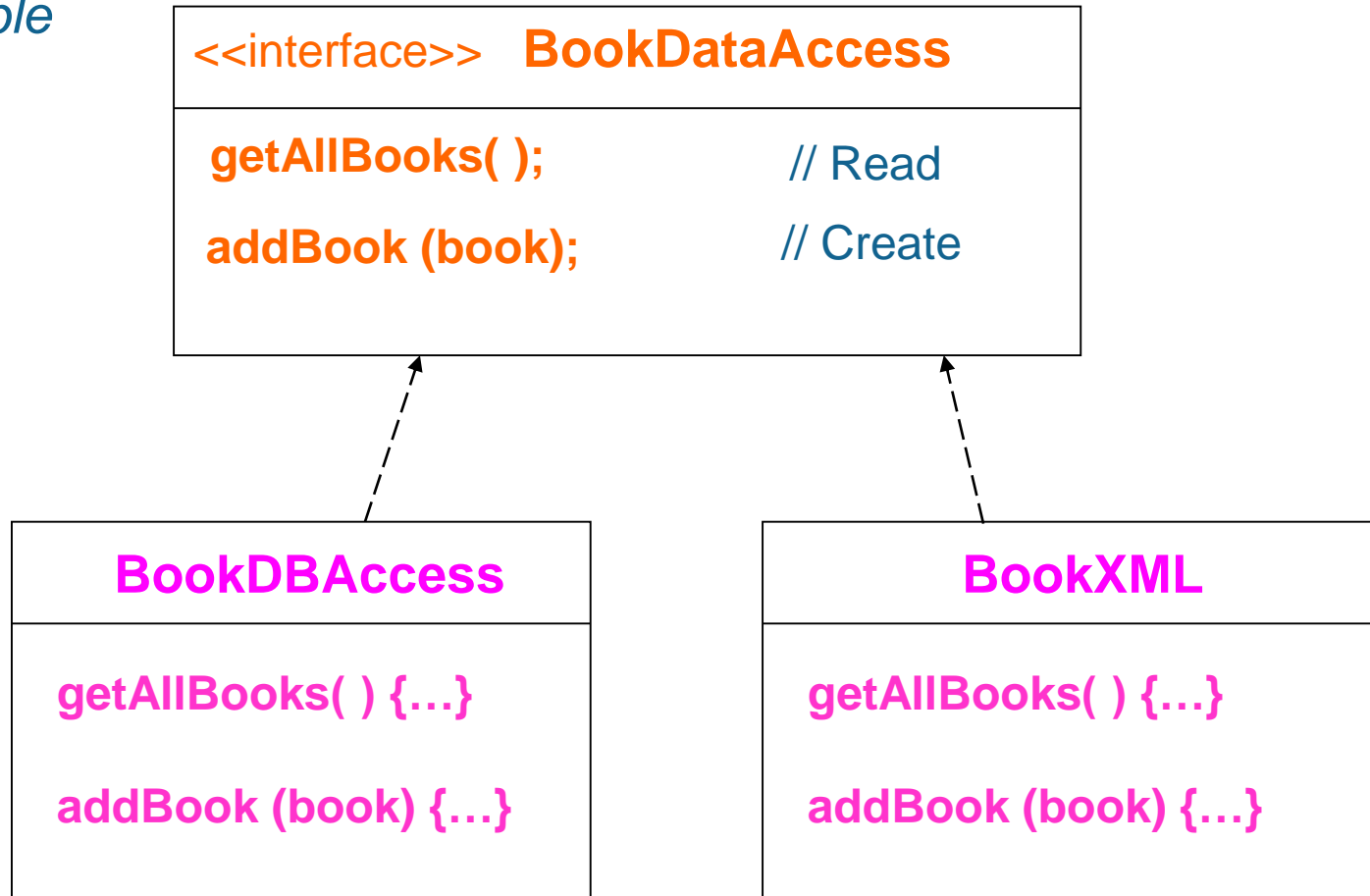
En cas de modification de la persistance des données (BD → fichiers XML), il suffit de prévoir dans la couche accès aux données des classes qui offrent les mêmes méthodes (mêmes déclarations/signatures).

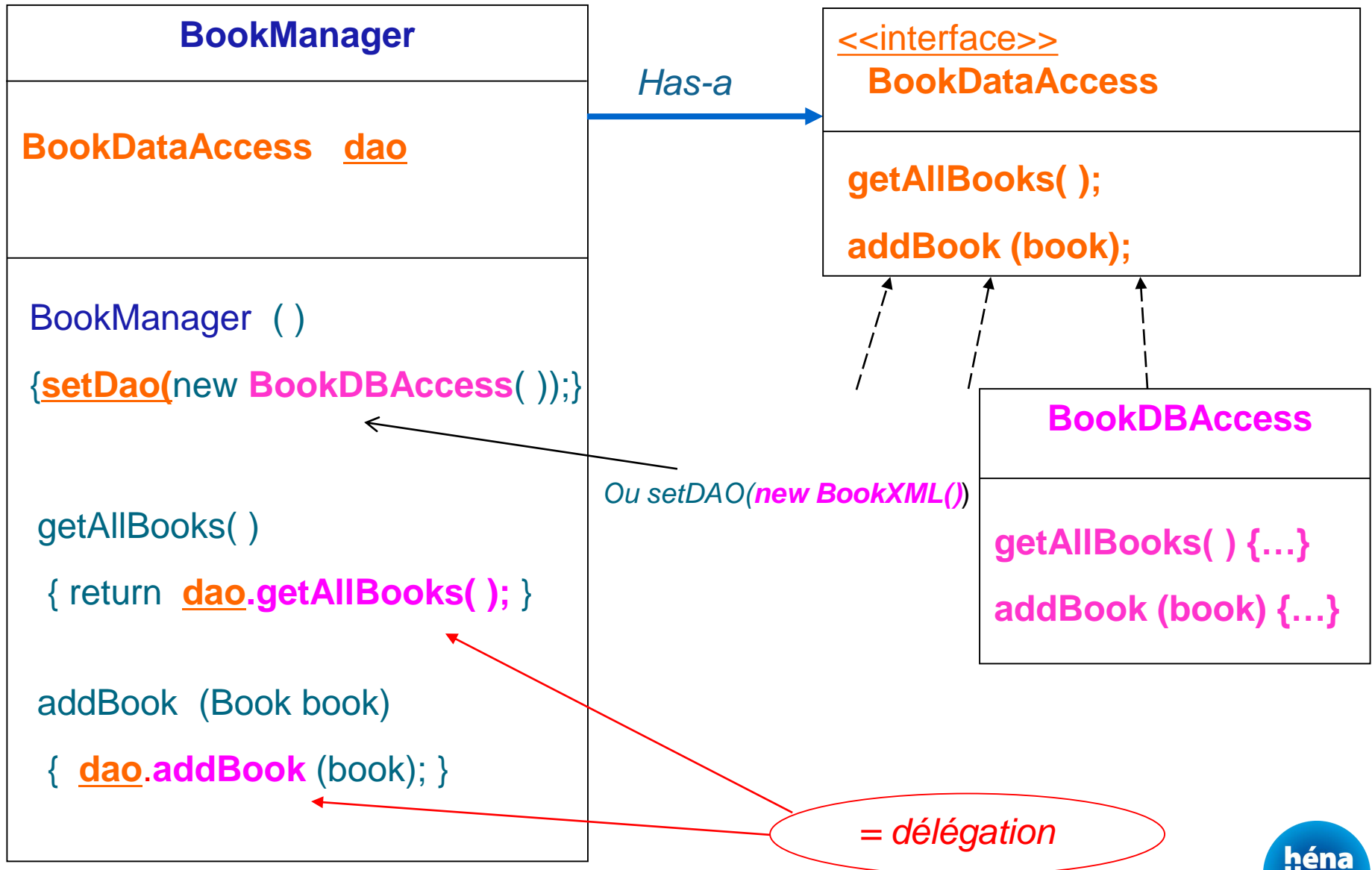
Bonne pratique de programmation (cfr **Design Pattern DAO**) :

- placer ces déclarations de méthodes dans une **interface** et
- créer des **classes qui implémentent** cette interface

Avantages de la découpe en couches

Exemple





Avantages de la découpe en couches

Ainsi, si on change la couche de persistance des données

ex: Base de données \Rightarrow fichiers XML

\Rightarrow Il n'y a que l'initialisation de la variable dao à modifier dans le constructeur de BookManager.

Donc **une seule ligne de code à modifier dans la couche Business !**

Avantages de la découpe en couches

Attention: ne **pas utiliser de static** dans les méthodes qui sont appelées par délégation entre couches!

Sinon, perte du bénéfice de la découpe en couches en cas de modifications d'une couche : il faudrait modifier le nom de la classe dans toutes les instructions d'appel des méthodes

Ex: si modification du stockage de données BD relationnelle \Rightarrow xml

```
BookDBAccess. getAllBooks( );  
BookDBAccess. addBook (book);  
BookDBAccess. ...
```



```
BookXML.getAllBooks( );  
BookXML.addBook (book);  
BookXML. ...
```


Avantages de la découpe en couches

Sécurité

Chaque couche est une boîte noire sécurisée (cf point 7)

⇒ Doit tester ses entrées

Architecture des applications

- 1. Model – View – Controller**
- 2. Modèle 3 couches (3-tiers Model)**
- 3. Découpe en couches**
- 4. Data Access Object Pattern**
- 5. Avantages de la découpe en couches**
- 6. Propagation des exceptions entre couches**

Scinder détection et traitement de l'erreur

Couche 2 :

Traitement de
l'erreur

Couche 1 :

Détection de
l'erreur

Propagation

Scinder détection et traitement de l'erreur

Séparation de la **détection** d'un incident (problème ou cas d'erreur)
de sa **prise en charge** :

cas d'erreur **déecté** dans le composant 1
et **traité** dans le composant 2



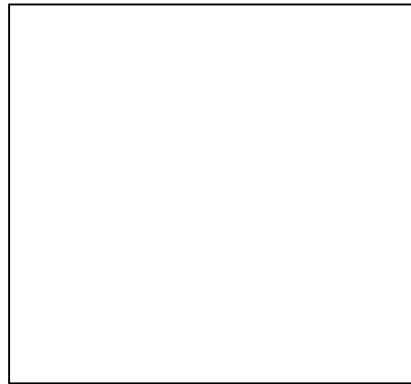
décide **comment réagir**

*ex : arrêter le programme,
lancer une procédure particulière,
avertir l'utilisateur via une boîte de dialogue,
afficher un simple message sur la console, ...*

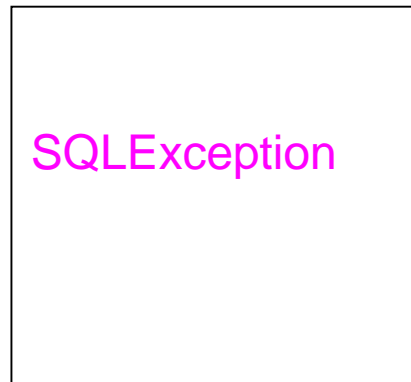
Transformation des SQLExceptions

Exemple:

Couche interface utilisateur :



Couche accès aux données :



~~Propagation de
SQLException?~~



Transformation:
SQLException en autre Exception

Transformation des SQLExceptions

...method (...) **throws OtherException**  Propagation

```
{ try {  
    ...  
}
```

→ Susceptible de générer une SQLException

```
catch (SQLException exception)  
{  
    throw new OtherException (...);  
}
```

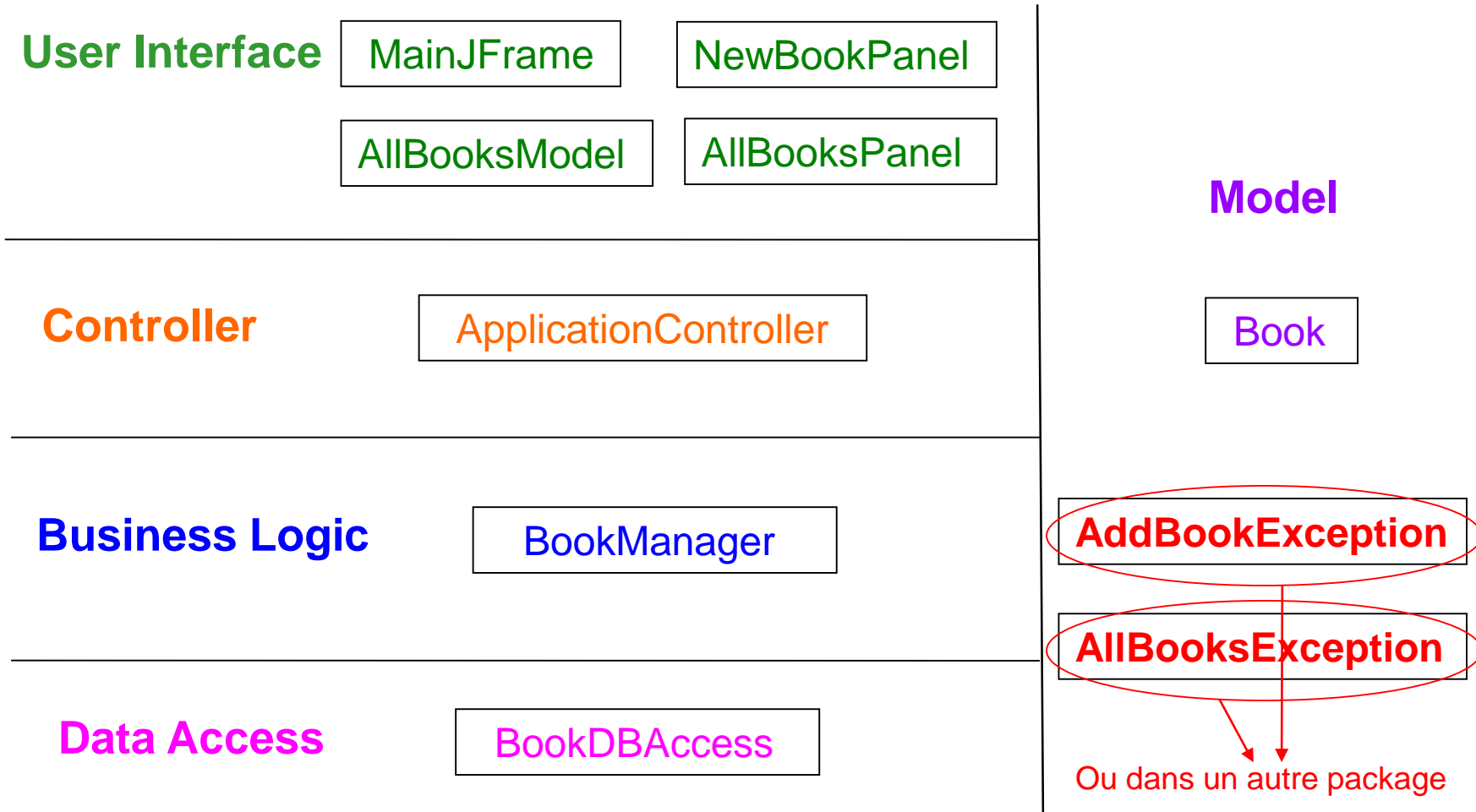
Exemples : un code correspondant au type d'exception
ou **exception.getMessage()**
ou n'importe quelle autre information sur l'erreur qui a eu lieu

Transformation des SQLExceptions

NB. Transformation des SQLExceptions :

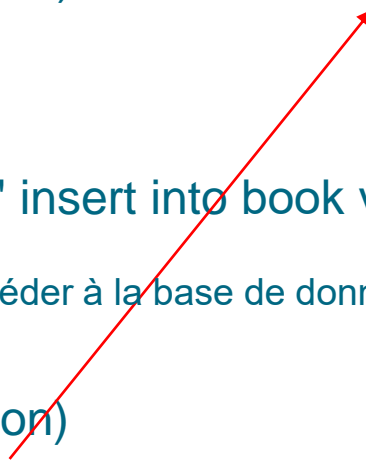
- en une classe Exception Java existante
- en toute autre classe créée par le programmeur
qui extends (une sous-classe de) Exception

Découpe en couches



Transformation des SQLExceptions

```
public class BookDBAccess {  
  
    public void addBook (Book book) throws AddBookException  
    { try  
        {  
            String instructionSQL = " insert into book values (...)" ;  
            ... —————→ Essayer d'accéder à la base de données et d'exécuter l'instruction SQL  
        }  
        catch (SQLException exception)  
        { throw new AddBookException (exception.getMessage( )); }  
    }  
}
```



Transformation des SQLExceptions

```
public class BookDBAccess {  
    public ArrayList <Book> getAllBooks( ) throws AllBooksException  
    { ArrayList <Book> allBooks = new ArrayList < >( );  
      try  
      { String instructionSQL = "select * from book";  
        ...  
      }  
      catch (SQLException exception)  
      { throw new AllBooksException (exception.getMessage( )); }  
      return allBooks;  
    }
```

... —————→ **Essayer** d'accéder à la base de données et d'exécuter l'instruction SQL.
Boucler sur toutes les lignes de la table book ramenées.
Créer les objets de type Book correspondants et les ajouter à la liste.

Architecture des applications

1. Model – View – Controller
2. Modèle 3 couches (3-tiers Model)
3. Découpe en couches
4. Data Access Object Pattern
5. Avantages de la découpe en couches
6. Propagation des exceptions entre couches
7. Tests et sécurité

Tests et sécurité

Une couche = une boîte noire

- indépendante des autres couches
- sécurisée au maximum

Les valeurs en entrée doivent être testées!

Chaque couche doit effectuer ses propres tests sur les valeurs en entrée

Couche vue

Validation du formulaire avant de créer des objets et de les envoyer aux autres couches

Tests et sécurité

Erreurs possibles sur une valeur introduite par un utilisateur :

- Champ obligatoire non rempli
- Valeur numérique contenant des caractères
- Nombre négatif ou nul (si valeur positive attendue)
- Valeur non comprise dans la liste des valeurs permises
- Nouveau login déjà existant
- Format non respecté (ex: email)

Voire **malveillance** volontaire

(ex: tentative d'injection SQL (cfr Chap. 10))



Couche Business

= Librairie de fonctions métiers

⇒ Tester les valeurs des arguments (= entrées) des méthodes

⇒ Si pas valables ⇒ Exception remontée

Couche Model

Empêcher de créer des "mauvais" objets

(c'est-à-dire avec des mauvaises valeurs dans leurs variables d'instance)

- ⇒ Variables d'instance **privées**
- ⇒ **Settors** publiques avec rôle de **filtres**
- ⇒ OK seulement si settors **appelés dans le constructeur**

Couche Model

Contre-exemple

```
public class Division
{ private int denominator; ...

  public int getDenominator() { return denominator; }

  public void setDenominator (int denominator) throws DenominatorException
  { if (denominator == 0) throw new DenominatorException();
    else this.denominator = denominator; }

  public Division (int denominator, ...) throws DenominatorException
  { this.denominator = denominator; } ⇒ { setDenominator(denominator); }
}
```

Base de données

Maintenir la BD dans un état cohérent via :

Bons types de colonnes

Not null éventuels

Primary key

Foreign Key

Checks

Tests et sécurité

Exemple: quantité commandée introduite par l'utilisateur

Couche vue

- Composants swing :
 - Tester la valeur introduite et
 - Afficher une boîte de dialogue si pas OK
- Page HTML : tester en **JavaScript**

Couche business

Tester la quantité commandée en *Java* et remonter une *exception* si pas OK

Base de données

Prévoir des **checks** dans le script SQL de création de la base de données
ex: *check (quantiteCommande > 0)*

Tests et sécurité

Intérêts de placer les mêmes tests dans différentes couches?

Intérêt de retester les valeurs en entrée dans la couche Business :

le Javascript peut être désactivé par les internautes!

Intérêt de placer des checks SQL dans la base de données :

une base de données n'est pas liée à une application,

elle peut être **réutilisée** dans le futur par **d'autres applications**