



FORMATION ARCHITECTURE LOGICIELLE

Laurent YAO

EPSI 19-20 Mai 2016



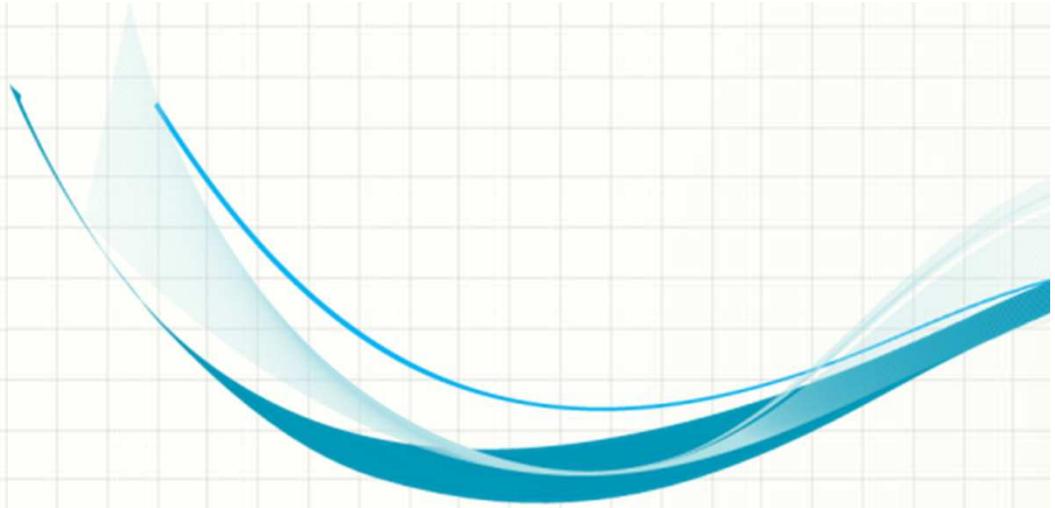
Plan du cours

- Partie 1 : Généralités sur le modèle MVC
 - Cours 1 : architecture MVC/événements : structure d'une application
- Partie 2 : Le modèle : cas de la gestion de la persistance Java
 - Cours 2 : Java Database Connectivity (JDBC)
- Partie 3 : Les view – cas des interfaces graphiques Java
 - Cours 3 : Les swing GUI de Java



Plan du cours (suite...)

- Partie 4 : Le contrôleur – gestion des évènements et traitements métiers
 - Cours 4 : gestion des évènements et traitements



Partie 1 : Généralités sur le modèle MVC



Cours 1 : Architecture MVC

- Principe général
- Architecture MVC
- MVC : le modèle (Model), la persistance
- MVC : la vue (View)
- MVC : le contrôleur (Controller)
- Rôle des éléments
- Importance de MVC



Architecture MVC : principe général

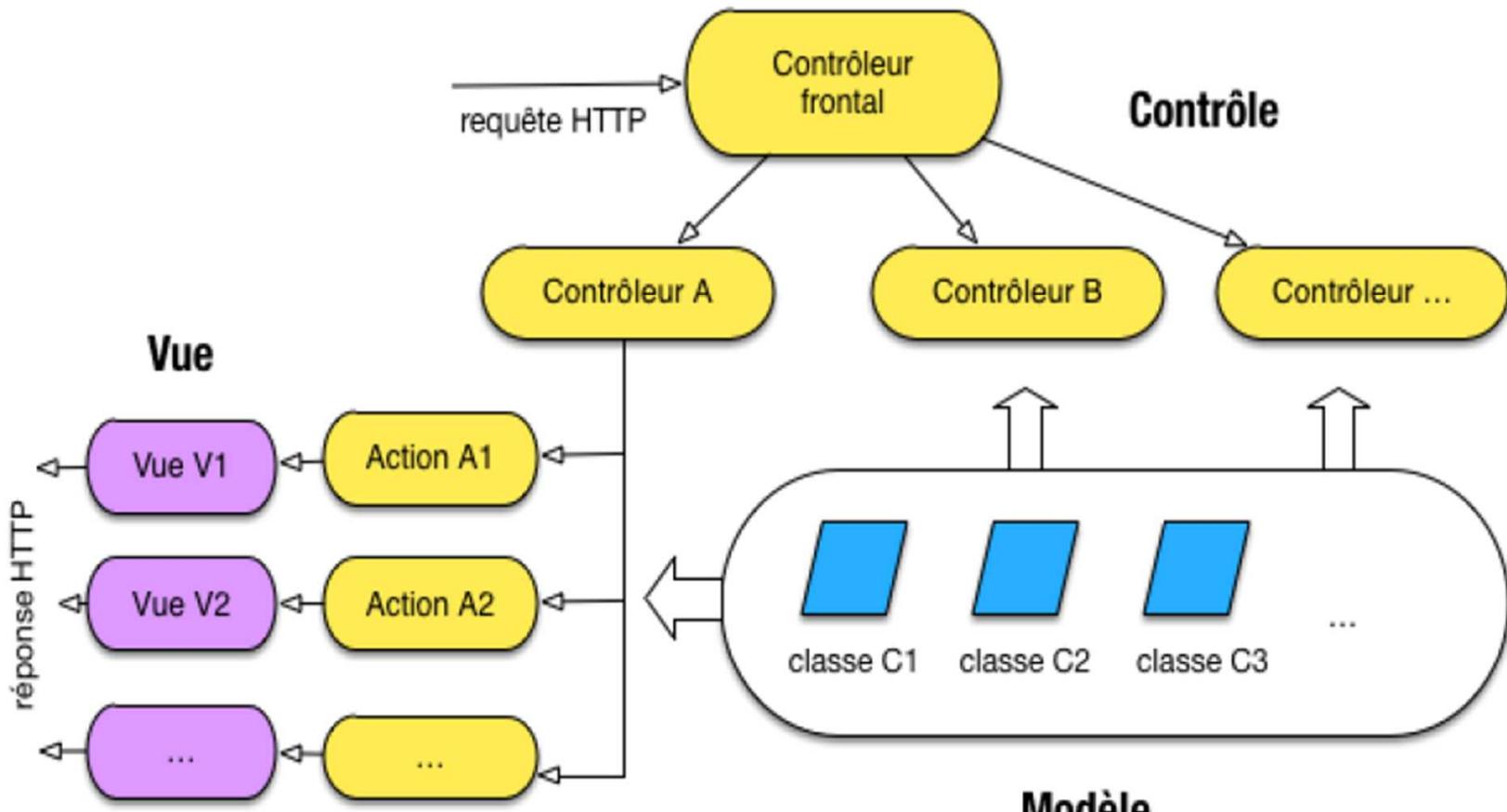
- MVC est un motif de conception (design pattern)
- Il mène à une organisation logique et rigoureuse du code
 - Séparation/indépendance des couches
 - Définition des règles permettant de savoir où ajouter une fonctionnalité
 - Simplification de la maintenance et de l'évolutivité de chacune des couches
 - Clarté indispensable pour les gros projets
- Il est très répandu et résulte de la formalisation des bonnes pratiques



Architecture MVC : principe général

- Le MVC sépare:
 - Les données (le modèle)
 - La présentation (la vue)
 - Les traitements (actions, coordonnées par des contrôleurs)

Architecture MVC



(cas framework web)



Architecture MVC : le modèle

- Le modèle:
 - Implante les fonctionnalités de l'application, indépendamment des aspects interactifs
 - préservation de l'état d'une application entre deux requêtes HTTP, ainsi que des fonctionnalités qui s'appliquent à cet état.
 - Toute donnée persistante doit être gérée par la couche modèle
 - des objets métiers non persistant implantant une fonctionnalité particulière (un calcul, un service) sont également dans cette couche



Architecture MVC : le modèle

- Le modèle (suite ...):
 - Le modèle gère les données de session (le panier dans un site de commerce électronique par exemple) ou les informations contenues dans la base de données (le catalogue des produits en vente, pour rester dans le même exemple).
 - Cela comprend également les règles, contraintes et traitements qui s'appliquent à ces données, souvent désignées collectivement par l'expression **logique métier de l'application**



Architecture MVC : le modèle

- Le modèle (suite ...):
 - Les composants de la couche modèle doivent pouvoir être utilisés dans des contextes applicatifs différents: Ils doivent donc impérativement être **totallement indépendants** de la gestion des interactions avec les utilisateurs, ou d'un environnement d'exécution particulier



Architecture MVC : la persistance

- La persistance:
 - est la propriété de l'état d'un objet à être préservé au-delà de la durée d'exécution de l'application en général, et d'une action en particulier
 - est obtenue le plus souvent par stockage dans une base de données
 - est un aspect d'un objet métier
 - elle ne doit pas impacter la logique applicative implantée par cet objet
 - on devrait pouvoir mettre à jour la persistance d'un objet indépendamment des services qu'il fournit



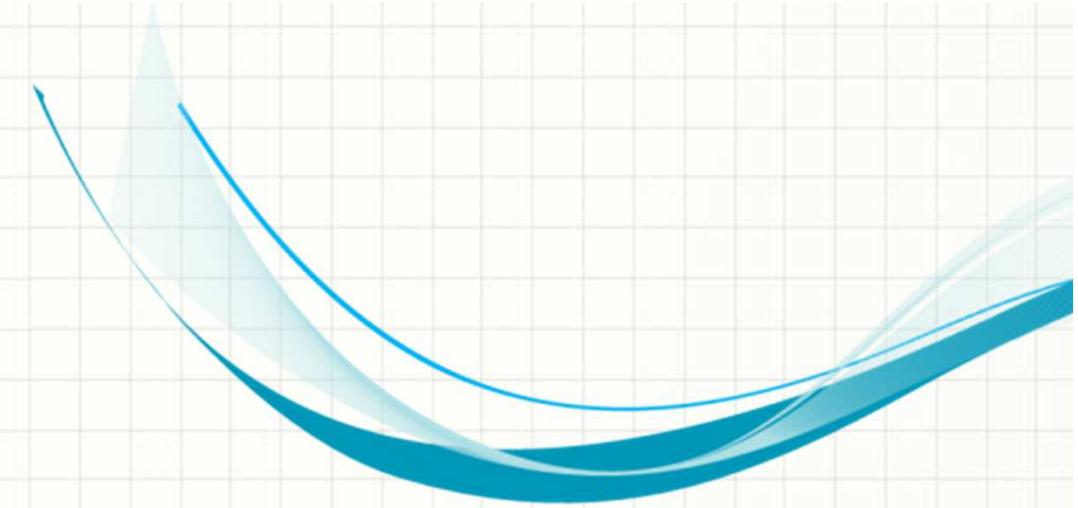
Architecture MVC : la vue

- La vue:
 - est responsable de l’interface utilisateur (GUI)
 - assure la mise en forme des données
 - ne doit pas intégrer les traitements complexes (“métiers”)
 - n’accède pas au modèle, obtient ses données de l’action
 - est souvent implantée par un moteur de templates



Architecture MVC : les contrôleurs

- Ils permettent de:
 - structurer hiérarchiquement l'application
 - faciliter la compréhension du code et la maintenance
- Comme pour la Vue, on évitera de placer de logique applicative dans un contrôleur, car le code **n'est pas réutilisable dans un autre contexte**



Partie 2: Le modèle : cas de la gestion de la persistance Java

Cours 2 : JDBC

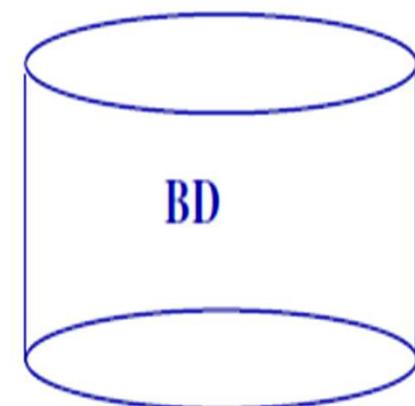
- Principe de JDBC
- Architecture de JDBC
- Les pilotes (driver) et types de pilotes
- Structure d'un programme JDBC
- Traitement des données renvoyées
- Activité pratique 1 : 1^{er} exemple utilisation de JDBC
- Correspondance type de données SQL/Java
- Activité pratique 2 : 2^{ème} exemple d'utilisation de JDBC
- L'interface PreparedStatement

Introduction

- JDBC : Java DataBase Connectivity



?





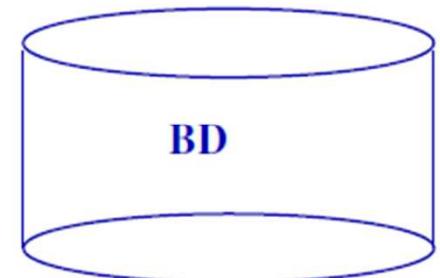
Introduction

- JDBC permet d'exécuter, depuis un programme Java, des ordres SQL reconnus par la BDD cible
- La BDD doit reconnaître le standard ANSI SQL 2
- JDBC va permettre:
 - De se connecter à la BDD
 - D'envoyer des requêtes SQL
 - De récupérer le résultat des requêtes

Introduction



?

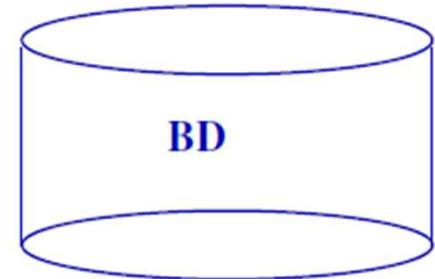


- C'est une API pour connecter des programmes Java à des BDD
- C'est un ensemble de classes et interfaces Java
- Package `java.sql.*`

Gestion du pilote



?



- Le pilote est un composant logiciel qui satisfait aux spécifications JDBC établies par SUN.
- Il va établir le lien avec la BDD
- Il existe des pilotes pour les principaux moteur de BDD (MySQL, PostGRE, Oracle, ..)



Gestion du pilote

- Le pilote est une classe Java qui implémente l'interface `java.sql.Driver`

Chargement du pilote

- Le pilote étant une classe Java, il faut charger cette classe.
- Pour cela la méthode Class.forName() est généralement utilisée
- Pour le driver MySQL, cette classe est com.mysql.jdbc.Driver

Chargement du pilote

- Exemple de chargement de pilote

```
import java.sql.*;  
public class QueryExample {  
    ...  
    try {  
        Class.forName("com.mysql.jdbc.Driver");  
    }  
    catch (ClassNotFoundException e) {  
        System.err.println("Driver loading error : " + e);  
    }  
    ...  
}
```

Connection à la BDD

- Méthode getConnection() de DriverManager

```
import java.sql.DriverManager;

public class JDBCUpdateExample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("----- PostgreSQL "
            + "JDBC Connection Testing -----");

        try {

            Class.forName("org.postgresql.Driver");

        } catch (ClassNotFoundException e) {

            System.out.println("Where is your PostgreSQL JDBC Driver? "
                + "Include in your library path!");
            e.printStackTrace();
            return;
        }

        System.out.println("PostgreSQL JDBC Driver Registered!");

        Connection connection = null;
        JFrame mainFrame = new JFrame(" JDBC Update Example");

        try {

            connection = DriverManager.getConnection(
                "jdbc:postgresql://localhost:5432/postgres", "postgres",
                "*****");

            Statement orderSELECT = connection.createStatement();

            //Ordre SQL pour INSERT
            int nbLignes = orderSELECT.executeUpdate("INSERT INTO bp_articles (bpart_code, bpart_description, bpart_qte, bpar
TOptionsBans.showMessageDialog(mainFrame, " NB de lignes insérées : " + nbLignes);
        }
    }
}
```

Connection à la BDD : Datasource

```
import java.sql.ResultSet;

import org.postgresql.ds.PGPoolingDataSource;

public class JDBCExample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("----- PostgreSQL JDBC Connection Testing -----");

        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException e) {

            System.out.println("Where is your PostgreSQL JDBC Driver? "
                + "Include in your library path!");
            e.printStackTrace();
            return;
        }

        System.out.println("PostgreSQL JDBC Driver Registered!");
        Connection connection = null;

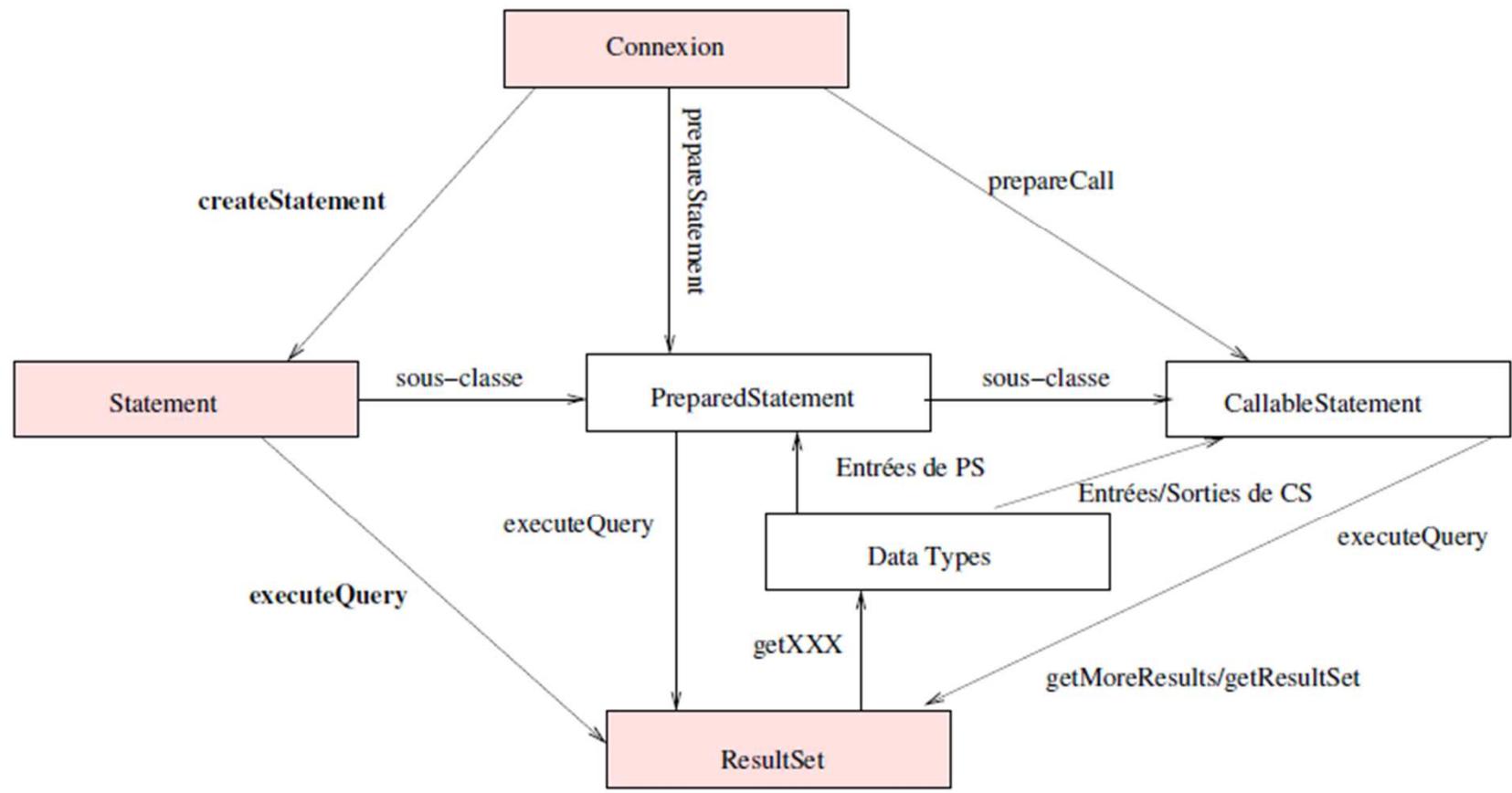
        try {
            PGPoolingDataSource source = new PGPoolingDataSource();
            source.setDataSourceName("Ma source de données Postgresql");
            source.setServerName("localhost");
            source.setDatabaseName("postgres");
            source.setPortNumber(5432);
            source.setUser("postgres");
            source.setPassword("*****");
            source.setMaxConnections(10);

            connection = source.getConnection();

            Statement orderSELECT = connection.createStatement();
            //
        }
    }
}
```

Package java.sql

- Classes et interfaces du package java.sql



Les URLs JDBC

- Afin de localiser la base de donnée sur le serveur de base de donnée, il est indispensable de spécifier une adresse de connexion sous la forme d'une URL. Ces URL commenceront toutes par "jdbc:".
- Pour Oracle, l'url est de la forme
 - jdbc:oracle:<drivertype>:@<host>:<port>:<db>
- Host : @IP ou nom du serveur
- Port : port utilisé par la BDD
- Db : oracle SID de la base de donnée



Etapes d'un programme JDBC

- Les étapes d'un programme utilisant JDBC sont les suivantes:
 - Chargement du pilote (driver) JDBC
 - Établir une connexion à la BDD
 - Exécuter des requêtes SQL
 - Utiliser les résultats des requêtes pour les afficher, etc.
 - Terminer la connexion



Etapes d'un programme JDBC

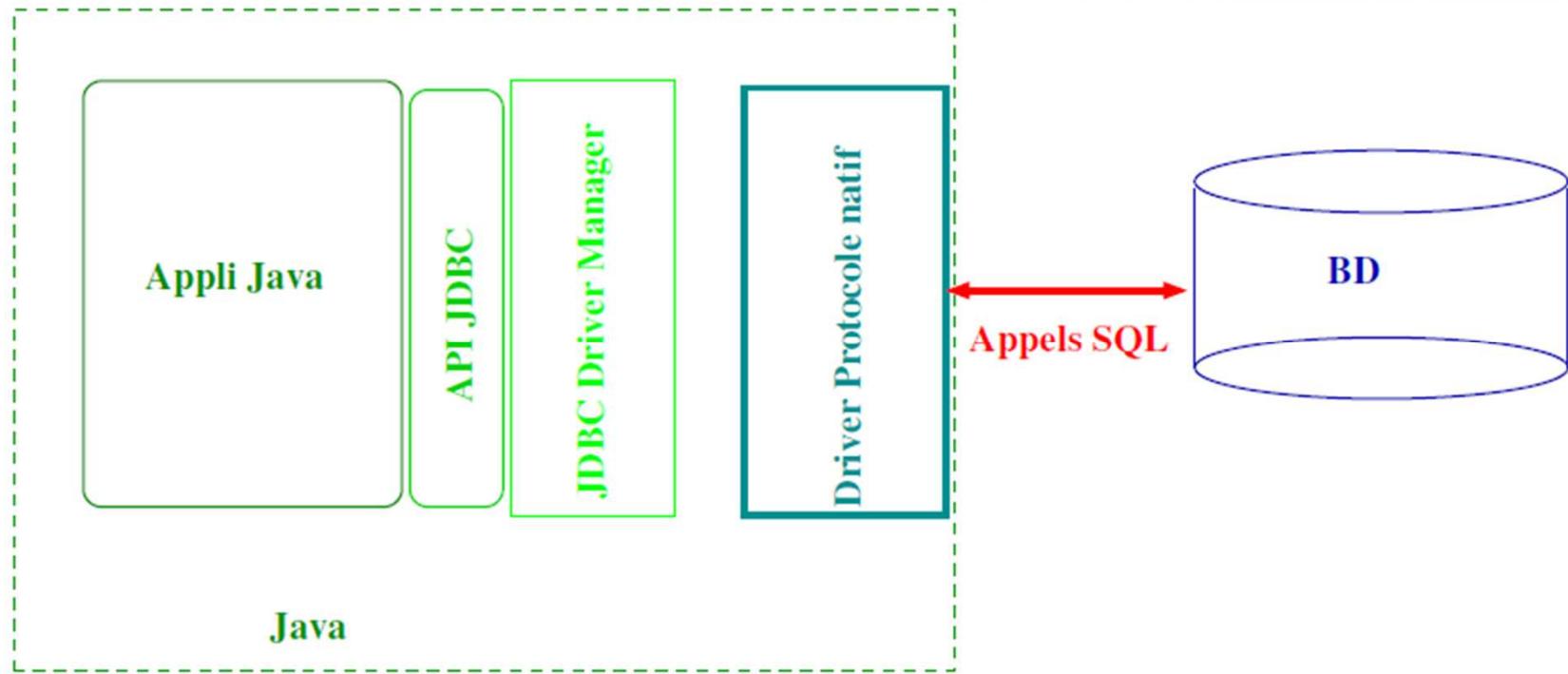
- exemples
 - Chargement d'un driver
 - Class.forName(« oracle.jdbc.driver.OracleDriver »)
 - Créer un objet connexion
 - Connection.maConnection = DriverManager.getConnection(url) où url est l'url décrivant la BDD
 - Créer un objet Statement
 - Statement stmt = maConnection.createStatement()
 - Exécuter la requête et récupérer les résultats
 - Terminer la connexion



Etapes d'un programme JDBC

- Exemples (suite...):
 - Exécuter la requête
 - String query = « SELECT * FROM table »
 - ResultSet resultSet = statement.executeQuery(query)
 - Terminer la connexion
 - resultSet.close()
 - statement.close()
 - maConnection.close()

Finalement



Statement

- Exemples:
 - Statement monStmt =
maConnection.createStatement()
- Suivant l'instruction SQL on utilisera

Instructions SQL	Méthode	Type retourné	Valeur retournée
SELECT	executeQuery	ResultSet	Lignes de résultat
UPDATE, INSERT,DELETE	executeUpdate	int	Nb lignes modifiées
Autres	execute	boolean	Faux si erreur

Traitement des résultats

- L'objet ResultSet permet d'avoir un accès aux données résultantes de notre requête en mode ligne par ligne. La méthode *ResultSet.next()* permet de passer d'une ligne à la suivante. Cette méthode renvoie false dans le cas où il n'y a pas de ligne suivante. Il est nécessaire d'appeler au moins une fois cette méthode, le curseur est placé au départ avant la première ligne (si elle existe).

Traitement des résultats

- La classe *ResultSet* dispose aussi d'un certain nombres d'accesseurs (*ResultSet.getXXX()*) qui permettent de récupérer le résultat contenu dans une colonne sélectionnée.
- On peut utiliser soit le numéro de la colonne désirée, soit son nom avec l'accesseur. La numérotation des colonnes commence à 1.
- XXX correspond au type de la colonne. Le tableau suivant précise les relations entre type SQL, type JDBC et méthode à appeler sur l'objet *ResultSet*.

Les correspondances de types

Type SQL	Type Java
CHAR, VARCHAR2,	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte []
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Exemple

```
try {  
    while (rs.next()) {  
        System.out.println(rs.getString("NAME") +  
            " - " + rs.getString("EMAIL"));  
    }  
}  
  
catch (SQLException e) {  
    System.err.println("Error browsing query  
        results: " + e.getMessage());  
}
```



Les valeurs NULL

- Problème : reconnaître dans Java le cas d'une valeur NULL.
- Conventions :
 - Pour les méthodes `getString()`, `getObject()`, `getDate()`, ... : Null Java (il existe)
 - Pour les méthodes `getInt()`, `getByte()`, `getShort()`, ... : la valeur 0 est renvoyée
 - Pour la méthode `getBoolean()`, la valeur Faux est renvoyée.



Les valeurs NULL

- Utiliser la méthode `wasNull()` de `ResultSet`.
- Fonctionnement :
 - lire la donnée,
 - tester avec `wasNull()` si elle vaut `NULL` au sens SQL



Instruction SQL paramétrée

- JDBC permet de profiter de ce type de fonctionnalité par l'utilisation de requêtes paramétrées ou de procédures stockées.
- Les requêtes paramétrées sont associées aux instances de l'interface *PreparedStatement* qui hérite de l'interface *Statement*

Création de requête paramétrée

- PreparedStatement pstmt = conn.prepareStatement("UPDATE emp SET sal=? WHERE nom=?;");
- Les '?' indiquent les emplacements des paramètres.
- Les valeurs des paramètres sont données par les méthodes setXXX(n,valeur).
- On choisit la méthode setXXX suivant le type SQL de la valeur que l'on veut mettre dans la requête.

Requête paramétrée : exemple

```
PreparedStatement pstmt =  
    conn.prepareStatement("UPDATE emp SET sal=?  
    WHERE nom=?;");  
  
for(int i=0; i<=10,i++)  
{  
    pstmt.setDouble(1, salaire[i]);  
    pstmt.setString(2,nom[i]);  
    pstmt.executeUpdate();  
}
```



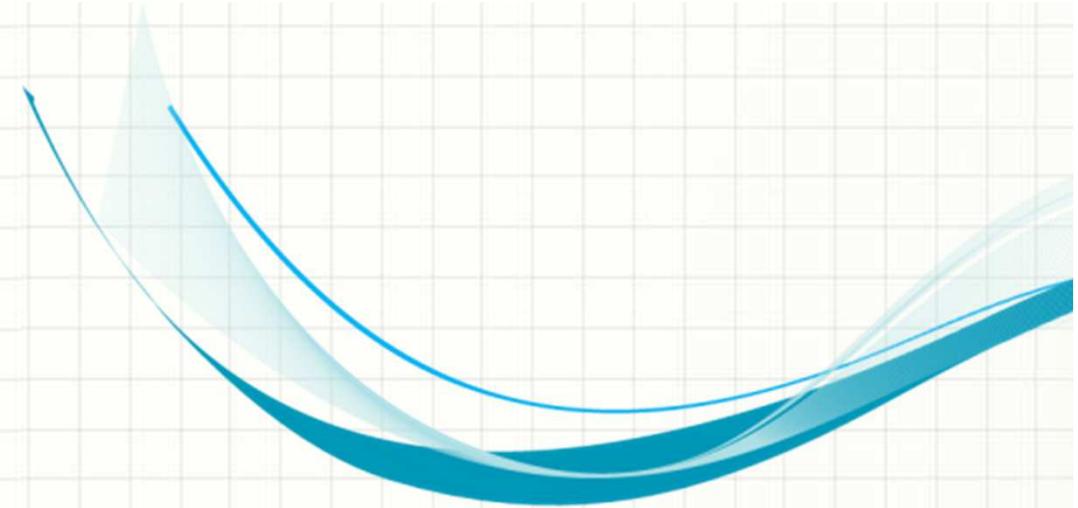
Requête paramétrée : avantages

- Traitement plus rapide si elles sont utilisées plusieurs fois avec plusieurs paramètres.
- Amélioration de la portabilité (indépendance des setXXX avec les SGBD).



Déconnexion

- Objectif: libérer les ressources
- Il faut donc libérer ResultSet et Statement puis fermer la Connection



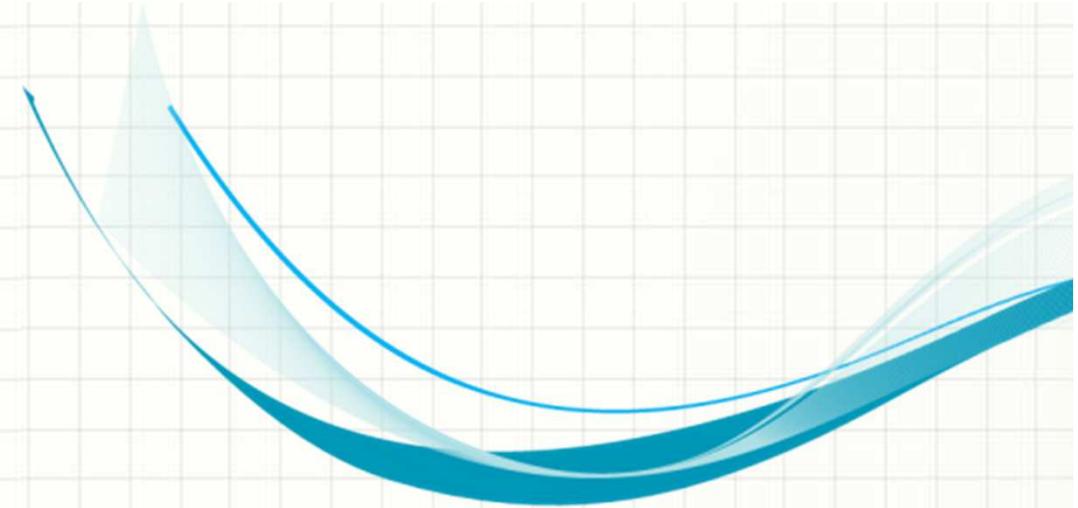
Partie 3 : les view

– cas des interfaces graphiques Java



Cours 3 : Les SWING GUI de Java

- Présentation des SWING
- Création d'une fenêtre SWING
- Ajout des composants légers
- Le gestionnaire JFrame : BorderLayout
- Gestionnaire de mise en forme
- Tour d'horizon de GridBagLayout
- Objet : Panel
- Quelques contrôles de menus
- Gestion des boîtes de dialogue



SWING :

Présentation

Présentation SWING: historique

- Au début était AWT (Abstract Windows Toolkit) dont les composants étaient basés sur l'OS
- Pour chaque composant il y avait une couche d'adaptation entre Java et l'OS
- Pb de compatibilités sont apparus
- Refonte de la bibliothèque, elle est renommée SWING
- Dans SWING les composants sont des éléments de la JVM (composants légers)



SWING: concepts de base

- Swing partage de nombreux concepts avec les library de création d'interfaces graphiques tout en les adaptant aux spécificités de Java.
La construction d'interfaces graphiques est basée sur quatre éléments principaux:
 - Les composants
 - Les conteneurs
 - Les gestionnaires de layout
 - Les gestionnaires d'évènements



SWING: concepts de base

- Les composants
 - Les éléments constitutants les interfaces graphiques (fenêtres, textes, boutons, images, etc.) sont appelés les composants
 - Dans SWING tous les composants descendent de la même classe
- Les conteneurs
 - Les conteneurs peuvent contenir des composants.
 - SWING propose une variété de conteneurs



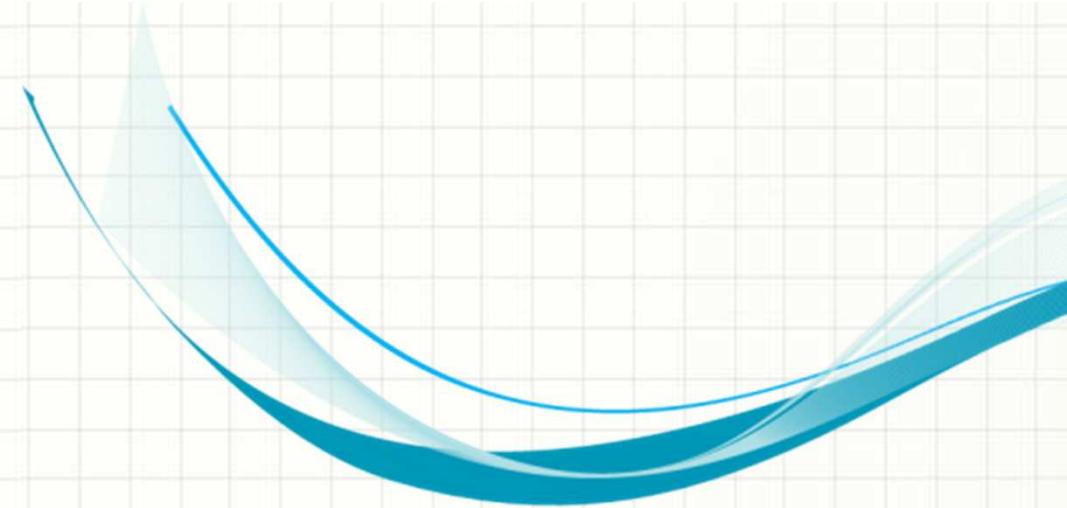
SWING: concepts de base

- Les gestionnaires de positionnement
 - Le positionnement des composants est confié à un gestionnaire de positionnement
 - SWING propose différents gestionnaire de positionnement
- Les gestionnaires d'évènements
 - Les actions de l'utilisateur sont représentées par des évènements
 - SWING propose une méthode souple de gestion des évènements



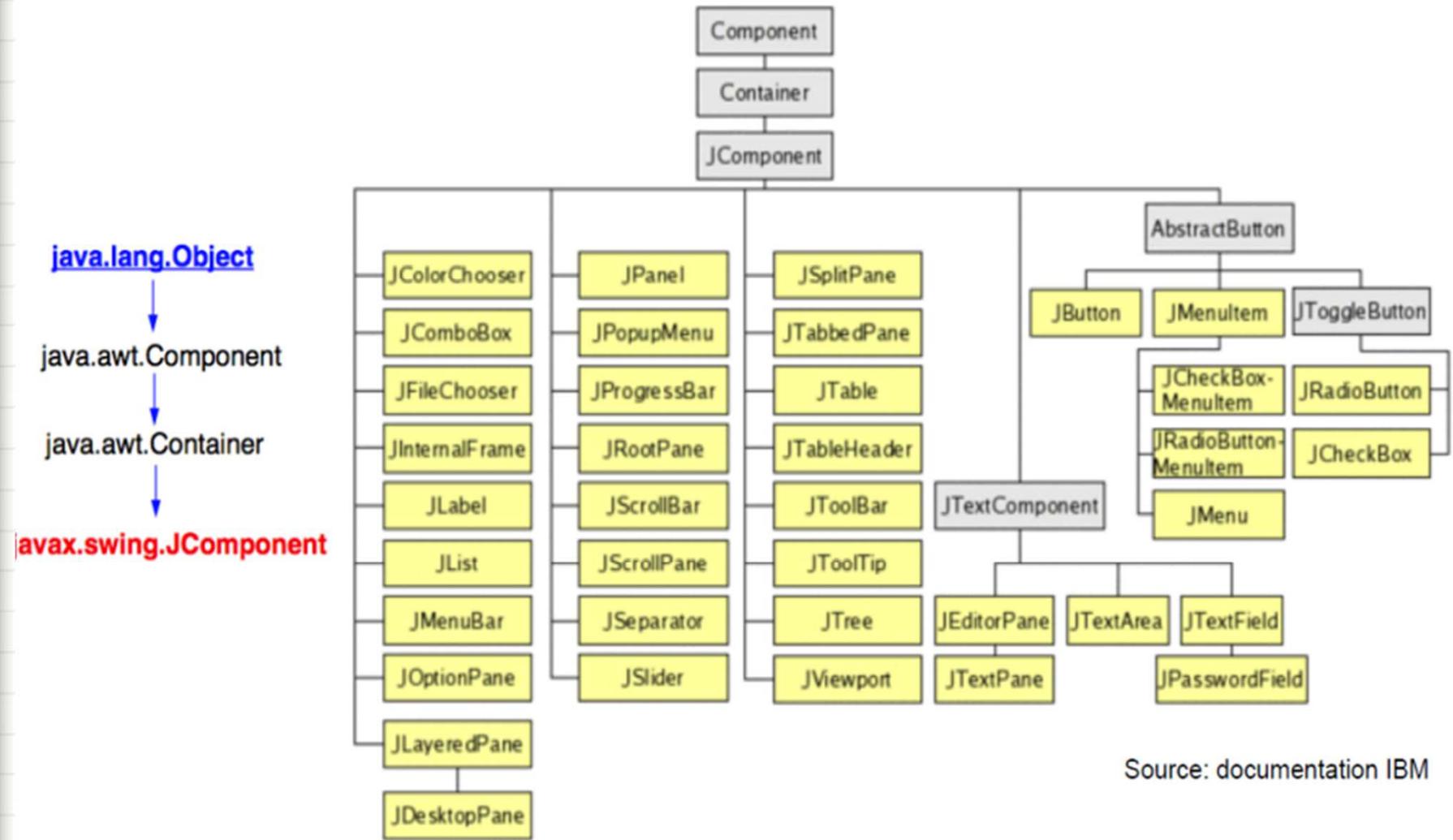
SWING: concepts de base

- Les gestionnaires d'évènements
 - Les évènements sont des objets qui sont transmis par un composants vers un gestionnaire d'évènements.
 - Ce gestionnaire modifie ensuite le programme pour prendre en compte les besoin de l'utilisateur.
 - La gestion des évènements est séparée de la création du UI ce qui facilite les modifications éventuelles de programme



SWING : les principaux composants

SWING: les composants





SWING: Component, JComponent

- Tous les composants de Swing descendent de la classe JComponent, qui descend elle même de la classe Component de AWT.
- Cette hiérarchie permet de proposer de nombreuses méthodes communes a tous les éléments.
- Relations entre les composants
 - Tous les composants sont hébergés par un conteneur.

SWING: Component, JComponent

- Les propriétés géométriques
 - Swing propose les gestionnaires de placement pour disposer les composants automatiquement tout en assurant une compatibilité graphique entre les différents systèmes d'exploitation.
 - La taille d'un composant peut être fixée avec la méthode setSize (et lue avec la méthode getSize).
- Les infobulles :
 - Tous les composants dérivant de JComponent peuvent être agrémentées d'info-bulles.

SWING: Component, JComponent

- Les infobulles
 - C'est la méthode `setToolTipText` qui instancie un composant `JToolTip` et l'associe au composant
- Les bordures
 - Les composants peuvent être encadrées par une bordure. Plusieurs bordures sont disponibles en creux, en relief, avec un texte,...
 - La classe `BorderFactory` fournit de nombreuses méthodes statiques pour créer des bordures prédéfinies.
 - *monComposant.setBorder(BorderFactory.createTitleBorder("Une bordure"));*



SWING: Conteneurs primaires

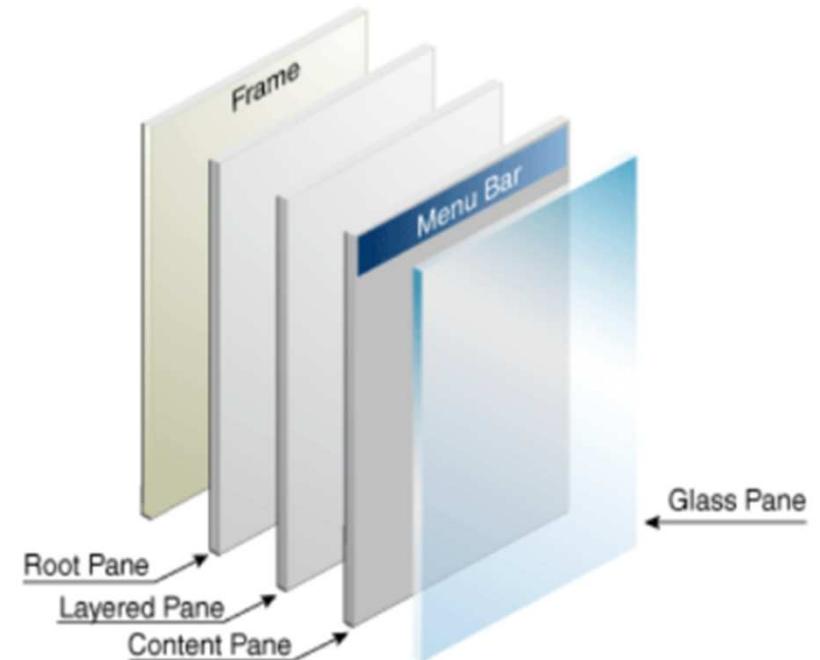
- Les conteneurs primaires sont les seuls composants qui sont dessinées par le système d'exploitation, ce sont donc des composants lourds.
- Toute application graphique doit utiliser un conteneur primaire.
- Il en existe 3 types:
 - Les applets (JApplets)
 - Les fenêtres (JFrame et JWindow)
 - Les boites de dialogues (JDialog)

SWING: Conteneurs primaires

- Propriétés communes

JRootPane implicitement créé

- par JApplet, JDialog, JFrame
- et JInternalFrame



- Les composants sont placés sur le ContentPane
- On y accède par getContentPane()



SWING: Conteneurs primaires

- Propriétés communes
 - On y accède par `getContentPane()` puis on appelle la méthode `add()` pour y ajouter des composants
 - Une autre approche consiste à utiliser un nouveau conteneur que l'on place ensuite dans le conteneur primaire. On utilise la méthode `setContentPane()` pour spécifier un nouveau conteneur

SWING: Jwindow, JFrame

- Swing propose deux types de fenêtres : les JWindow et les JFrame.
- Méthodes communes
 - setVisible()
 - Pack()
- Différences entre Jwindow et Jframe
 - La classe JWindow permet de créer une fenêtre graphique dans le système de fenêtrage utilisé. Cette fenêtre n'a aucune bordure et aucun bouton.



SWING: JWindow, JFrame

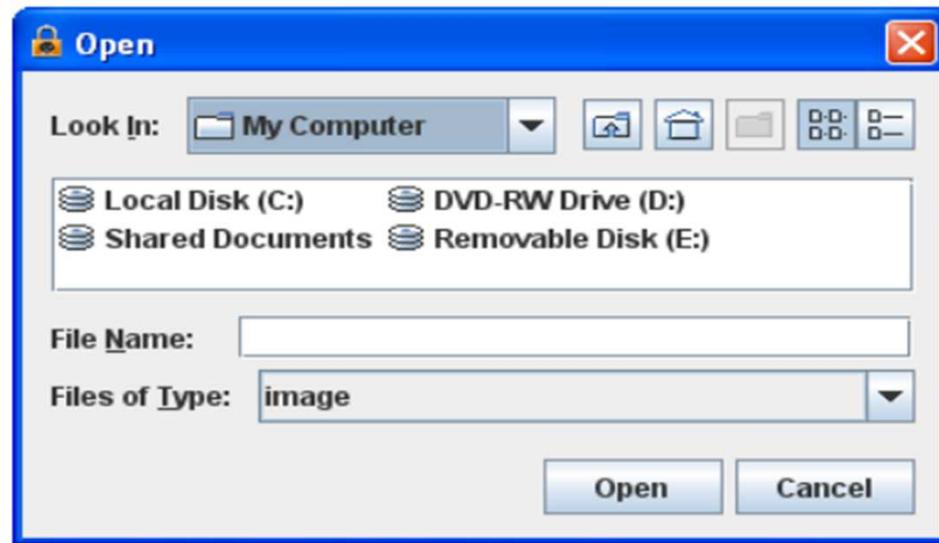
- Différences entre JWindow et JFrame
 - D'une manière générale, JWindow n'est utilisée que pour créer des fenêtres particulières comme les splash screens ou des écrans d'attente.
 - La plupart des applications sont construites à partir d'une (ou plusieurs) JFrame.
 - JFrame construit des fenêtres qui comportent une bordure, un titre, des icônes et éventuellement un menu.



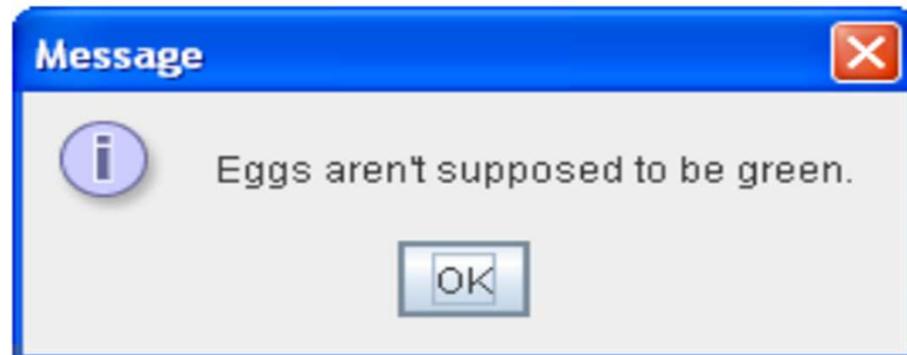
SWING: Boites de dialogue

- Swing propose des boites de dialogues afin de communiquer rapidement avec l'utilisateur.
- La classe JOptionPane permet de créer des boites de dialogues
- Il y a différents types de boites de dialogue
 - Les dialogues de message (showMessageDialog)
 - Les dialogues de confirmation/question (showConfirmDialog)
 - Les dialogues de saisie (showInputDialog)

SWING: Boites de dialogues

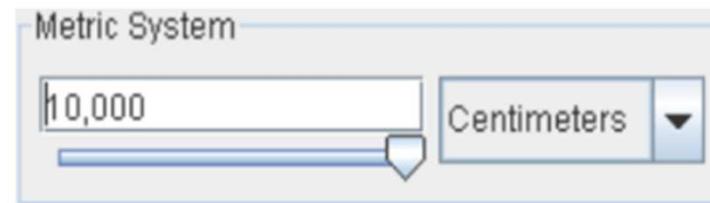
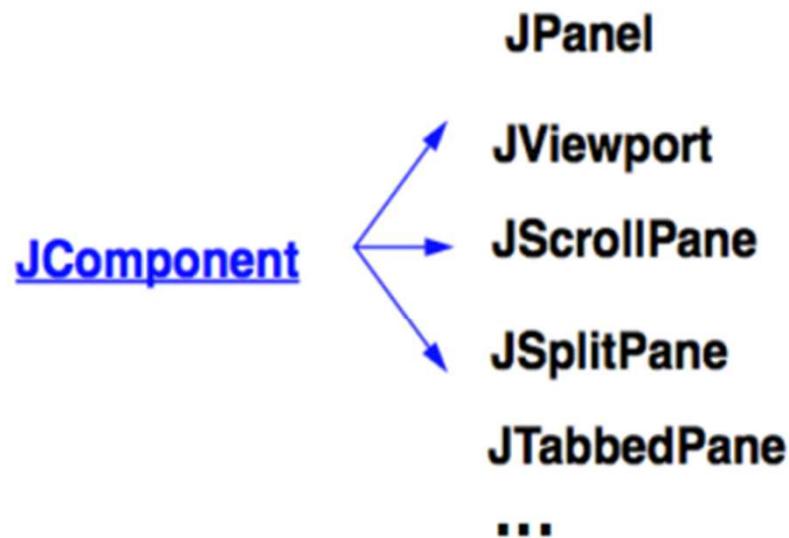


JFileChooser



JOptionPane (multiples variantes)

SWING: conteneurs secondaires



JPanel: conteneur générique

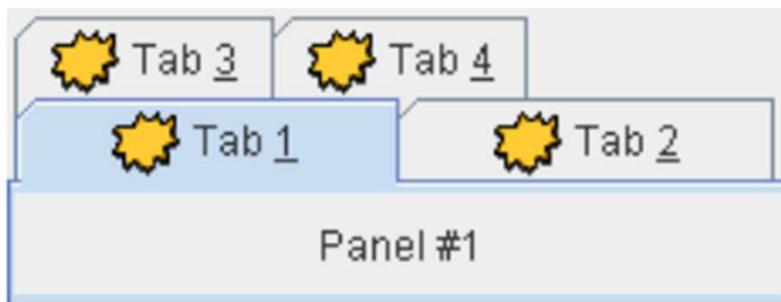


JScrollPane:
avec ascenseurs intégrés



JSplitPane:
avec « diviseur » intégré

SWING: conteneurs secondaires



JTabbedPane:
onglets

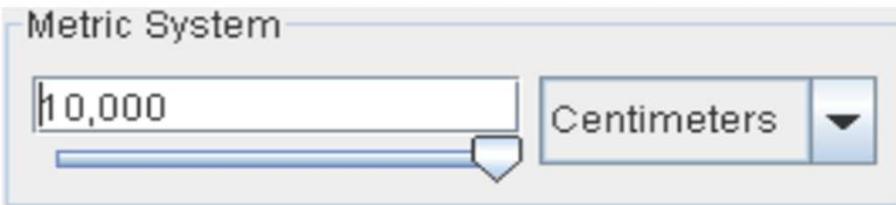
- Swing propose de nombreux conteneurs secondaires pour créer des interfaces ergonomiques. Ce sont des composants légers.



SWING: conteneurs secondaires

- Les principaux conteneurs secondaires sont:
 - Le panneau : JPanel
 - Le panneau à défilement : JScrollPane
 - Le panneau à onglets : JTabbedPane
 - La barre d'outil : JToolBar
 - Les bureaux : JDesktopPane
 - ...

Le panneau: JPanel



JPanel: conteneur générique

- C'est le conteneur léger le plus simple de SWING
- C'est le conteneur par défaut de JFrame et Jwindow
- Il permet de regrouper les composants selon un politique de placement (layout).
- Pour ajouter un composant on utilise la méthode *add*

Le panneau à défilt: JScrollPane



JScrollPane:
avec ascenseurs intégrés

- Des ascenseurs sont placés sur les côtés pour permettre de se déplacer dans le document
- Ne peut héberger qu'un seul composant
- S'il doit accueillir plusieurs composants on les place dans un JPanel que l'on place dans le JScrollPane

Le panneau divisé: JSplitPane



JSplitPane:
avec « diviseur » intégré

- Permet de séparer une interface en 2 volets horizontaux ou verticaux
- Ne peut accueillir que 2 composants

Le panneau à onglets: JTabbedPane



JTabbedPane:
onglets

- Permet de regrouper les composants de manière thématique pour alléger l'interface
- La méthode `addTab()` permet de rajouter un composant dans une nouvelle feuille
- Généralement on utilise un conteneur de type JPanel

Le panneau à onglets: JTabbedPane

- Les onglets sont numérotés à partir de 0
- L'onglet affiché peut être modifié à partir de la méthode `setSelectedIndex()`
- La position des onglets peut être modifiées en utilisant `setTabPlacement()`
- Les onglets peuvent être positionnés en haut, en base, à gauche, à droite
- Tous les composants présents sont créés même s'ils ne sont pas visible => il ne faut pas surcharger les onglets

La barre d'outil: JToolBar



JToolBar: barre d'outils
(sous la barre de menus)

- Permet de construire des barres d'outils regroupant n'importe quel composant
- On peut ajouter des composants à l'aide de la méthode add()
- Il est possible de regrouper les composants entre eux en matérialisant un espace entre deux groupes à l'aide de la méthode addSeparator()



Le bureau: JDesktopPane

- Permet l'ouverture de plusieurs documents simultanément. Ces interfaces sont nommées MDI
- Le composant JDesktopPane propose une implémentation de ce comportement. Il permet d'afficher des fenêtres (JInternalFrame) à l'intérieur de l'application.



SWING: les composants atomiques

- Les composants atomiques sont les composants élémentaires de SWING:
 - Les labels : JLabel
 - Les boutons : JButton et JToggleButton
 - Les cases à cocher : JCheckBox
 - Les boutons radio : JRadioButton
 - Les listes de choix : Jlist
 - Les boîtes combo : JComboBox
 - Les menus : JMenu
 - Les glissières : JSlider

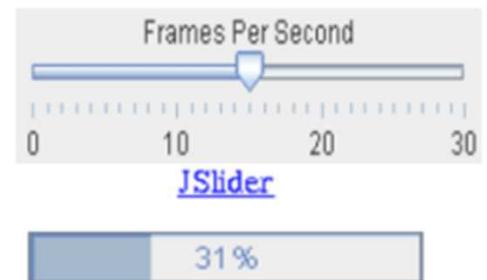
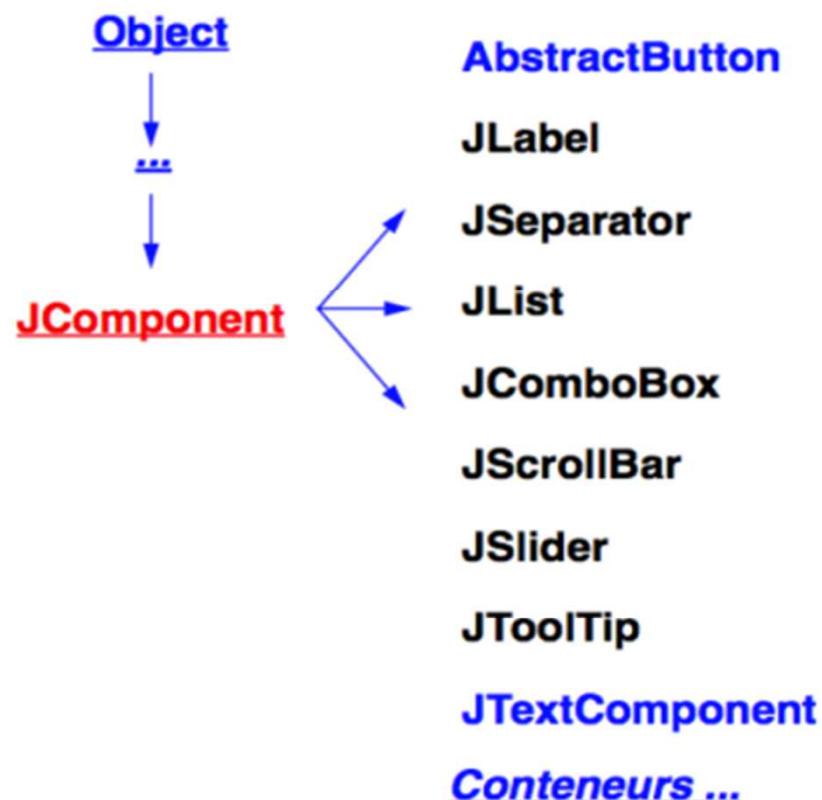


SWING: les composants atomiques

- Les composants atomiques sont les composants élémentaires de SWING:
 - Les boites de dialogue de sélection de fichier : JFileChooser
 - Les composants orientés text : JTextField, JTextArea, JEditorPane
 - Etc.

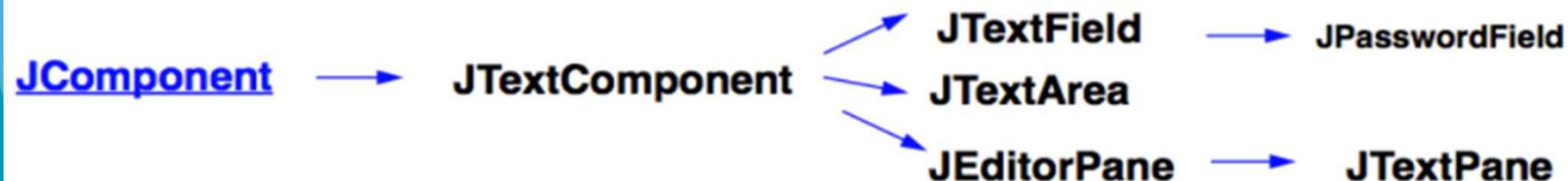
SWING: les composants atomiques

- Les interacteurs



SWING: les composants atomiques

- Texte



City: Santa Rosa

JTextField

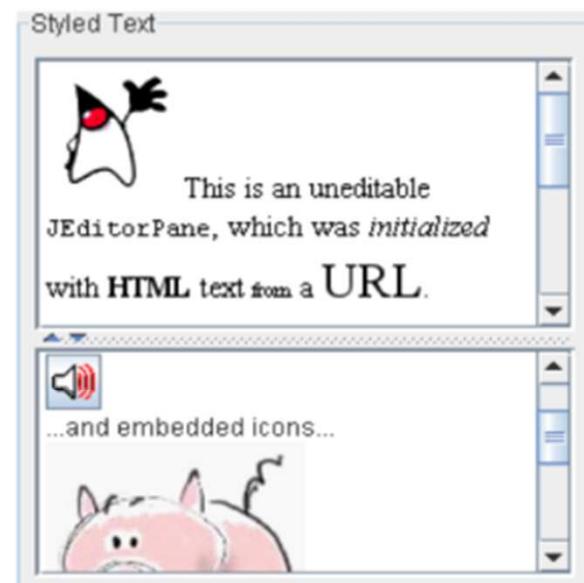
Enter the password:

JPasswordField

This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

JTextArea :
texte simple multilignes

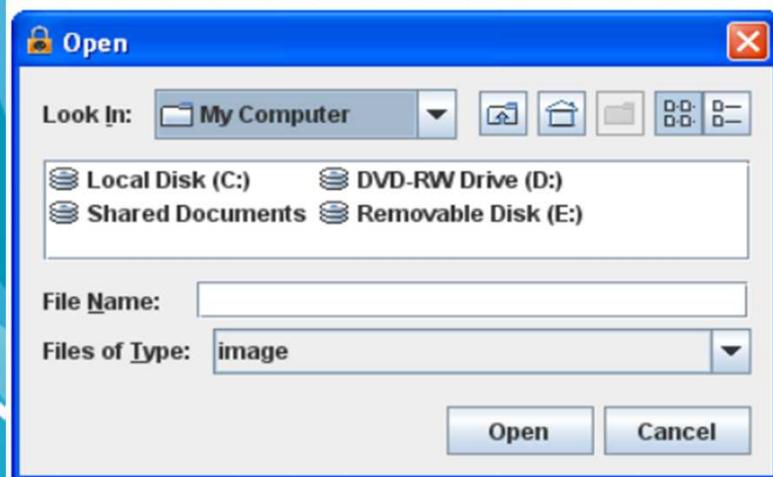
Ascenseur :
cf. **JScrollPane**



JEditorPane : texte avec styles compatible HTML et RTF

SWING: les composants atomiques

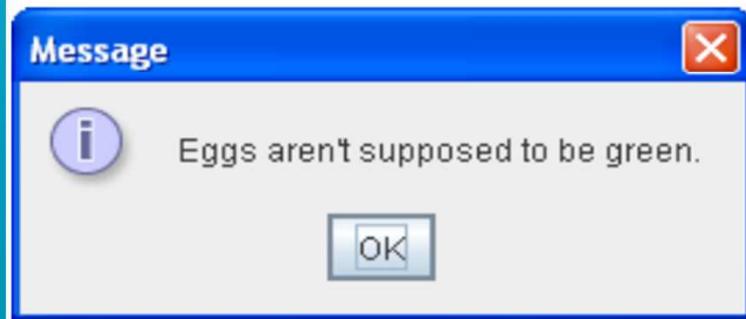
- Boîte de dialogue prédéfinie



JFileChooser



JColorChooser



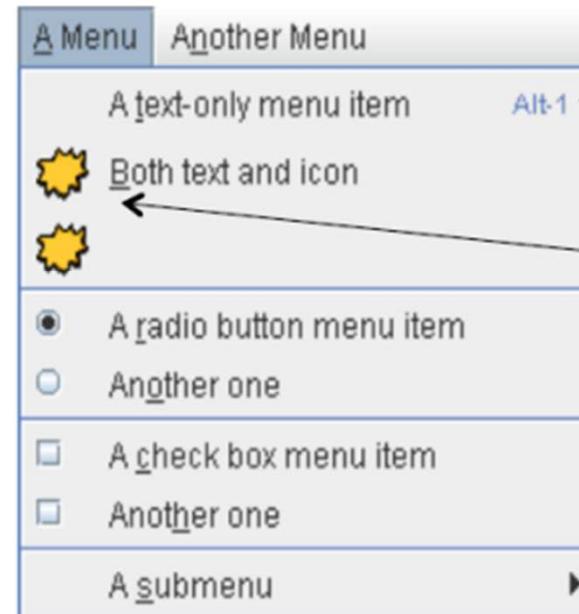
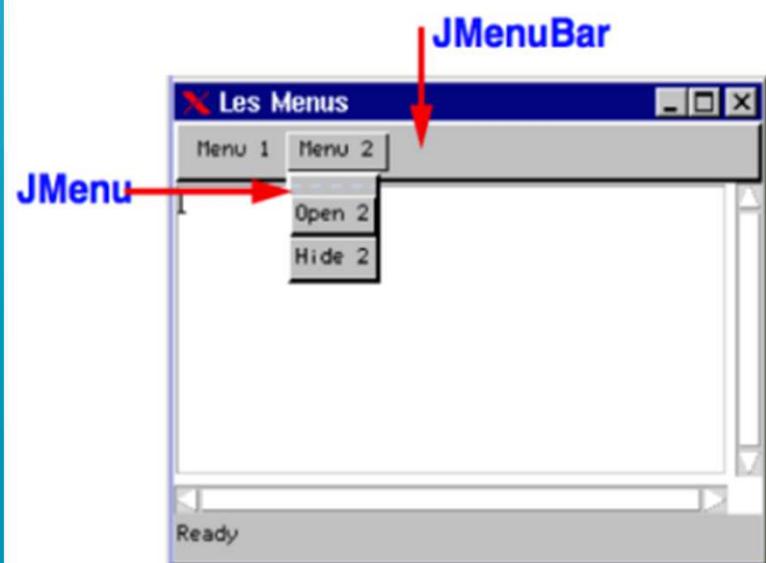
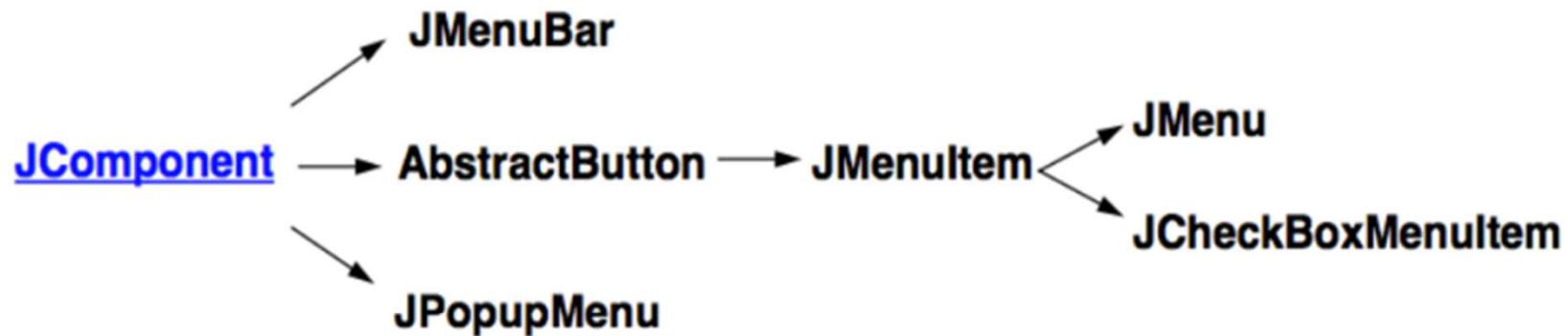
JOptionPane (multiples variantes)

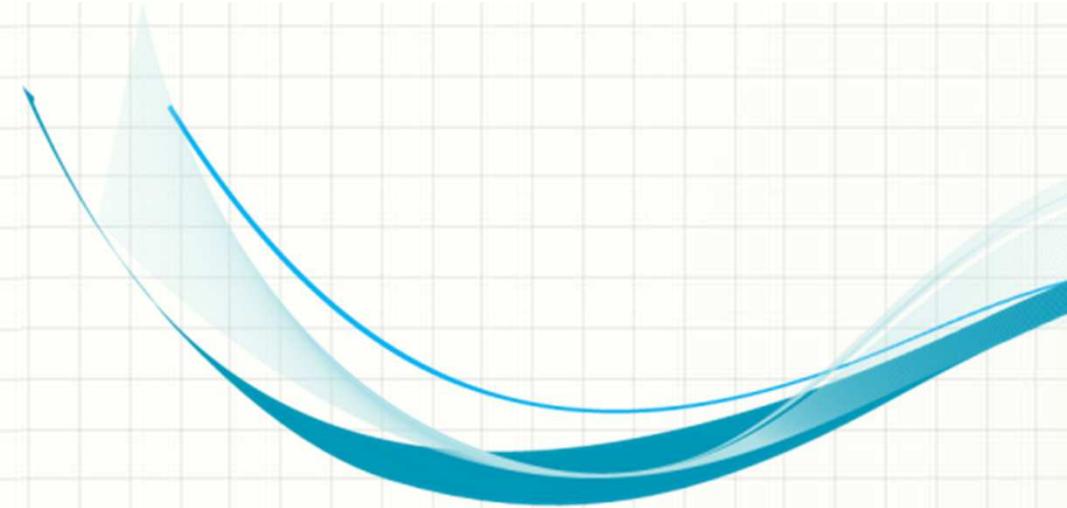
Particularité

- peuvent être créés :
 - comme composants internes
 - ou comme boîtes de dialogue

SWING: les composants atomiques

- Les menus





SWING : les gestionnaires de présentations



SWING: Gestionnaires de présentation

- A chaque conteneur est associé un gestionnaire de présentation (*layout manager*)
- Le gestionnaire de présentation gère le positionnement et le (re)dimensionnement des composants d'un conteneur.
- Le ré-agencement des composants dans un conteneur a lieu lors de :
 - ◆ la modification de sa taille,
 - ◆ le changement de la taille ou le déplacement d'un des composants.
 - ◆ l'ajout, l'affichage, la suppression ou le masquage d'un composant.



SWING: Gestionnaires de présentation

- Les principaux gestionnaires de présentation de SWING sont : **FlowLayout, BorderLayout, GridLayout, CardLayout, GridBagLayout**
- Tout conteneur possède un gestionnaire de présentation par défaut.
 - ◆ Tout instance de **Container** référence une instance de **LayoutManager**.
 - ◆ Il est possible d'en changer grâce à la méthode **setLayout()**.



SWING: Gestionnaires de présentation

- Les gestionnaires de présentation par défaut sont :
 - ◆ Le **BorderLayout** pour **JWindow** et ses descendants (**JFrame**, **JDialog**, ...)
 - ◆ Le **FlowLayout** pour **JPanel** et ses descendants (**JApplet**, etc.)
- Une fois installé, un gestionnaire de présentation fonctionne "tout seul" en interagissant avec le conteneur.



SWING: le FlowLayout

- Le FlowLayout est le plus simple des managers de SWING
- Gestionnaire de présentation utilisé par défaut dans les JPanel si aucun LayoutManager n'est spécifié.
- Un FlowLayout peut spécifier :
 - ◆ une justification à gauche, à droite ou centrée,
 - ◆ un espacement horizontal ou vertical entre deux composants.
 - ◆ Par défaut, les composants sont centrés à l'intérieur de la zone qui leur est allouée.

SWING: le FlowLayout

- La stratégie de disposition du FlowLayout est la suivante :
 - ◆ Respecter la **taille préférée** de tous les composants contenus.
 - ◆ Disposer autant de composants que l'on peut en faire tenir horizontalement à l'intérieur de l'objet **Container**.
 - ◆ Commencer une nouvelle rangée de composants si on ne peut pas les faire tenir sur une seule rangée.
 - ◆ Si tous les composants ne peuvent pas tenir dans l'objet **Container**, ce n'est pas géré (c'est-à-dire que les composants peuvent ne pas apparaître)



SWING: le FlowLayout

- Le FlowLayout cache réellement et effectivement les composants qui ne rentrent pas dans le cadre.
- Le FlowLayout n'a d'intérêt que quand il y a peu de composants.
- La présentation FlowLayout positionne les composants ligne par ligne.
 - ◆ Chaque fois qu'une ligne est remplie, une nouvelle ligne est commencée.

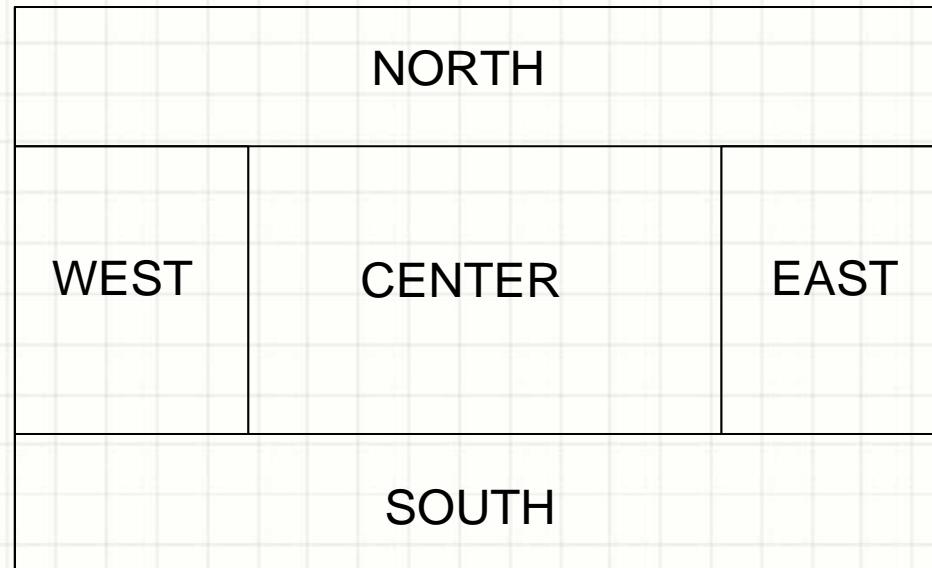
SWING: le BorderLayout

- BorderLayout divise son espace de travail en cinq zones géographiques : North, South, East, West et Center.
- Les composants sont ajoutés par nom à ces zones (un seul composant par zone).
 - ◆ Exemple

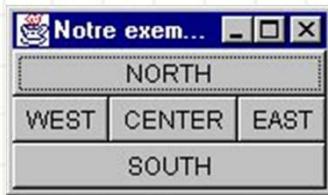
```
add("new JButton("Le bouton nord
```
 - ◆ Si une des zones de bordure ne contient rien, sa taille est 0.

SWING: le BorderLayout

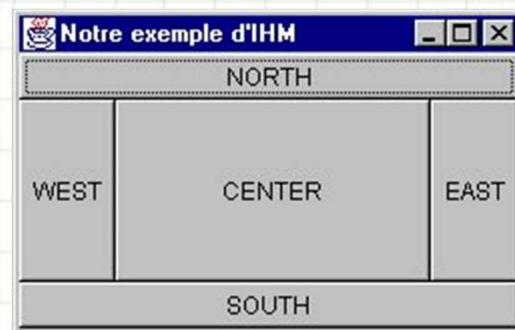
- Division de l'espace avec BorderLayout



SWING: le BorderLayout



Redimensionnement



Redimensionnement



SWING: le GridLayout

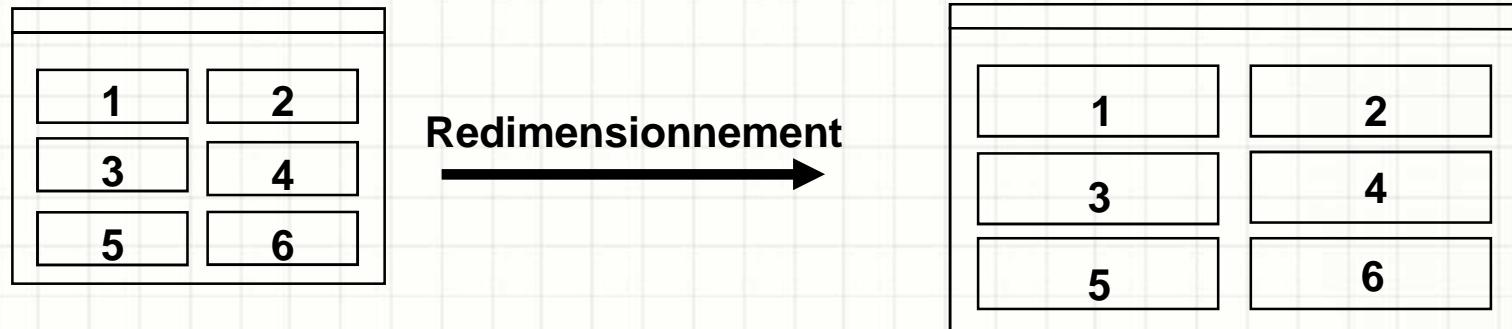
- Le GridLayout dispose les composants dans une grille.
 - ◆ Découpage de la zone d'affichage en lignes et en colonnes qui définissent des cellules de dimensions égales.
 - ◆ Chaque composant à la même taille
 - quand ils sont ajoutés dans les cellules le remplissage s'effectue de gauche à droite et de haut en bas.
 - ◆ Les 2 paramètres sont les rangées et les colonnes.
 - ◆ Construction d'un GridLayout : `new GridLayout(3,2);`

nombre de lignes

nombre de colonnes

SWING: le GridLayout

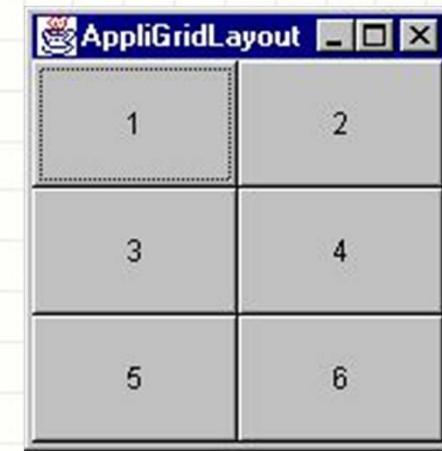
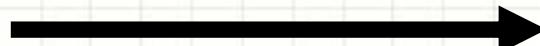
- Lors d'un redimensionnement les composants changent tous de taille mais leurs positions relatives ne changent pas.



SWING: le GridLayout



Redimensionnement



SWING: le GridBagLayout

- Le gestionnaire GridBagLayout fournit des fonctions de présentation complexes
 - ◆ basées sur une grille dont les lignes et les colonnes sont de taille variables.
 - ◆ permet à des composants simples de prendre leur taille préférée au sein d'une cellule, au lieu de remplir toute la cellule.
 - ◆ permet aussi l'extension d'un même composant sur plusieurs cellules.
- Le constructeur GridBagLayout n'admet aucun paramètre, ils sont passés par la méthode add via un objet de type GridBagConstraints.

SWING: la classe GridBagConstraints

- La classe GridBagConstraints
 - Le gestionnaire GridBagLayout utilise plus d'une dizaine de paramètres, pour éviter une surcharge trop lourde de la méthode add, on passe un objet de la classe GridBagConstraints qui représente tout ou partie de ces paramètres.
 - *GridBagConstraints contraintes;*
...
monConteneur.setLayout(new GridBagLayout());
...
contraintes = new GridBagConstraints();
...
monConteneur.add(monComposant , contraintes);

Le positionnement sur la grille (1)

- Les deux premières variables sont la position sur la grille, représentée par gridx et gridy dans la classe GridBagConstraints.
- La taille de la grille est calculée en fonction des maxima des variables gridx et gridy mais aussi de leurs valeurs relatives.

Le positionnement sur la grille (2)

```
public class TestGridBagLayout extends JPanel {  
    ...  
    this.setLayout(new GridBagLayout());  
    GridBagConstraints constraintes = new GridBagConstraints();  
  
    constraintes.gridx=0;  
    constraintes.gridy=0;  
    this.add(new JButton("Un"), constraintes);  
  
    constraintes.gridx=1;  
    constraintes.gridy=0;  
    this.add(new JButton("Deux"), constraintes);  
  
    constraintes.gridx=0;  
    constraintes.gridy=1;  
    this.add(new JButton("Trois"), constraintes);  
  
    constraintes.gridx=1;  
    constraintes.gridy=1;  
    this.add(new JButton("Quatre"), constraintes);  
  
    constraintes.gridx=2;  
    constraintes.gridy=2;  
    this.add(new JButton("Cinq"), constraintes);  
    ...
```



Remplissage des cellules (1)

- Il est possible de demander le remplissage des cellules à l'aide de la variable fill.
- Quatre constantes sont utilisées pour le réglage:
 - GridBagConstraints.NONE : aucun remplissage des cellules (résultat ci-dessus),
 - GridBagConstraints.HORIZONTAL : remplissage dans le sens horizontal,
 - GridBagConstraints.VERTICAL : remplissage dans le sens vertical,
 - GridBagConstraints.BOTH : remplissage dans les deux sens.

Remplissage des cellules (2)

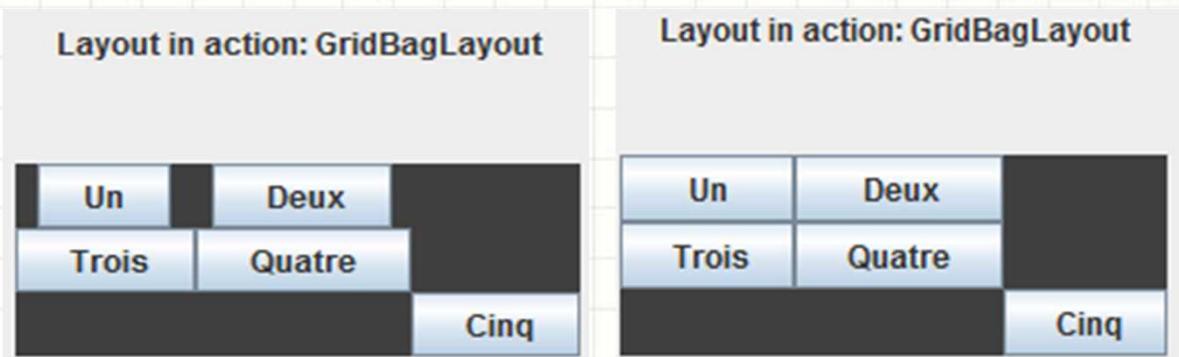
- Pour que cette valeur soit prise en compte il faut que les valeurs weightx et weighty de l'objet GridBagConstraints soient non nulles.
- Ce comportement est aussi utilisé si la fenêtre est redimensionnée.

```
GridBagConstraints gbc = new GridBagConstraints();
```

```
gbc.fill = GridBagConstraints.BOTH;
```

```
gbc.weightx = 1.0;
```

```
gbc.weighty = 1.0;
```



Nombre de cases occupées

- L'un des points forts de GridBagLayout est la possibilité de placer des composants sur plusieurs lignes et/ou plusieurs colonnes.
- Ce sont les variables gridwidth et gridheight qui définissent la largeur et la hauteur d'un composant (en nombre de cellules).

Nombre de cases occupées (exple)

```
gbc. fill = GridBagConstraints.BOTH;
gbc. weightx =1.0;
gbc. weighty =1.0;
gbc. gridx =0;
gbc. gridy =0;
panel.add (new JButton ("Un") , gbc);

gbc. gridx =1;
gbc. gridy =0;
gbc. gridwidth=2;
panel.add (new JButton (" Deux") , gbc);

gbc. gridx =0;
gbc. gridy =1;
gbc. gridwidth=1;
gbc. gridheight=2;
panel.add (new JButton (" Trois") , gbc);

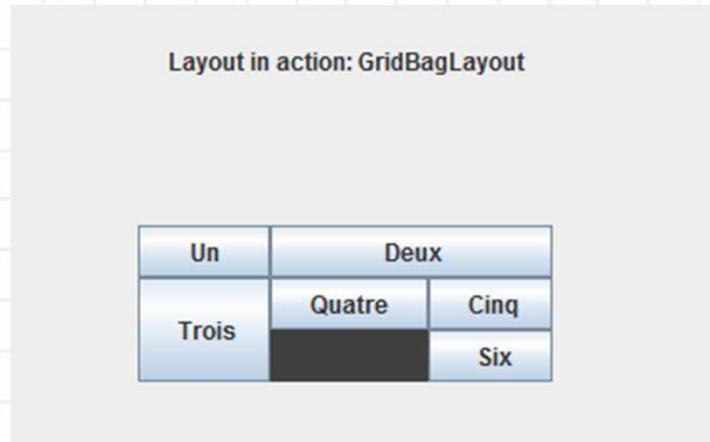
gbc. gridx =1;
gbc. gridy =1;
gbc. gridwidth=1;
gbc. gridheight=1;
panel.add (new JButton (" Quatre ") , gbc);

gbc. gridx =2;
gbc. gridy =1;
gbc. gridwidth=1;
gbc. gridheight=1;
panel.add (new JButton (" Cinq") , gbc);

gbc. gridx =2;
gbc. gridy =2;
gbc. gridwidth=1;
gbc. gridheight=1;
panel.add [new JButton (" Six") , gbc];
```

Nombre de cases occupées

- Les composants se placent alors en fonction des contraintes pour obtenir l'affichage suivant:





Le poids des composants (1)

- Lorsque la variable `fill` est utilisée, les composants se redimensionnent pour occuper tout l'espace possible.
- Il est possible de paramétriser la manière dont les composants vont se partager cet espace supplémentaire à l'aide des poids : `weightx` et `weighty`.
- Les poids sont évalués et les composants occupent l'espace libre de manière proportionnelle à leurs poids

Le poids des composants (2)

- Exemple

```
//Poids des composants
gbc.fill = GridBagConstraints.BOTH;
gbc.weightx =1;
gbc.weighty =1;
gbc.gridx =0;
gbc.gridy =0;
panel.add (new JButton ("Un") , gbc);

gbc.gridx =1;
gbc.gridy =0;
gbc.weightx =2;
panel.add (new JButton (" Deux") , gbc);

gbc.gridx =2;
gbc.gridy =0;
gbc.weightx =4;
panel.add (new JButton (" Trois") , gbc);

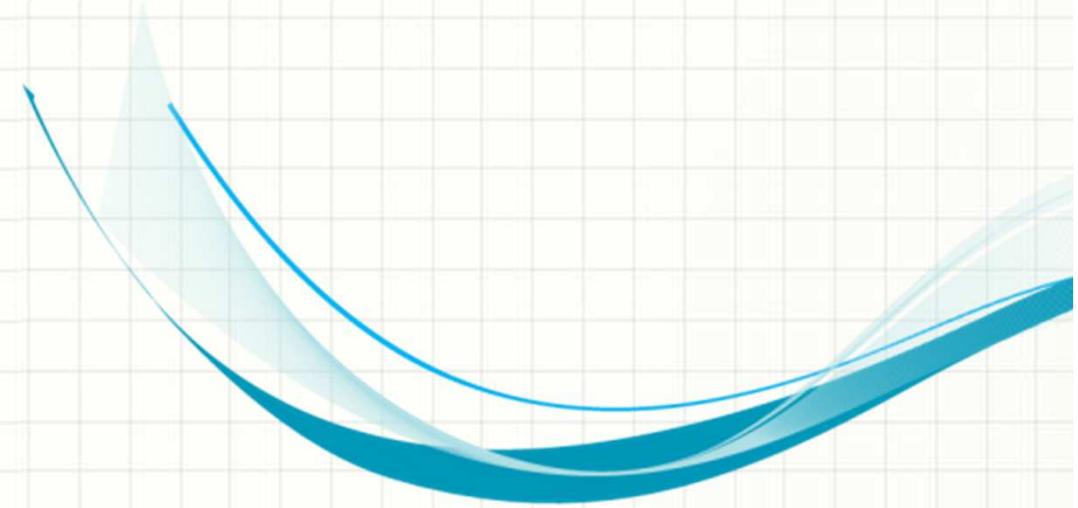
controlPanel.add(panel);
```

- Résultat



Récapitulatif

- FlowLayout
 - ◆ Flux : composants placés les uns derrière les autres
- BorderLayout
 - ◆ Ecran découpé en 5 zones (« North », « West », « South », « East », « Center »)
- GridLayout
 - ◆ Grille : une case par composant, chaque case de la même taille
- GridBagLayout
 - ◆ Grille complexe : plusieurs cases par composant

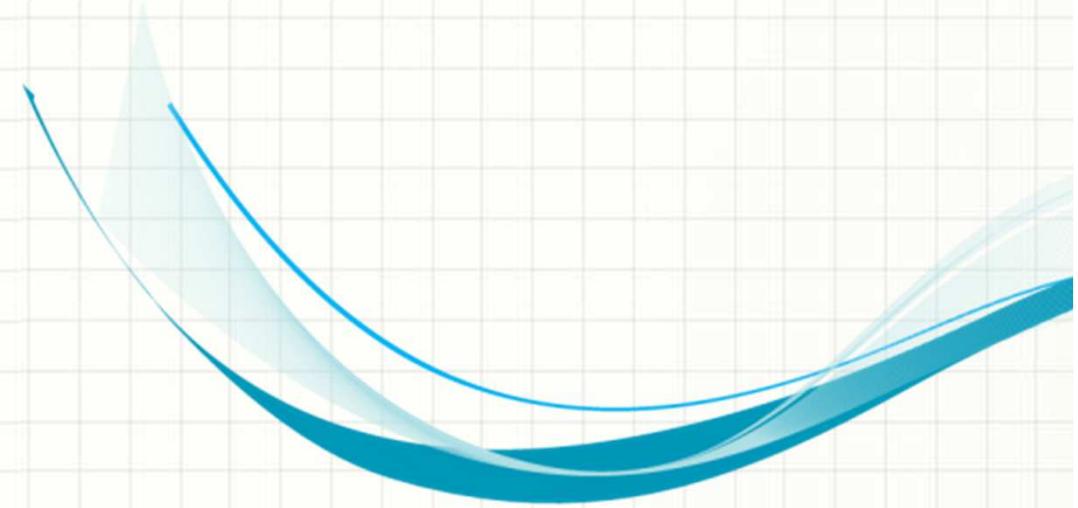


Construction d'une interface graphique



Démarche générale

- Les principales étapes de la démarche pour la construction d'une interface graphique sont:
 - Présentation de l'interface voulue
 - Identifier les différents composants : menus, boutons, zone de texte, label, liste, etc.
 - Choix du/des gestionnaire(s) de placement
 - Grille de placement
 - Identification du gestionnaire de placement



Partie 4 : le contrôleur – gestion des évènements et traitements métier



Cours 4 : Gestion des évènements et traitements

- Gestion de l'interface MouseListener
- Personnalisation de l'objet
- Les classes « Adapters »
- Gestion du Listener avec une classe anonyme
- Mettre fin à l'application
- Les classes métier
- Les traitements métier

Gestion des évènements: principe

- Java propose des mécanismes de communication entre les objets basés sur des évènements.
- L'évènement est émis par un objet (la source) et reçu par un ou plusieurs objets (les auditeurs). On dit que la source notifie un évènement aux auditeurs
- Pour pouvoir recevoir un (ou des) évènement(s) les auditeurs doivent s'enregistrer au près de la (des) source(s).

Gestion des évènements: principe

- Exemple de mise en œuvre

```
private void showEventDemo() {
    headerLabel.setText("Control in action: Button");

    JButton okButton = new JButton("OK");
    JButton submitButton = new JButton("Submit");
    JButton cancelButton = new JButton("Cancel");

    okButton.setActionCommand("OK");
    submitButton.setActionCommand("Submit");
    cancelButton.setActionCommand("Cancel");

    okButton.addActionListener(new ButtonClickListener());
    submitButton.addActionListener(new ButtonClickListener());
    cancelButton.addActionListener(new ButtonClickListener());

    controlPanel.add(okButton);
    controlPanel.add(submitButton);
    controlPanel.add(cancelButton);
    controlPanel.addMouseListener(new CustomMouseListener());

    mainFrame.setVisible(true);
}

private class ButtonClickListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        if (command.equals( "OK" )) {
            statusLabel.setText("Ok Button clicked.");
        }
        else if( command.equals( "Submit" ) ) {
            statusLabel.setText("Submit Button clicked.");
        }
        else {
            statusLabel.setText("Cancel Button clicked.");
        }
    }
}
```

Gestion des évènements dans SWING

- Tous les composants de Swing (et de AWT) créent des évènements en fonction des actions de l'utilisateur.
- Les évènements sont des descendants de EventObject, notés XXXEvent.
- Les composants proposent des méthodes du type addXXXListener pour enregistrer un auditeur.
- L'auditeur doit implémenter l'interface correspondante qui est de la forme XXXListener



Les différents évènements

- Dans le cadre de Swing deux types d'évènements existent : les évènements “de base” et les évènements sémantiques.
- Les premiers, qui sont communs à tous les descendants de Component (et donc de JComponent), couvrent les évènements bas-niveau comme les mouvements de la souris, la modification des composants ou l'utilisation du clavier

Les différents évènements

- Les évènements sémantiques représentent des actions de haut-niveau de l'utilisateur comme la sélection d'un menu, la modification du curseur dans les composants textuels.
- Les évènements de base correspondants sont les suivants:

Type d'événement	événement	classe d'événement
Clavier	touche appuyée, touche lâchée ou touche tapée	KeyEvent
Souris	clic, entrée ou sortie du curseur de la souris dans le composant, bouton de la souris appuyé ou lâché. Souris déplacée éventuellement avec un bouton appuyé	MouseEvent
Visibilité	devient visible ou devient non visible	FocusEvent

Les différents évènements

- Les classes modèles et les méthodes associées à ces évènements de base sont les suivants

Type d'événement	événement	classe modèle interface modèle	classe d'événement	nom de méthode associée
Visibilité	devient visible	FocusAdapter FocusListener	FocusEvent	focusGained
	devient non visible			focusLost
Clavier	touche appuyée	KeyAdapter KeyListener	KeyEvent	keyPressed
	touche lâchée			keyReleased
	touche tapée			keyTyped
Souris	clic	MouseAdapter MouseListener	MouseEvent	mouseClicked
	le curseur rentre dans le composant			mouseEntered
	le curseur sort du composant			mouseExited
	un bouton de la souris appuyé			mousePressed
	un bouton de la souris lâché			mouseReleased
	souris déplacée avec un bouton appuyé	MouseMotionAdapter MouseMotionListener	MouseMotionEvent	mouseDragged
	souris déplacée			mouseMoved

Les évènements concernant les fenêtres (1)

- les fenêtres sont sensibles à des évènements concernant leur ouverture, fermeture, mise en icône, etc.
- Les évènements correspondants sont les suivants:

Fenêtre	événement	classe d'événement
normales (JFrame , JDialog et JApplet)	activation, désactivation, ouverture, fermeture, mise en icône et restitution	WindowEvent
internes (JInternalFrame)		InternalFrameEvent

Les évènements concernant les fenêtres (2)

- les contrôleurs seront obtenus par héritage de la classe modèle. Les classes modèles et les méthodes associés à ces évènements sont:

Fenêtre	événement	classe modèle interface modèle	classe d'événement	nom de méthode associée
normales : JFrame , JDialog et JApplet	La fenêtre devient active (elle recevra les saisies au clavier)	WindowAdapter WindowListener	WindowEvent	windowActivated
	La fenêtre devient inactive			windowDeactivated
	La fenêtre est fermée			windowClosed
	La fenêtre est en cours de fermeture (pas encore fermée)			windowClosing
	La fenêtre est mise ne icône			windowIconified
	La fenêtre est restaurée			windowDeiconified
	La fenêtre est visible pour la première fois			windowOpened
internes : JInternalFrame	La fenêtre devient active (elle recevra les saisies au clavier)	InternalFrameAdapter InternalFrameListener	InternalFrameEvent	InternalFrameActivated
	La fenêtre devient inactive			InternalFrameDeactivated
	La fenêtre est fermée			InternalFrameClosed
	La fenêtre est en cours de fermeture (pas encore fermée)			InternalFrameClosing
	La fenêtre est mise ne icône			InternalFrameIconified
	La fenêtre est restaurée			InternalFrameDeiconified
	La fenêtre est visible pour la première fois			InternalFrameOpened

Les évènements des composants UI

Type d'événement	événement	classe d'événement
Action	action sur le composant (clic sur un bouton, choix dans un menu ...)	ActionEvent
Ajustement	déplacement d'un ascenseur	AdjustmentEvent
Changement	modification de la position d'un composant	ChangeEvent
Elément	modification de l'état d'un élément (liste, case à cocher ...)	ItemEvent
Document	modification dans un texte	DocumentEvent
Curseur	déplacement du curseur d'insertion dans un texte	CaretEvent
Sélection	sélection d'un élément dans une liste	ListSelectionEvent

- Le tableau suivant donne les évènements par composant:

Les évènements des composants UI

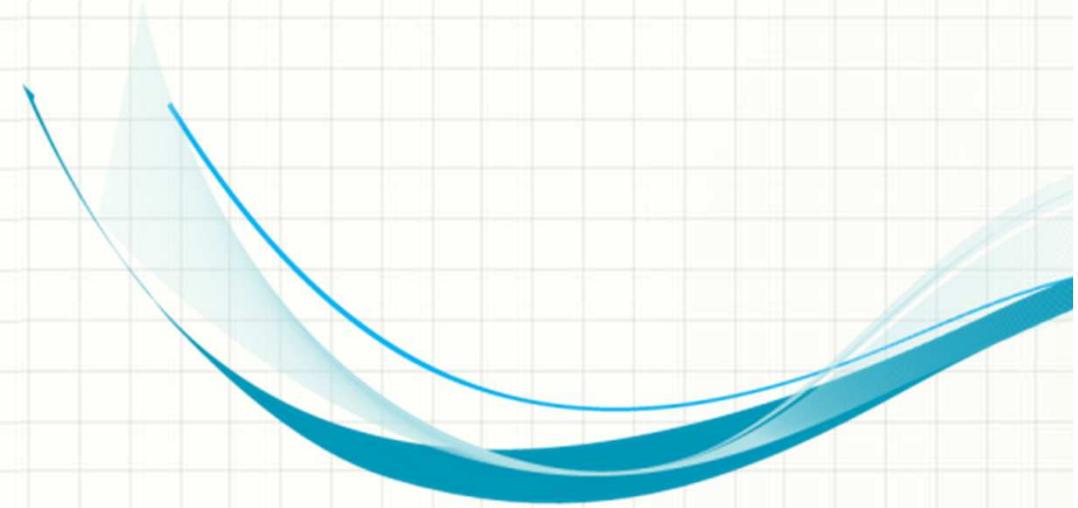
- Le tableau suivant donne les évènements par composant:

Evénement Composant	Action	Ajustement	Changement	Elément	Document	Curseur	Sélection
JButton	X		X	X			
JCheckBox	X		X	X			
JComboBox				X			
JList							X
JScrollBar		X					
JSlider			X				
JProgressBar			X				
JTextField	X				X ⁽¹⁾	X	
JTextArea					X ⁽¹⁾	X	
JMenu	X		X	X			
JMenuItem	X		X	X			
FileChooser	X						

Les évènements des composants UI

- Les interfaces de contrôle et les méthodes associées sont les suivants:

Type d'événement	événement	interface de contrôle	classe d'événement	nom de méthode associée
Action	activation de bouton, case cochée, choix dans un menu ou choix d'un fichier, saisie de texte	ActionListener	ActionEvent	actionPerformed
Ajustement	modification de la position de l'ascenseur	AdjustmentListener	AdjustmentEvent	adjustmentValueChanged
Changement	modification de la position d'un curseur ou d'une barre de progression	ChangeListener	ChangeEvent	stateChanged
Elément	sélection d'un élément	ItemListener	ItemEvent	itemStateChanged
Document	modification, insertion ou suppression de texte	DocumentListener	DocumentEvent	changedUpdate removeUpdate insertUpdate
Curseur	déplacement du curseur d'insertion	CaretListener	CaretEvent	caretUpdate
Sélection	sélection d'un ou plusieurs éléments	ListSelectionListener	ListSelectionEvent	valueChanged



Construction d'applications graphiques



Principes et vocabulaire

- Une application graphique est composée de trois types d'éléments
 - L'interface graphique qui regroupe les composants, les conteneurs et les gestionnaires de placement
 - Les évènements qui émis par les composants et reçus par les éditeurs
 - La ou les classes d'applications qui traitent les données , manipulent les fichiers, accèdent aux bases de données



Principes et vocabulaire

- Selon le modèle MVC
 - le modèle est la partie qui gère le comportement de l'application : traitement des données, manipulation de fichiers, interactions avec les bases de données, connections réseaux,,
 - la vue est l'interface par laquelle l'utilisateur agit. Ce peut être une interface graphique, des pages HTML,.
 - le contrôleur se place entre ces deux éléments et transmet les requêtes de la vue au modèle. En théorie, il n'effectue aucun traitement

Principes et vocabulaire

- Relation entre les différentes entités dans le modèle MVC





Les différentes approches

- 1ere approche : Modèle, Vue et Controleur dans la même classe
 - Dans ce cas dans la même classe on a :
 - la gestion des éléments graphiques,
 - la gestion des évènements se fait par des classes internes anonymes,
 - Les règles de gestion sont implémentées par des méthodes dédiées
 - Cette implémentation est fonctionnelle mais peu évolutive. Pour des applications simples cette solution est rapide à mettre en oeuvre
 - Voir exemple



Les différentes approches

- 2ème approche : Modèle, Vue et Controleur sont séparés
 - On sépare les modèle, vue et contrôleur dans différentes classes
 - Pour des questions de sécurité les données seront stockées sous la forme de membre private et accédées à l'aide de gette et setter
- La classe modèle
 - On y place tous les éléments liés à la modélisation du problème



Les différentes approches

- La classe contrôleur: un seul contrôleur
 - Elle implémente les interfaces correspondant aux événements qu'elle va gérer
 - Elle instancie la classe modèle
 - La classe ne comporte que des méthodes liées aux événements
- La classe vue
 - Instanciation des composants et gestion du Layout
 - Le contrôleur est instancié dans la vue
 - On associe les listener aux contrôles correspondants



Les différentes approches

- Cette approche sépare clairement les trois éléments du modèle MVC en trois classes
- L'ajout d'une fonctionnalité est facile
- La nécessité d'identifier le composant dans le contrôleur peut être problématique



Les différentes approches

- La classe contrôleur : autant de contrôleurs que d'évènements possibles:
 - Les différents contrôleurs doivent se référer à la même vue et au même modèle.
 - La solution la plus simple consiste `a leur passer la vue et le modèle en paramètre lors de la construction
- La classe vue
 - Le modèle est créé dans le vue
 - Il est passé en paramètre au contrôleur



Les différentes approches

- 3ème approche : Vue et Controleur sont groupés, le modèle est autonome
 - Une même classe regroupe la vue et le contrôleur
 - Le modèle est dans une autre classe