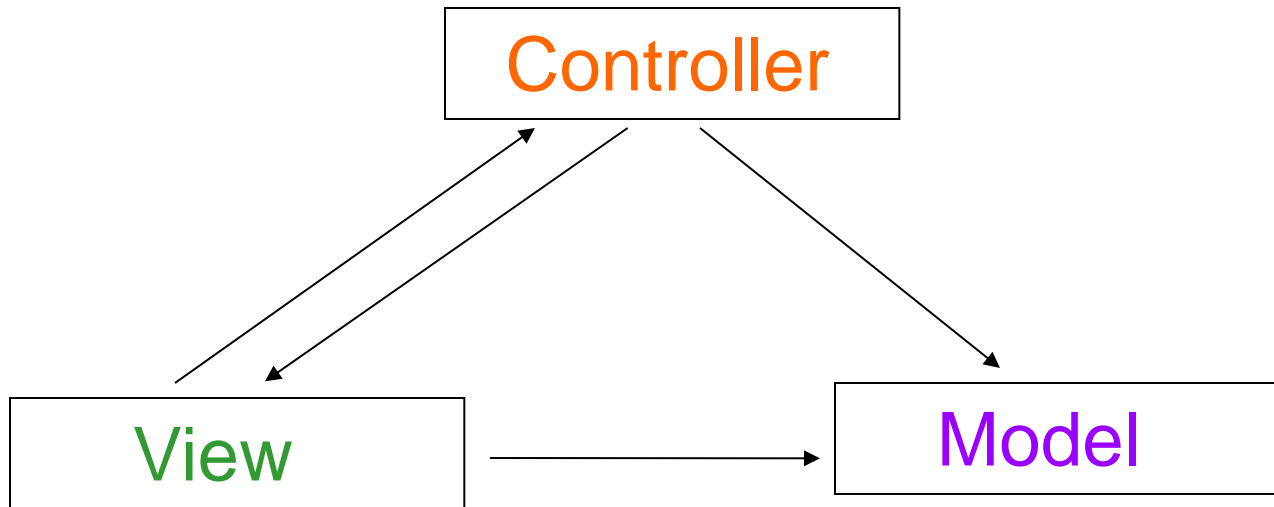


8. Architecture des applications

- 8.1. Model – View – Controller (MVC)

Decoupling of data
access and data
presentation



Data presentation

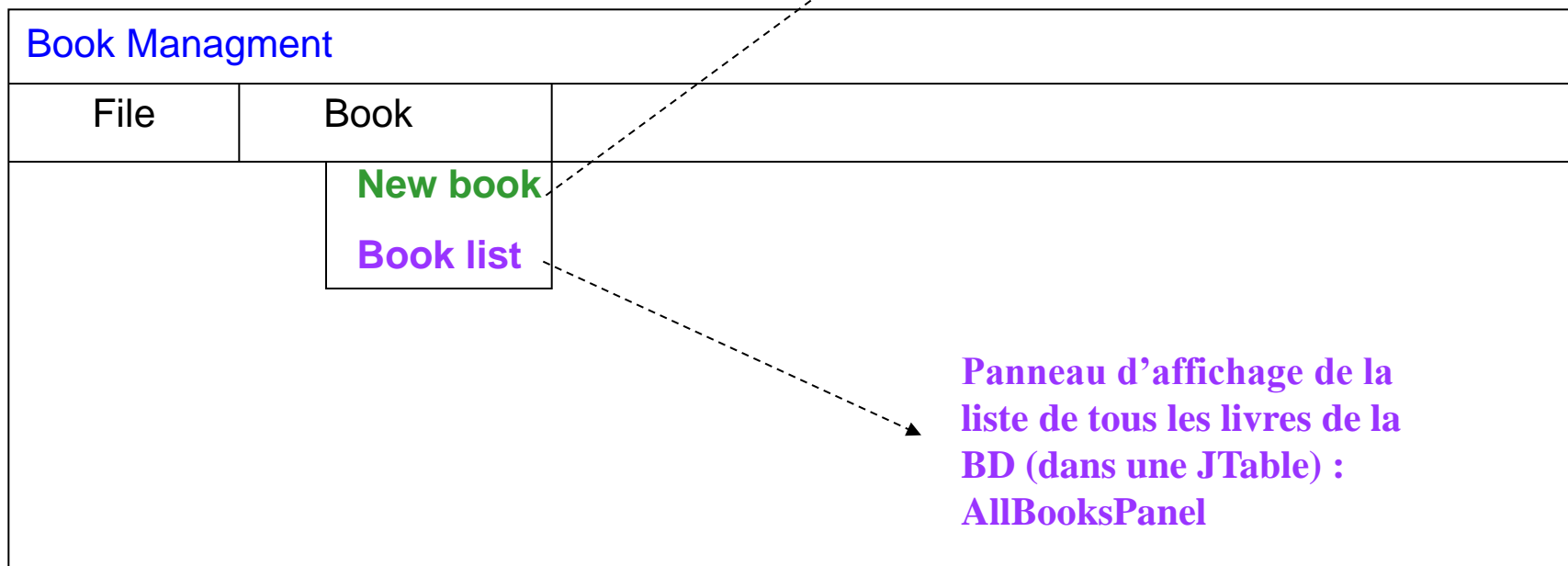
User interaction

Data access

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

View

MainJFrame

NewBookPanel

AllBooksPanel

Controller

ApplicationController

Model

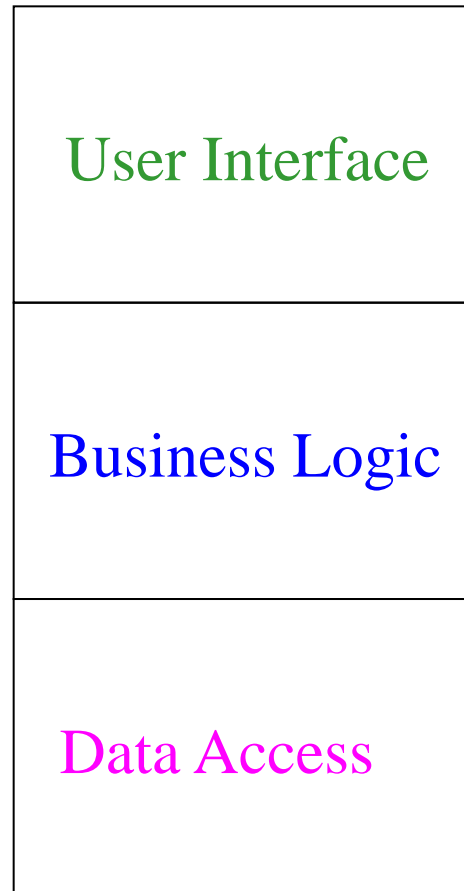
8. Architecture des applications

- 8.1. Model – View – Controller (MVC)
- 8.2. Modèle 3 couches (3-tiers Model)



Surtout au niveau
physique:

si l'application est
distribuée sur des
machines différentes



**Couche présentation de
l'application**

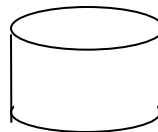
*Ex: pages Web exécutées
chez le client*

**Couche métier: traitement
de l'information**

*Ex: sur serveur dans
l'entreprise*

**Couche accès et
stockage des données**

*Ex: serveur de BD à
distance*



User Interface

MainJFrame

NewBookPanel

AllBooksPanel

Business Logic

BookManager

(Couche peu développée dans cet exemple: pourrait contenir des calculs élaborés (ex: factures, fiches de paie, ...), statistiques, intelligence artificielle, ...)

Book

Data Access

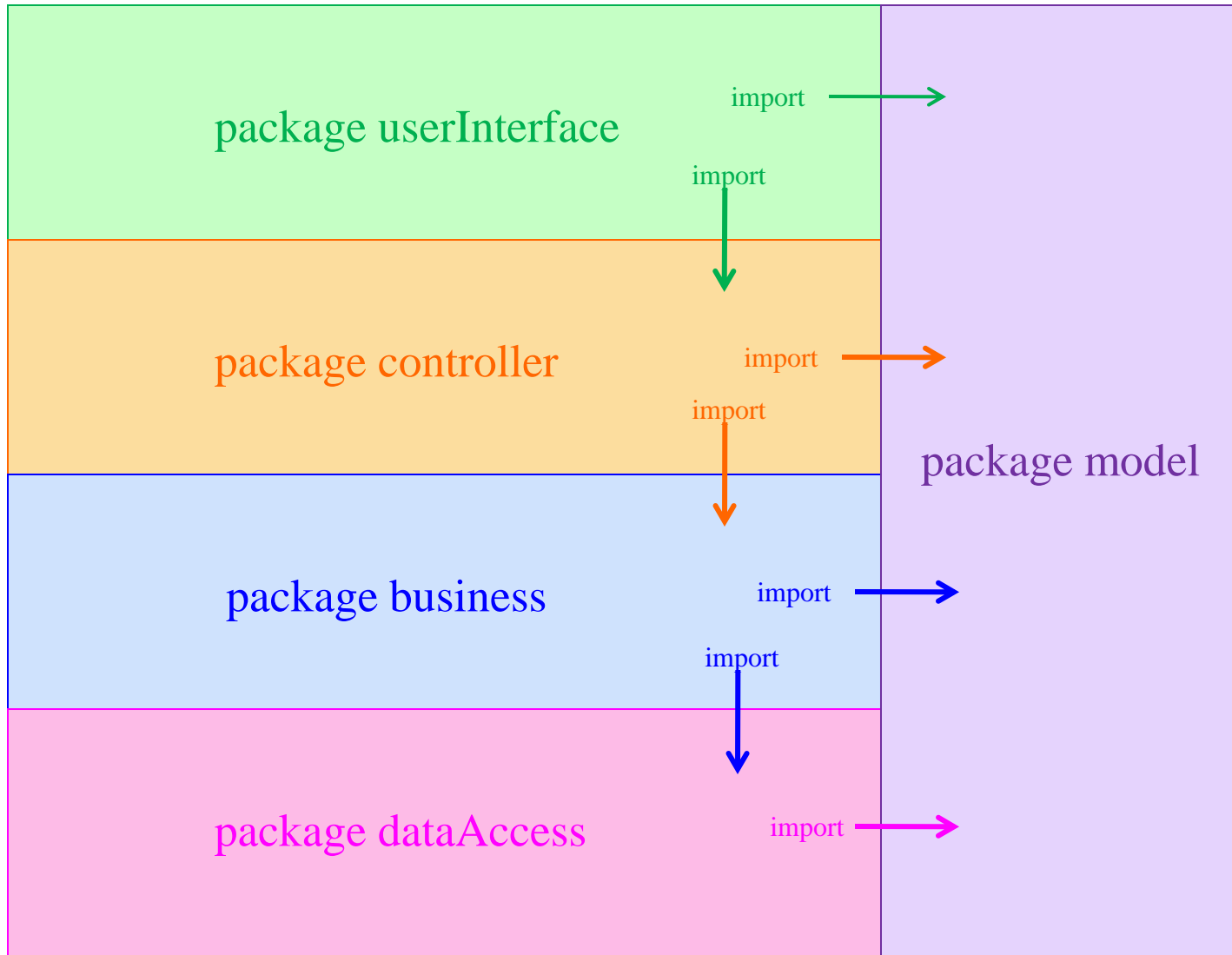
BookDBAccess

Conversion java ↔ SQL

8. Architecture des applications

- 8.1. Model – View – Controller (MVC)
- 8.2. Modèle 3 couches (3-tiers Model)
- 8.3. Découpe en couches

Structure des packages



User
Interface

MainJFrame

NewBookPanel

AllBooksPanel

Model

Controller

ApplicationController

Business
Logic

BookManager

Book

(Couche peu développée dans cet exemple)

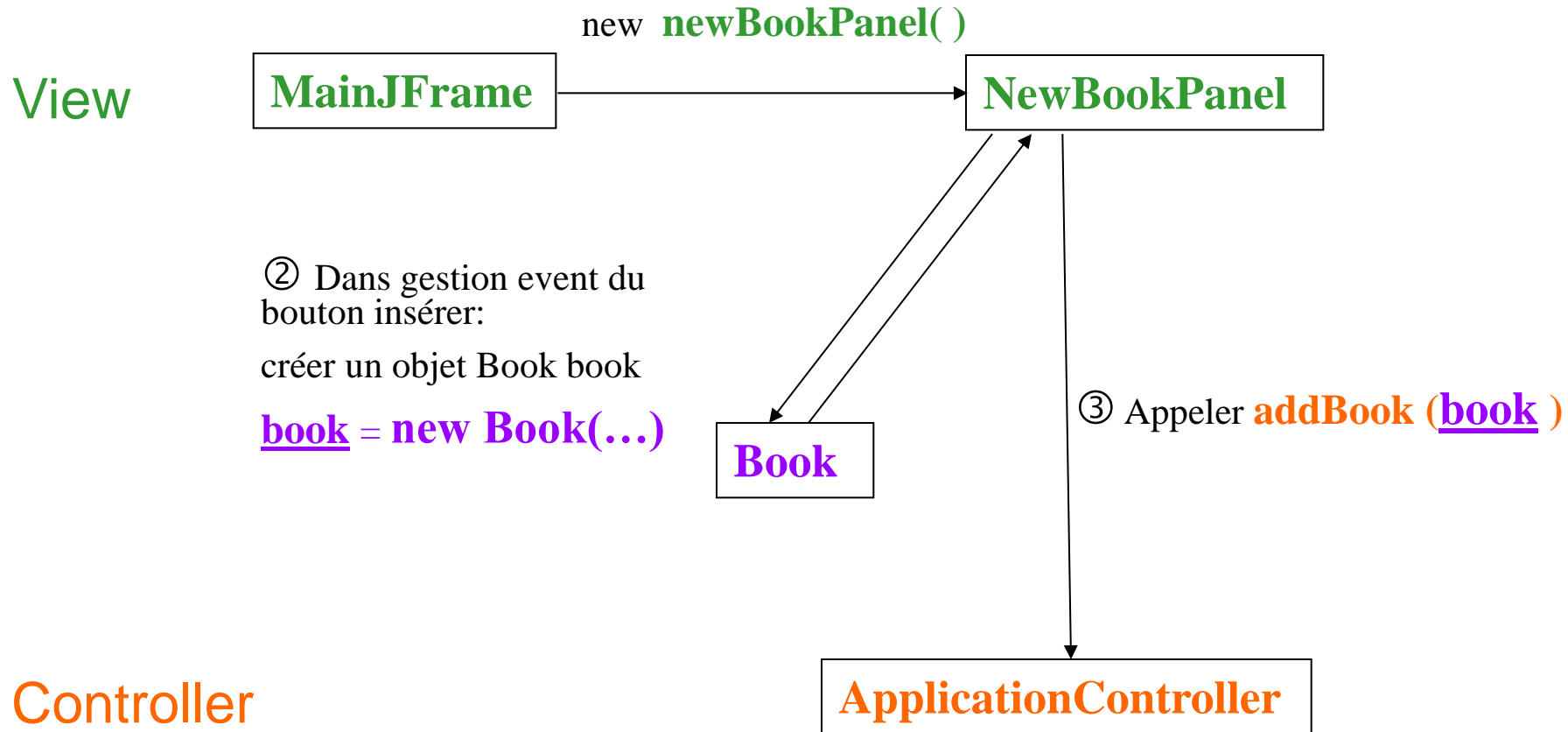
Data
Access

BookDBAccess

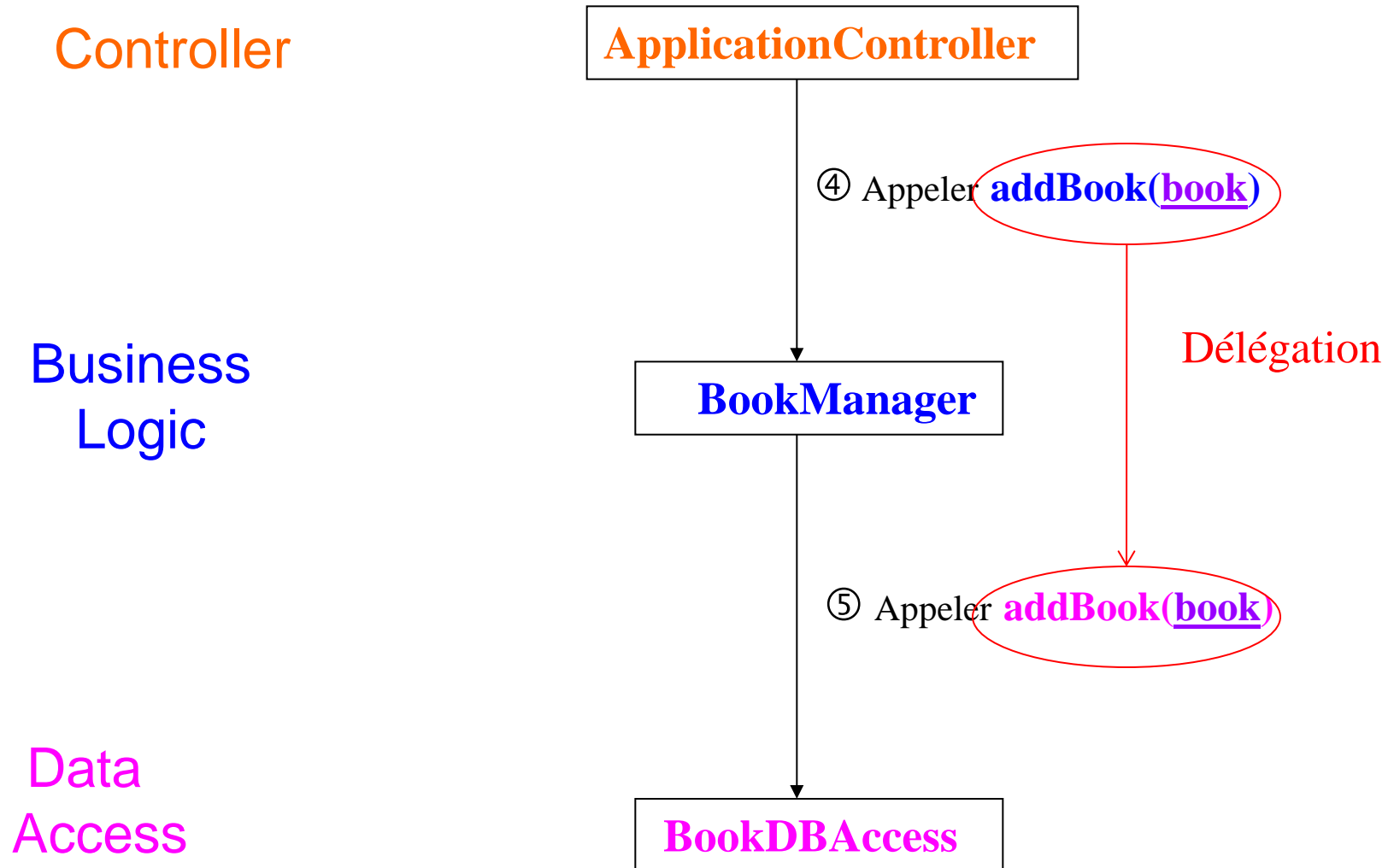
Conversion java ⇔ SQL

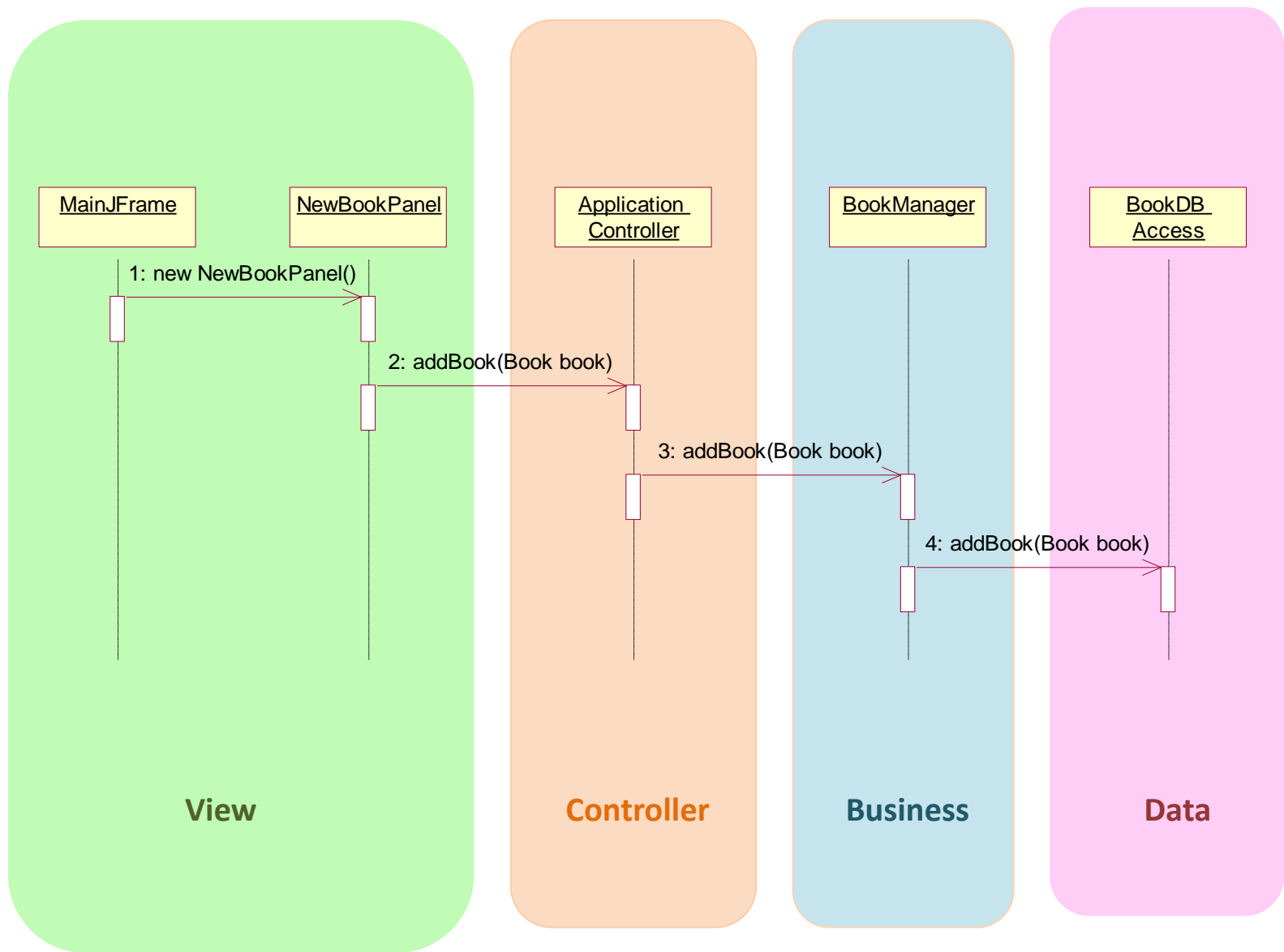
Si choix de l'option de menu *New book*:

① Dans gestion event de l'option de menu New book:

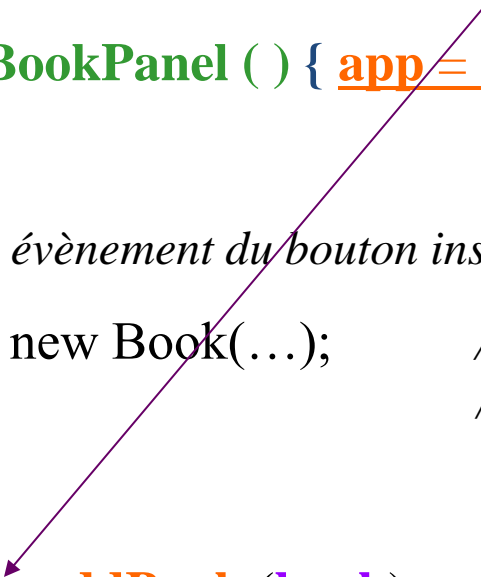


Si choix de l'option de menu *New book* (suite):





```
public class NewBookPanel extends JPanel {  
    private ApplicationController app;  
    public NewBookPanel ( ) { app = new ApplicationController ( ); }  
    ...  
    // Dans gestion évènement du bouton insérer:  
    Book book = new Book(...);           // création du livre à partir des données  
                                           // introduites par l'utilisateur  
    try  
        { app.addBook (book);  
        }  
    catch (AddBookException ex) // cfr point 8.5  
        { ...  
        }  
}
```



```
public class ApplicationController {  
    private BookManager bm ;
```

```
    public ApplicationController ( )  
    {  
        bm = new BookManager( );  
    }
```

```
    public void addBook(Book book) throws AddBookException  
    {  
        bm.addBook(book);  
    }  
}
```

Délégation

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


```
    public BookManager ( )  
    {  
        bda = 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)
```

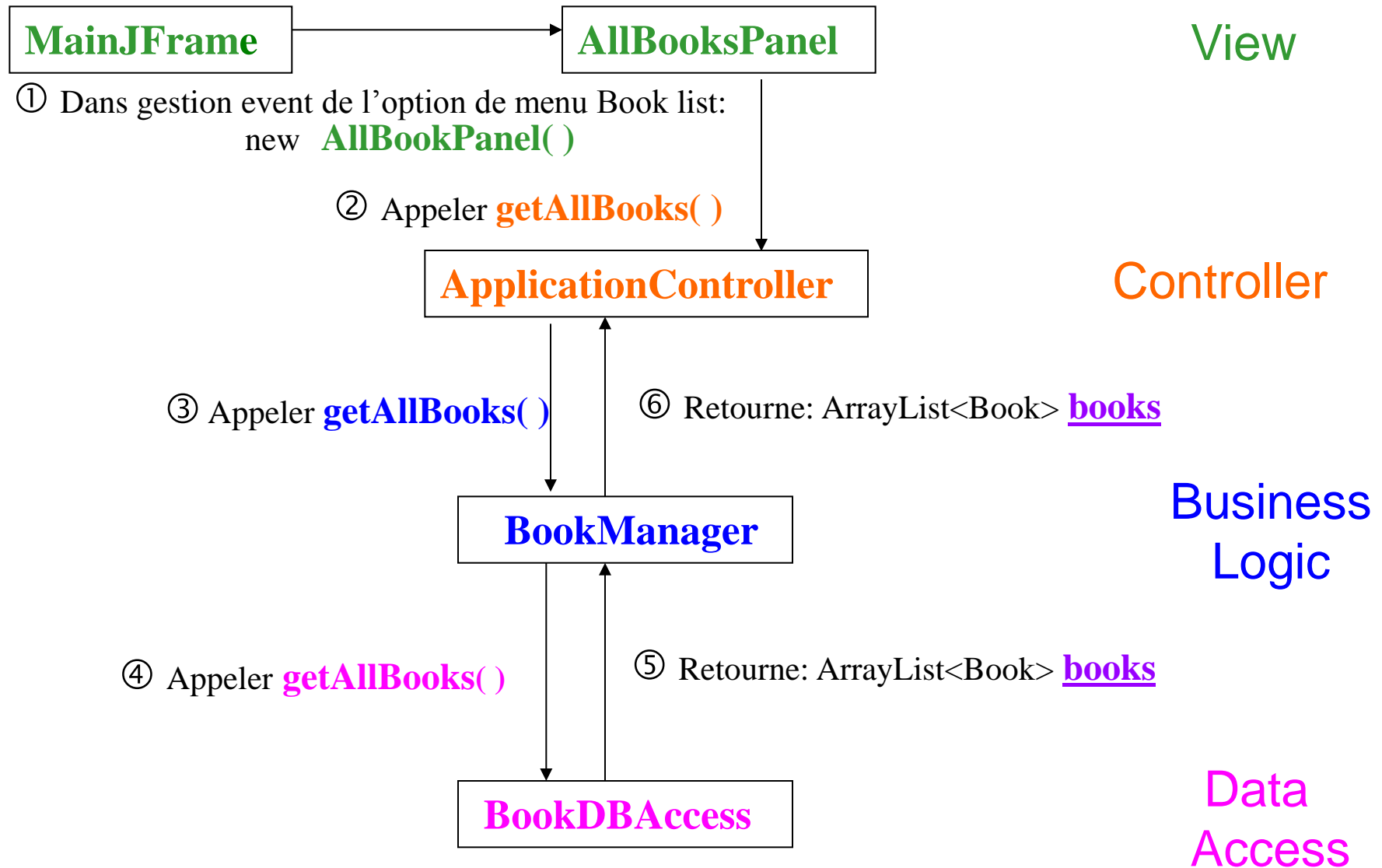
```
        bda.addBook (book);
```

Délégation


```
public class BookDBAccess {  
  
    public void addBook (Book book) throws AddBookException  
    {  
        "insert into Book values (...);"  
        // accéder à la base de données et exécuter l'instruction SQL  
    }  
}
```



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



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

View

AllBooksPanel

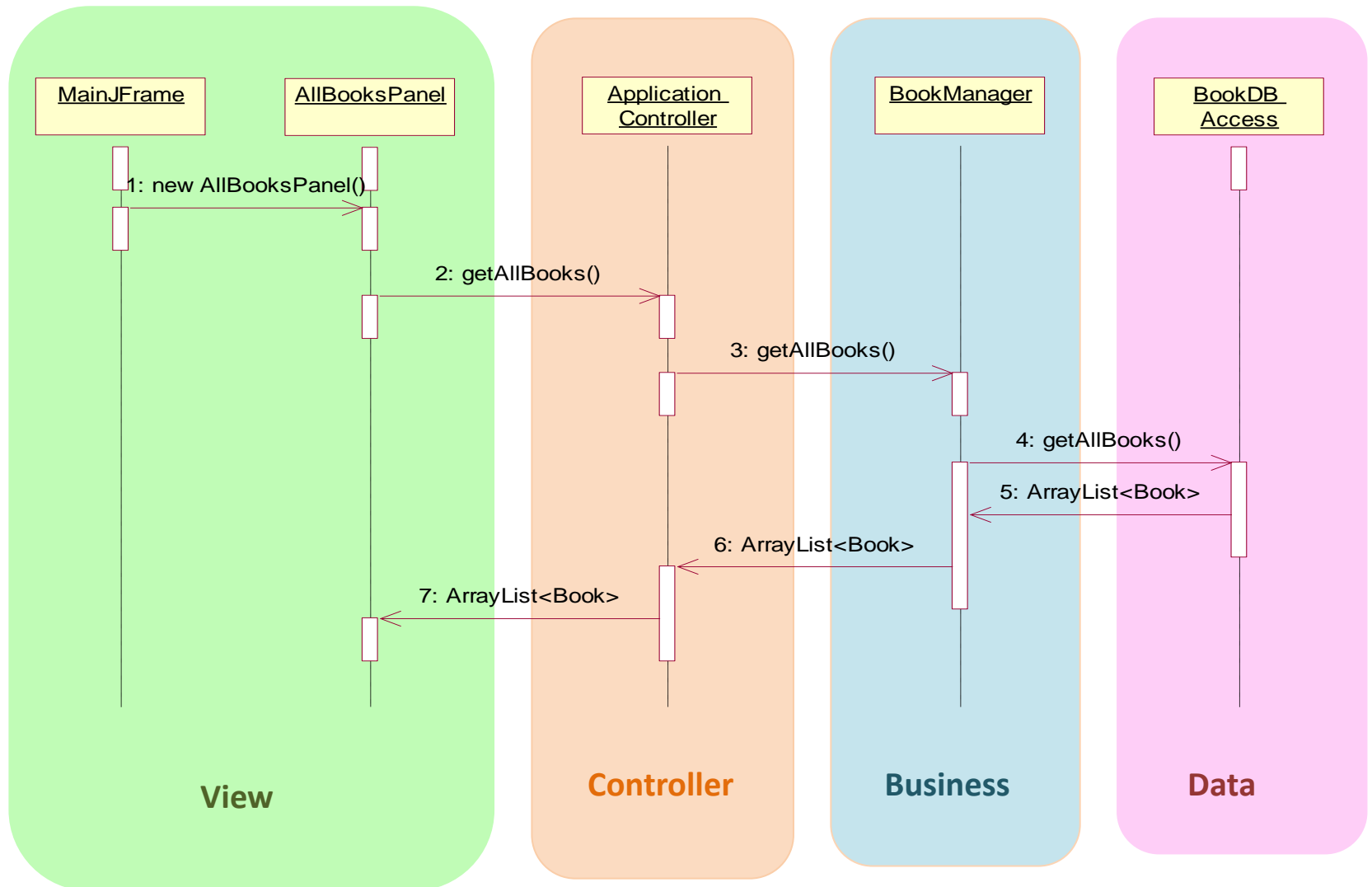
⑦ model = new AllBooksModel(books)

AllBooksModel

⑥ Retourne: ArrayList<Book> books


Controller

ApplicationController



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

public AllBooksPanel ( )
{ app = new ApplicationController ( );
  try
    { ArrayList<Book> books = app.getAllBooks( );
      // afficher les livres à l'écran
    }
  catch (AllBooksException ex)           // cfr point 8.5
    { ... }
}
```



```
public class ApplicationController {
```

```
    private BookManager bm ;
```

```
    public ApplicationController ( )
```

```
{
```

```
    bm = new BookManager( );
```

```
}
```

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

```
{
```

```
    return bm.getAllBooks( );
```

```
}
```

```
}
```

Délégation

```
public class BookManager {  
    private BookDBAccess bda ;  
  
    public BookManager ( )  
    {  
        bda = new BookDBAccess( );  
    }  
    public ArrayList <Book> getAllBooks( ) throws AllBooksException  
    { ArrayList <Book> bookList = bda.getAllBooks( );  
        ...           // traitements éventuels sur la liste de livres  
        return bookList ;  
    }  
}
```

Délégation

```
public class BookDBAccess {  
  
    public ArrayList <Book> getAllBooks() throws AllBooksException  
    {  
        ...  
        "select * from Book";  
  
        ArrayList <Book> allBooks = new ArrayList <Book>( );  
  
        // accéder à la base de données et 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;  
    }  
}
```


8. Architecture des applications

- 8.1. Model – View – Controller (MVC)
- 8.2. Modèle 3 couches (3-tiers Model)
- 8.3. Découpe en couches
- 8.4. Avantages

Avantage de la découpe en couches

En cas de **remplacement d'une couche complète 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!

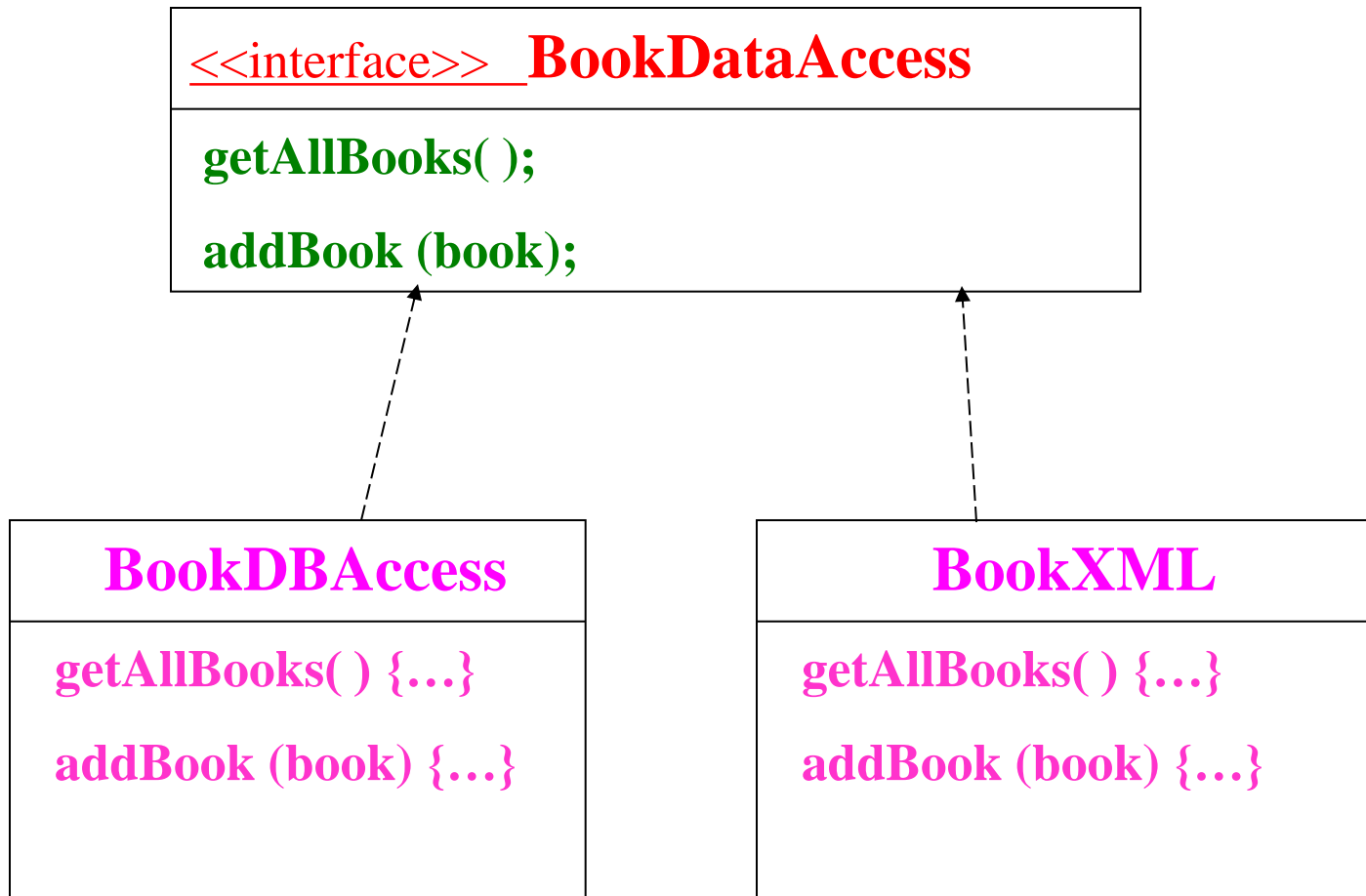
Avantage de la découpe en couches

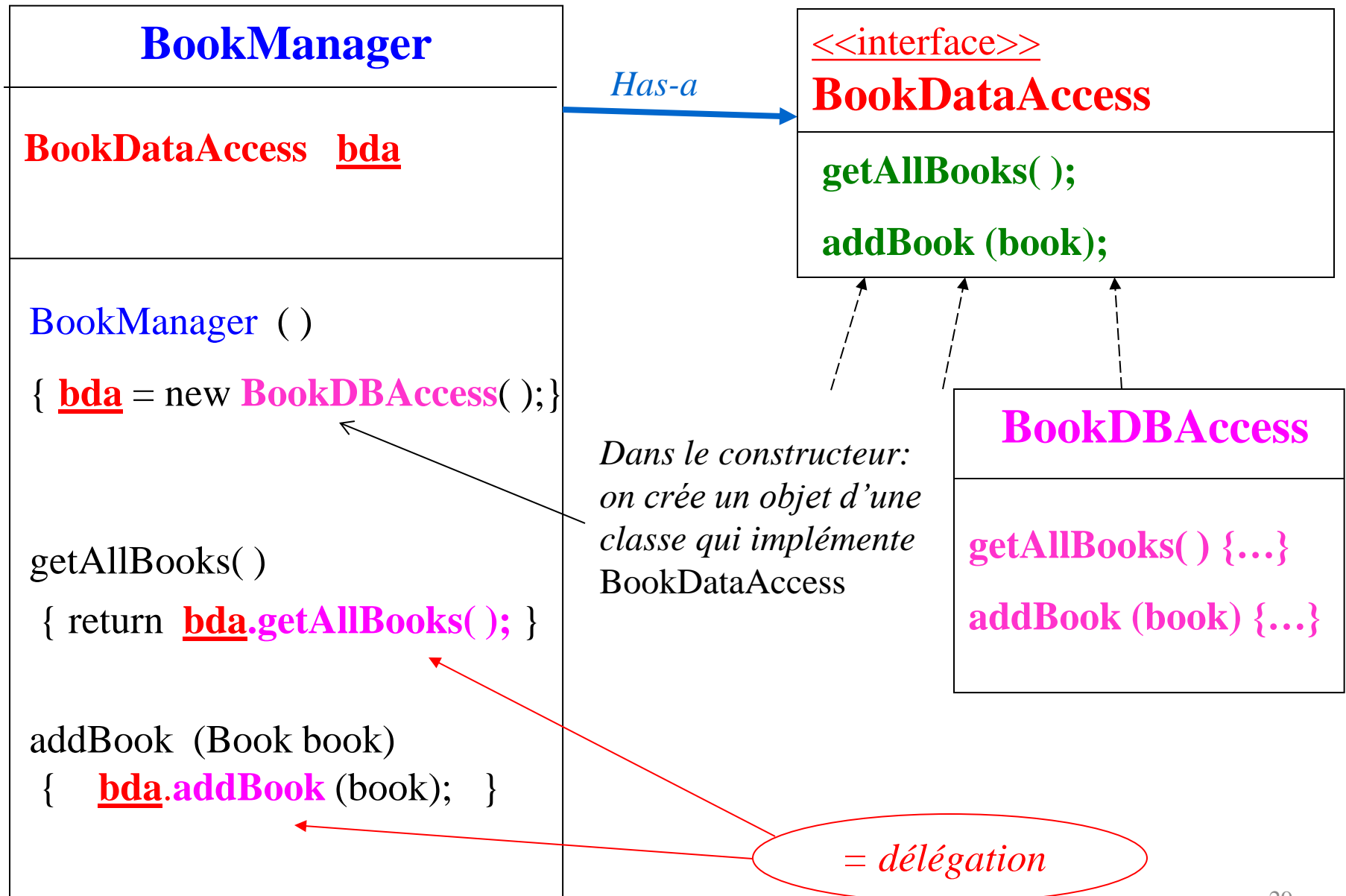
Exemple:

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**) :

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





Ainsi, si on change la couche de persistance des données

ex: Base de données \Rightarrow fichiers XML

il n'y a que l'initialisation de la variable **bda** à modifier dans le constructeur de BookManager.

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

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 → xml

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



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

8. Architecture des applications

- 8.1. Model – View – Controller (MVC)
- 8.2. Modèle 3 couches (3-tiers Model)
- 8.3. Découpe en couches
- 8.4. Avantages
- 8.5. Propagation des exceptions entre les couches

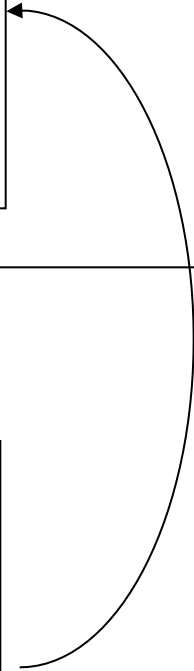
Couche 2 :

Traitement de
l'erreur

Couche 1 :

Detection de
l'erreur

Propagation



séparation de la *détection* d'un incident (problème ou cas d'erreur)

de sa *prise en charge*:

cas d'erreur **déecté** dans la couche 1

et **traité** dans la couche 2



décide **comment réagir**

ex : arrêter le programme,

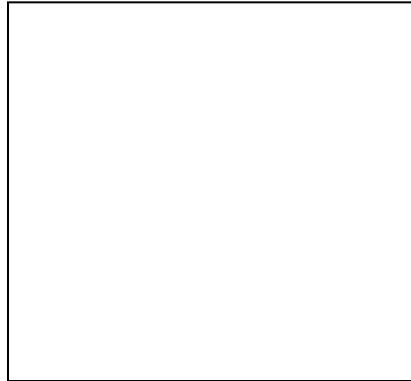
lancer une procédure particulière,

avertir l'utilisateur via une boîte de dialogue,

afficher un simple message sur la console, ...

Exemple:

Couche interface utilisateur :




Couche accès aux données :



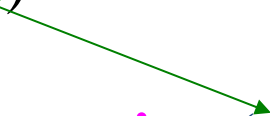
~~Propagation de
SQLException?~~



Transformation:
SQLException
en autre Exception

...myMethod (...) **throws MyException**  propagation

{try {
 ... → *Susceptible de générer une SQLException*
}

catch (**SQLException e**)
 {
 throw new MyException (e); 
 }
}

NB: Transformation:

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

User Interface

MainJFrame

NewBookPanel

AllBooksPanel

Model

Controller

ApplicationController

Business Logic

BookManager

(Couche peu développée dans cet exemple)

Data Access

BookDBAccess

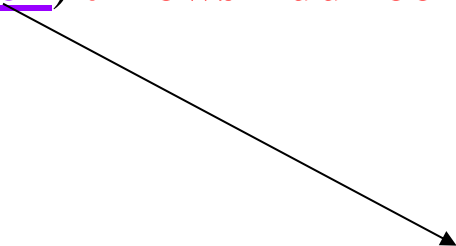
Conversion java \Leftrightarrow SQL

Book

AddBookException

AllBooksException

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



```
public class BookDBAccess {  
    public ArrayList <Book> getAllBooks( ) throws AllBooksException  
    { ArrayList <Book> allBooks = new ArrayList <Book>( );  
        try  
        {   String instructionSQL = "select * from Book";  
            // accéder à la base de données et 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 collection  
        }  
        catch (SQLException e)  
        { throw new AllBooksException (e.getMessage( )); }  
        return allBooks;  
    }  
}
```


8. Architecture des applications

- 8.1. Model – View – Controller (MVC)
- 8.2. Modèle 3 couches (3-tiers Model)
- 8.3. Découpe en couches
- 8.4. Avantages
- 8.5. Propagation des exceptions entre les couches
- 8.6. 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

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

Voire **malveillance** volontaire

(ex: tentative d'injection SQL (cfr Chap. 9))

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* en *SQL* dans la base de données
ex: *check (quantiteCommande > 0)*

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**