

Introduction à Java Annexes: GUI, Multithreading & JDBC

Nicolas van Zeebroeck
IRIDIA – Université Libre de Bruxelles



Introduction à Java Annexes: GUI, Multithreading & JDBC

A. Interfaces graphiques et Gestion d'événements



Interfaces graphiques Java: SWING et AWT

- **Qu'est-ce qu'une interface graphique (GUI)?**
- **Le paradigme « Model-View-Control » (MVC)**
- **L'affichage et ses composants (SWING)**
 - Structure de l'API SWING
 - Composants et conteneurs
 - Composants et conteneurs utiles
 - Exemples
 - Constructeurs
 - Méthodes utiles
- **La gestion des événements (AWT)**
 - Principes
 - Mise en œuvre
- **Connecter l'interface graphique à l'application OO et à la BD**

Interfaces graphiques Java: SWING et AWT

Qu'est-ce qu'une interface graphique?

- **Une interface graphique ou GUI (*Graphical User Interface*) est:**
 - Une couche applicative destinée à fournir aux utilisateurs d'ordinateurs un moyen attractif et facile pour interagir avec un programme
 - L'interface d'un programme qui profite des capacités graphiques d'un ordinateur pour le rendre plus facile d'utilisation. Des interfaces graphiques bien conçues évitent à l'utilisateur d'avoir à apprendre de complexes langages de commandes.
 - Un ensemble d'écrans de présentations qui utilisent des éléments graphiques (tels boutons et icônes) pour rendre un programme plus facile d'utilisation.

Interfaces graphiques Java: SWING et AWT

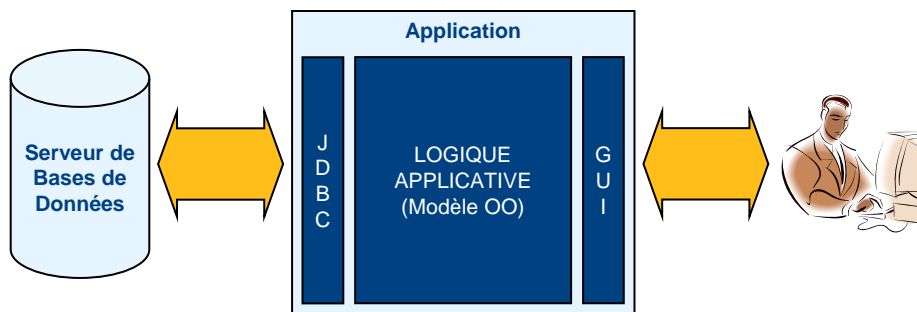
Qu'est-ce qu'une interface graphique?

- Une même application peut offrir différentes interfaces à l'utilisateur
 - Une interface graphique *stricto sensu* qui s'exécute sur son ordinateur
 - Une interface web qui s'exécute dans un navigateur Internet
 - Une interface WAP qui s'exécute sur un téléphone
 - Une interface « terminal » qui s'exécute sur des appareils spécifiques
 - Des « services web » qui offrent ses fonctionnalités à d'autres applications

Interfaces graphiques Java: SWING et AWT

Qu'est-ce qu'une interface graphique?

- Notons que l'interface peut:
 - Etre intégrée à l'application (elle forme un tout avec l'application) (architecture « 2-tiers »)

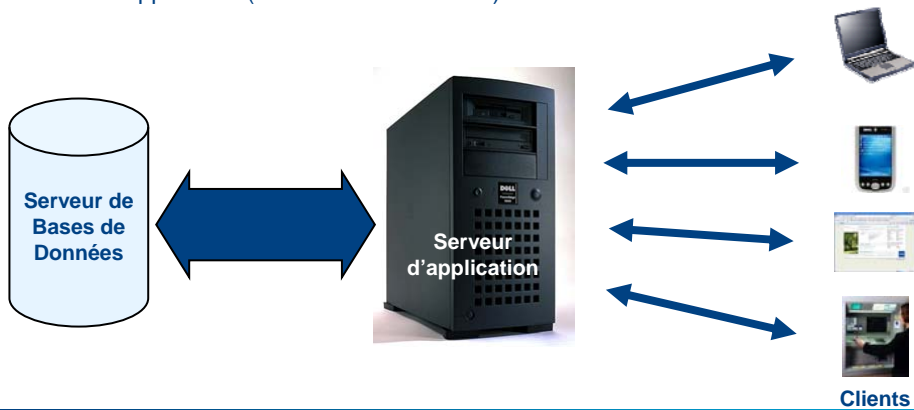


Interfaces graphiques Java: SWING et AWT

Qu'est-ce qu'une interface graphique?

- **Notons que l'interface peut:**

- Etre indépendante de l'application (différentes interfaces indépendantes utilisent la même application sous-jacente) et ne sont que la « partie visible » de l'application (architecture « 3-tiers »)



Interfaces graphiques Java: SWING et AWT

Model-View-Control

- **Fondement: Séparer**

- Les responsabilités relatives à la saisie des événements
- Celles relatives à l'exécution des commandes en réponse aux événements

- **Séparer au mieux**

- La gestion de l'affichage
- Le contrôle du composant
- Les informations intégrées dans le composant

- **Avantages:**

- Un même événement peut être envoyé à plusieurs objets écouteurs
 - Utile si un événement est potentiellement intéressant pour plusieurs écouteurs
- Facilite la réutilisation des composants
- Permet le développement de l'application et de l'interface séparément
- Permet d'hériter de super-classes différentes suivant les fonctionnalités
- Règle essentielle en OO: modulariser au plus ce qui est modularisable

Les interfaces graphiques AWT v/s SWING

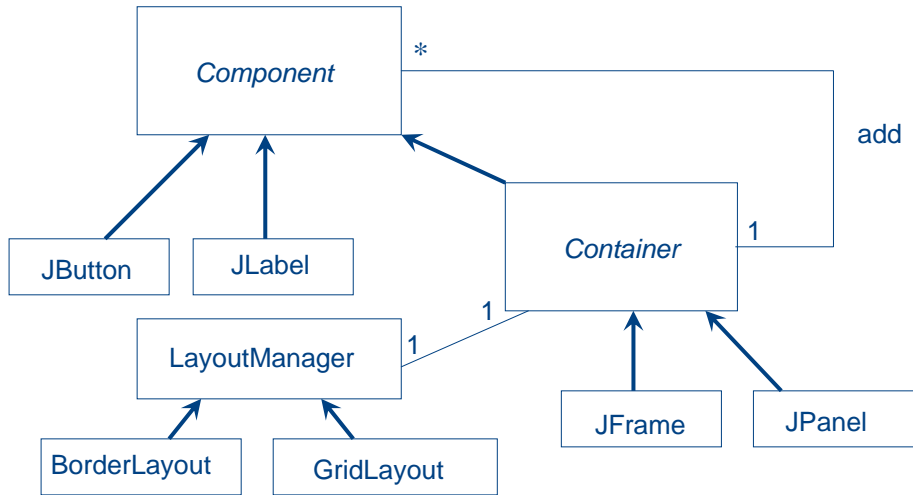
- **Les composants des interfaces graphiques sont fournis en Java par deux packages particuliers (et concurrents):**
- **Java 1: AWT (Abstract Window Toolkit)**
 - Composants graphiques « lourds »
 - Chaque composant est relié à son équivalent dans l'OS par un « peer »
 - Look & Feel dépendant de l'OS
- **Java 2: SWING**
 - Nouveau package
 - Composants graphiques « légers », en pur Java
 - Tous les composants sont détachés de l'OS
 - Look & Feel indépendant de l'OS

Interfaces graphiques Java: SWING et AWT Affichage (SWING)

- **SWING offre trois types d'éléments graphiques**
 - Les « Components » (composants ou contenus)
 - Les « Containers » (contenants, qui sont eux-mêmes des composants)
 - Les « LayoutManagers »
- **Les « Components »**
 - Constituent différents éléments de l'affichage (boutons, barres de menus, etc.)
 - Ex: JButton, JLabel, JTextArea, JCheckbox, etc.
- **Les « Containers »**
 - Sont destinés à accueillir des composants
 - Gèrent l'affichage des composants
 - Ex: JFrame, JPanel, JScrollPane
- **Les « LayoutManagers »**
 - Gèrent la disposition des composants au sein d'un conteneur

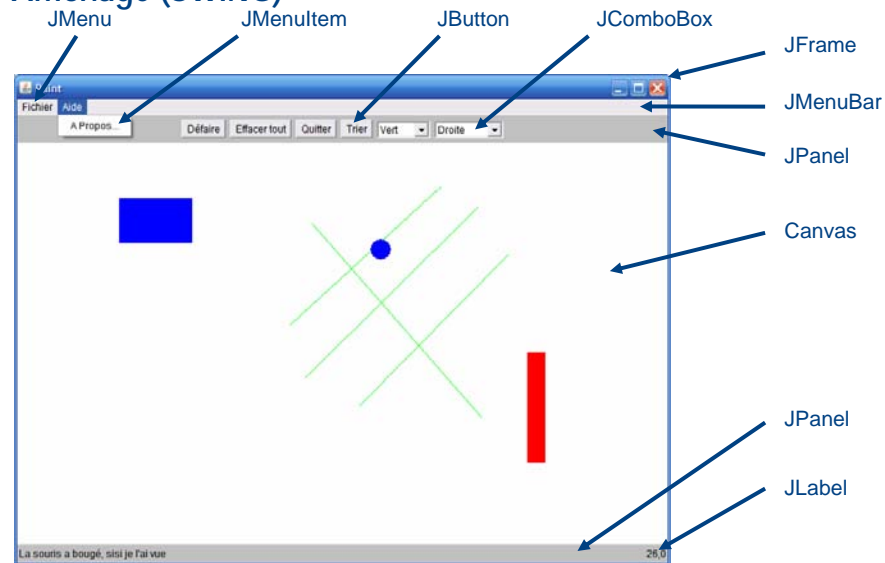
Interfaces graphiques Java: SWING et AWT

Affichage (SWING)



Interfaces graphiques Java: SWING et AWT

Affichage (SWING)



Interfaces graphiques Java: SWING et AWT

Affichage (SWING)

• Exemples de composants:

- JTextArea (zone de texte)
 - Contient une chaîne de caractères
- JTable (table de données)
 - Contient deux Arrays ([]):
 - Les données (tableau multidimensionnel d'objets)
 - Les intitulés des colonnes (vecteur de chaînes de caractères)
 - Gère automatiquement l'affichage des données en colonnes
 - Permet le déplacement de colonnes
 - Permet ou interdit la modification des données de la table
- JButton (bouton)
- JCheckBox (case à cocher)
 - Contient un intitulé
 - Peut être sélectionnée ou non
 - Peut être activée ou désactivée (on peut ou non la cocher)
- JLabel (étiquette/intitulé)
 - Contient un intitulé
- JComboBox (liste déroulante)
 - Contient une liste d'éléments sous forme de chaînes de caractères
 - Contient un entier représentant le numéro de la ligne sélectionnée



Nicolas van Zeebroeck

Introduction à Java / Février 2007 / 13
Nicolas.van.Zeebroeck@ulb.ac.be – www.vanzeebroeck.net



Interfaces graphiques Java: SWING et AWT

Affichage (SWING)

• Exemples de conteneurs

- JFrame
 - Composant du plus haut niveau
 - La fenêtre d'une application est une instance de cette classe
 - Le Frame contient les différents composants graphiques de l'application
 - Ne peut être intégré dans un autre conteneur
 - Peut contenir une barre de menu (JMenuBar)
- JPanel
 - Peut être lui-même intégré dans un autre conteneur (par ex. un JFrame)
 - Peut contenir lui-même autant de composants que nécessaire
 - Plusieurs moyens d'y disposer les composants
- JScrollPane
 - Peut être lui-même intégré dans un autre conteneur (par ex. un JFrame)
 - Offre une « vue » sur un seul composant ou conteneur (JTable, JPanel, etc.)
 - Permet de changer le contenu en cours de route
 - Offre automatiquement des barres de défilement si nécessaire



Introduction à Java / Février 2007 / 14
Nicolas.van.Zeebroeck@ulb.ac.be – www.vanzeebroeck.net



Interfaces graphiques Java: SWING et AWT

Affichage (SWING)

- **Barres et éléments de menus**

- JMenuBar
 - Une barre de menu (une seule par JFrame)
 - Contient différents menus (JMenu)
- JMenu
 - Représente un menu dans une barre de menu
 - Contient différents éléments (JMenuItem)
 - Contient un intitulé
- JMenuItem
 - Représente un élément de menu
 - Contient un intitulé
 - Peut être associé à des raccourcis clavier

Interfaces graphiques Java: SWING et AWT

Affichage (SWING)

- **Constructeurs des composants:**

- JTextArea (zone de texte)
 - `new JTextArea(String texteAAfficher);`
- JTable (table de données)
 - `new JTable(Object[][] donnees, String[] intitules);`
 - Il faut un tableau de chaînes de caractères contenant les intitulés des colonnes
 - Et une matrice d'objets dont chaque rangée est une ligne du tableau
- JButton (bouton)
 - `new JButton(String intitule);`
- JCheckBox (case à cocher)
 - `new JCheckBox(String intitule);`
- JLabel (étiquette/intitulé)
 - `new JLabel(String intitule);`
- JComboBox (liste déroulante)
 - `new JComboBox();`

Interfaces graphiques Java: SWING et AWT

Affichage (SWING)

• Constructeurs des conteneurs

- JFrame
 - `new JFrame();`
- JPanel
 - `new JPanel();`
- JScrollPane
 - `new JScrollPane();`

• Constructeurs des menus

- JMenuBar
 - `new JMenuBar();`
- JMenu
 - `new JMenu(String intitulé);`
- JMenuItem
 - `new JMenuItem(String intitulé);`

Interfaces graphiques Java: SWING et AWT

Affichage (SWING)

• Méthodes utiles (composants):

- JTextArea (zone de texte)
 - `setText(String)` → Modifier le contenu
- JTable (table de données)
 - `setSelectionMode(int)` → Définir le mode de sélection (ex: une ligne à la fois)
 - `getSelectionMode()` → Récupérer le mode de sélection défini
- JButton (bouton)
 - `setText(String)` → Modifier l'intitulé
- JCheckBox (case à cocher)
 - `setText(String)` → Modifier l'intitulé
 - `setEnabled(boolean)` → Déterminer si la case est activée ou pas (peut être cochée ou non)
 - `isSelected(boolean)` → Déterminer si la case est cochée ou pas
 - `isEnabled()` → Renvoie *true* si la case est activée et *false* si elle ne l'est pas
 - `isSelected()` → Renvoie *true* si la case est cochée et *false* si elle ne l'est pas
- JLabel (étiquette/intitulé)
 - `setText(String)` → Modifie l'intitulé
- JComboBox (liste déroulante)
 - `addItem(String)` → Ajouter un élément (chaîne de caractères) dans la liste
 - `getSelectedIndex()` → Renvoie le numéro de la ligne sélectionnée

Interfaces graphiques Java: SWING et AWT

Affichage (SWING)

• Méthodes utiles (conteneurs)

• JFrame

- setSize(int width, int height) → Fixe les dimensions de la fenêtre
- setTitle(String titre) → Définit le titre de la fenêtre
- setJMenuBar(JMenuBar) → Installe la barre de menu
- add(Component c) → Intègre le composant indiqué
- setVisible(true) → Fait apparaître la fenêtre à l'écran

NORTH		
WEST	CENTER	EAST
SOUTH		

• JPanel

- setLayout(new BorderLayout()) → Divise le conteneur en 5 zones: N, S, E, W, C et permet d'ajouter un composant différent dans chaque zone (il n'est pas obligatoire d'utiliser les 5 zones)
- add(Component c, "North") → Ajoute le composant « c » au « Nord »
- setLayout(new GridLayout(x,y)) → Divise le conteneur en x lignes et y colonnes. Les composants ajoutés viennent remplir la grille de gauche à droite et de haut en bas
- add(Component c) → Ajoute le composant « c » dans la première case disponible

• JScrollPane

- setViewportView(Component c) → Affiche le composant/conteneur indiqué

Interfaces graphiques Java: SWING et AWT

Affichage (SWING)

• Méthodes utiles (menus)

• JMenuBar

- add(JMenu) → Ajoute un menu dans la barre (de gauche à droite)

• JMenu

- add(JMenuItem) → Ajoute un élément dans le menu (de haut en bas)
- addSeparator() → Ajoute une ligne de séparation dans le menu

• JMenuItem

- setText(String) → Modifie l'intitulé de l'élément du menu

Interfaces graphiques Java: SWING et AWT

Affichage (SWING)

- **Comment utiliser les composants et conteneurs prédéfinis?**

- Par héritage
 - Créer une nouvelle classe qui hérite de la classe prédéfinie qui nous intéresse
 - Permet de créer « sa » propre fenêtre, « sa » propre barre de menus, etc.
 - Généralement ce qu'on fait avec les conteneurs
 - Plus rarement avec les composants
 - Exemple:

```
-public Fenetre extends JFrame{ ... }
```

- Par association
 - Consiste simplement à déclarer un objet du type du composant prédéfini comme variable membre dans une classe
 - On se sert alors des méthodes du composant pour définir les propriétés du composant qui nous intéressent (intitulé, contenu, état, etc...)
 - Exemple

```
-public ZonePrincipale extends JPanel{  
    JLabel l = new JLabel("Bonjour");
```

Les interfaces graphiques

Les « LayoutManagers » (1/5)

- **Rôle**

- Gérer la disposition des composants au sein d'un conteneur

- **Types principaux:**

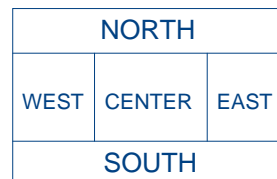
- BorderLayout: divise le conteneur en 5 zones
- FlowLayout: rajoute les composants au fur et à mesure
- GridLayout: applique une grille au conteneur pour aligner les composants
- CardLayout: pour un conteneur qui contient plusieurs cartes
- GridBagLayout: grille de cellules élémentaires

Les interfaces graphiques

Les « LayoutManagers » (2/5)

BorderLayout

- **Principe**
 - Positionne les composants suivants les points cardinaux
 - Layout par défaut des « Frame »
- **Répartition**
 - La largeur l'emporte pour le Nord et le Sud
 - La hauteur l'emporte pour l'Est et l'Ouest
 - Le centre occupe tout ce qui reste
- **Utilisation**
 - `add(unBouton, BorderLayout.NORTH)`
 - `new BorderLayout(int, int)`
 - Intervalles horizontal et vertical entre les éléments



Les interfaces graphiques

Les « LayoutManagers » (3/5)

FlowLayout

- **Principe**
 - Rajoute les composants au fur et à mesure
 - La taille des composants l'emporte
- **Utilisation**
 - `new FlowLayout (int alignment)`
 - alignment: LEFT, CENTER, RIGHT
 - `new FlowLayout (int alignment, int hintervalle,int vintervalle)`
 - alignment: LEFT, CENTER, RIGHT
 - hintervalle: L'intervalle horizontal entre les composants
 - vintervalle: L'intervalle vertical entre les composants

Les interfaces graphiques

Les « LayoutManagers » (4/5)

GridLayout

- **Principe**
 - Définit une grille a 2 dimensions
- **Utilisation**
 - `new GridLayout (int, int)`
 - Définit le nombre de lignes et de colonnes
 - `new GridLayout (int, int, int, int)`
 - Définit le nombre de lignes et de colonnes
 - Définit l'alignement horizontal et vertical

CardLayout

- **Principe**
 - Pour un container qui contient plusieurs cartes
 - Seule une carte est visible à chaque fois
 - L'utilisateur peut passer d'une carte à l'autre

Les interfaces graphiques

Les « LayoutManagers » (5/5)

GridBagLayout

- **Principe**
 - Le plus compliqué et le plus flexible
 - Une grille de cellules élémentaires
 - Les composants graphiques peuvent s'étaler indifféremment sur ces cellules
 - Pour positionner un composant, il faut utiliser un objet `GridBagConstraints`
 - On peut spécifier le comportement des composants quand on étire la fenêtre
- **Utilisation**
 - `new GridBagLayout()`
 - `new GridBagConstraints();`

Exercice

Une première application graphique

- Créer une classe « Fenetre » héritant de *JFrame*
- Construire la fenêtre d'application en ajoutant:
 - Au Nord: un *JPanel*
 - Contenant lui-même trois *JButton*
 - A placer dans un vecteur
 - Au Centre: un *Canvas*
 - Servira d'aire de jeu qui s'occupera de tous les objets du jeu (balle, monstre, avion, princesse,...)
 - Au Sud: un *JPanel*
 - Contenant lui-même deux *JLabel*
- Créer une classe principale contenant uniquement le main:

```
public static void main(String args[]) {  
    new Fenetre();  
}
```

PROJET

- **Votre application devra permettre :**
 - de lister les clients et les réservations existantes
 - d'enregistrer une nouvelle réservation (en tenant compte des dates de priorité)
 - de sélectionner les places associées à une réservation (création des tickets)
 - de recevoir le paiement d'une réservation
 - par débit du porte-monnaie électronique pour les non-abonnés
 - par crédit pour les abonnés

PROJET

Semaine Intensive Solvay 2006

Action Consultation

Liste des réservations

ID	Client	Spéctacle	Représentation	Nb	Réserve le	Payé	Tick.
1	Bersini, Hugues	Avelle Red - Avelle Red en conc...	2007-01-21 20:00:00	1	2006-11-27	Oui	4
11	Bersini, Hugues	Peter Pan - Peter Pan on Ice	2006-12-28 19:00:00	1	2006-11-28	Oui	0
2	Bersini, Hugues	Peter Pan - Peter Pan on Ice	2006-12-30 11:00:00	2	2006-11-15	Oui	2
14	van Zeebroeck, Nic...	Peter Pan - Peter Pan on Ice	2006-12-31 15:00:00	2	2006-11-28	Oui	0
3	van Zeebroeck, Nic...	Pascal Obispo - Obispo en conc...	2007-03-11 18:00:00	1	2006-11-27	Oui	0
4	Philemotte, Christo...	Disney on Ice - Les Princesses	2007-02-14 15:00:00	4	2006-11-28	Oui	4
12	Salihoglu, Utku	Avelle Red - Avelle Red en conc...	2007-01-21 20:00:00	6	2006-11-28	Oui	6
5	Salihoglu, Utku	Peter Pan - Peter Pan on Ice	2006-12-28 19:00:00	1	2006-11-27	Oui	0
6	Pignon, Françoise	Peter Pan - Peter Pan on Ice	2006-12-28 19:00:00	1	2006-11-27	Non	0
7	Brochant, Pierre	Peter Pan - Peter Pan on Ice	2006-12-28 19:00:00	1	2006-11-27	Non	0
8	Grandjeur, Joséphi...	Disney on Ice - Les Princesses	2007-02-14 15:00:00	1	2006-11-27	Non	0
13	Campiana, Lucienne	Peter Pan - Peter Pan on Ice	2006-12-28 19:00:00	4	2006-11-28	Non	0
15	Bonnetot, Josiane	Peter Pan - Peter Pan on Ice	2006-12-29 18:00:00	290	2006-11-28	Non	0
10	Trembleur, Marcel	Michel Polnareff - Polnareff en c...	2007-03-29 20:30:00	1	2006-11-28	Non	0
16	Deville, Jonathan	Peter Pan - Peter Pan on Ice	2006-12-29 19:00:00	10	2006-11-28	Non	0
9	Deville, Jonathan	Johnny Hallyday - Johnny en co...	2007-02-20 20:00:00	10	2006-11-27	Non	0

Ready

Fenêtre d'application (JFrame)

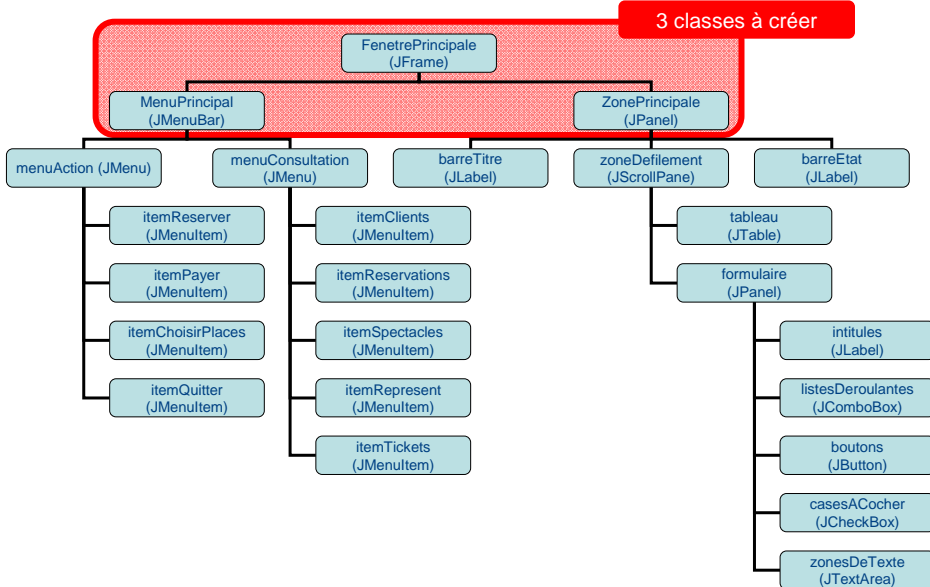
Barre de menus (JMenuBar)

Titre (JLabel)

Zone de défilement (JScrollPane)
pouvant contenir une table
(JTable) ou un formulaire (JPanel)

Barre d'état (JLabel)

PROJET



Gestion d'événements

Mécanismes et structure (1/4)

- **Une source d'événements**
 - Génère des objets événements
 - Les fait écouter par un ensemble d'écouteurs d'événements
 - En général: un composant ou conteneur graphique
- **Les objets événements**
 - xxxEvent
 - Contiennent la description et les caractéristiques d'un événement
- **Les objets écouteurs**
 - xxxListener ou xxxAdapter
 - Concrétisent des méthodes définies dans les Interfaces
 - Indiquent leur réaction en réponse aux événements
 - Sont des interfaces implémentables dans les classes
 - Peuvent être implémentés par les sources d'événements elles-mêmes (Une source d'événements peut « s'écouter » elle-même)

Gestion d'événements

Mécanismes et structure (2/4)

1. **Un événement se produit**
2. **La source d'événement dans laquelle il se produit génère un objet de type événement**
3. **La source transmet l'événement à son (ses) écouteur(s)**
4. **L'écouteur appelle la méthode correspondant au type d'événement et lui passe en argument l'objet événement**
5. **La méthode en question spécifie les traitements à réaliser lorsqu'un événement du type correspondant se produit**
6. **Dans ses traitements, la méthodes peut examiner les caractéristiques de l'événement (position du curseur de la souris, code de la touche pressée au clavier...) et adapter son comportement en fonction de ces caractéristiques**

Gestion d'événements Mécanismes et structure (3/4)

- **Evénements**

- ActionEvent, AdjustmentEvent, ComponentEvent, ContainerEvent, FocusEvent, ItemEvent, KeyEvent, MouseEvent, TextEvent, WindowEvent

- **Les Interfaces Ecouteurs**

- ActionListener, AdjustmentListener, ComponentListener, ContainerListener, FocusListener, ItemListener, KeyListener, MouseListener, MouseMotionListener, WindowListener

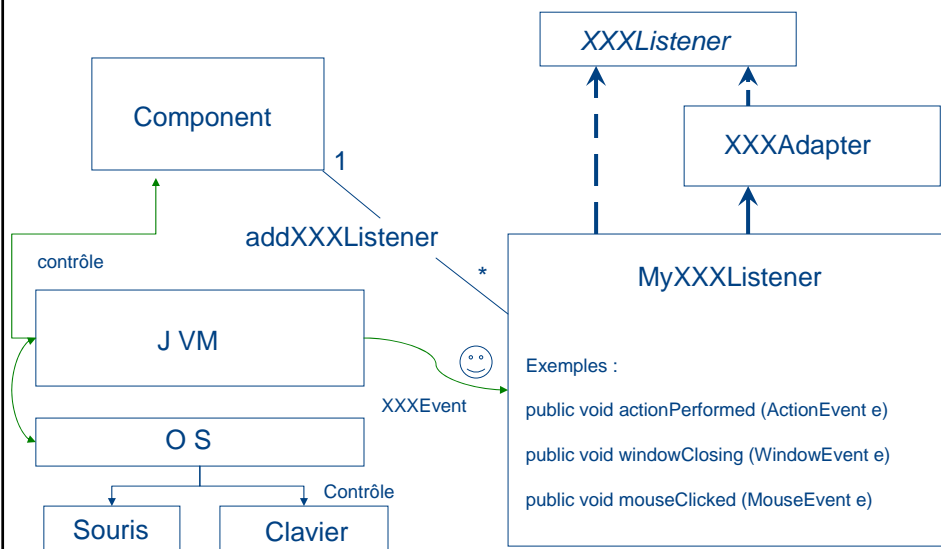
- **Les Adapteurs correspondants**

- ActionAdapter, WindowAdapter, KeyAdapter, MouseAdapter, etc.

- **Les Sources d'événements**

- Button, List, MenuItem, TextField, ScrollBar, CheckBox, Component, Container, Window

Gestion d'événements Mécanismes et structure (4/4)



Gestion d'événements

Mise en œuvre

• Par implémentation de l'interface

- Usage
 - ➔ `public class MaClasse implements MouseListener`
- Avantages et inconvénients
 - ☺ Meilleur sur le plan orienté objet
 - ☺ La classe peut hériter d'une autre classe
 - ☺ Consistance

• Par héritage de l'adapteur

- Usage
 - A chaque interface correspond un adapteur qui l'implémente lui-même
 - ➔ `public class MaClasse extends MouseAdapter`
- Avantages et inconvénients
 - ☺ Code simple (l'adapteur redéfinit déjà les méthodes, etc.)
 - ☹ La classe ne peut plus hériter d'une autre classe

Gestion d'événements

Mise en œuvre – Clics de souris

```
class MonFrame extends Frame implements MouseListener
{
    public MonFrame()
    {
        addMouseListener(this);
    }
    public void mousePressed(MouseEvent e) {}
    public void mouseClicked(MouseEvent e)
    {
        if(e.getX()>50 && e.getY()<100){...}
    }
    public void mouseReleased(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
}
```

Gestion d'événements

Mise en œuvre – Déplacements de souris

```
class MonFrame extends Frame implements MouseMotionListener
{
    public MonFrame()
    {
        addMouseMotionListener(this);
    }
    public void mouseDragged(MouseEvent e) {}
    public void mouseMoved(MouseEvent e)
    {
        if(e.getX()>50 && e.getY()<100){...}
    }
}
```

Gestion d'événements

Mise en œuvre – Clavier

```
class MonFrame extends Frame implements KeyListener
{
    public MonFrame()
    {
        addKeyListener(this);
    }
    public void keyPressed(KeyEvent e) {}
    public void keyTyped(KeyEvent e)
    {
        if(e.getKeyCode()==KeyEvent.VK_Q){System.exit(0)}
    }
    public void keyReleased(KeyEvent e) {}
}
```

Gestion d'événements

Mise en œuvre – Evénements de fenêtre

```
class MonFrame extends Frame implements WindowListener
{
    public MonFrame()
    {
        addWindowListener(this);
    }

    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowOpened(WindowEvent e) {}
    public void windowClosing(WindowEvent e) {System.exit(0);}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
}
```



Gestion d'événements

Mise en œuvre – Actions

```
public class MonPanel extends Panel implements ActionListener
{
    // MonPanel devient un écouteur d'événements
    public void actionPerformed(Action evt)
    {
        // Ce qui est fait en réponse à l'événement...
        // je peux extraire de l'information sur l'événement
        // transmis par evt
        // je suis obligé de re-définir cette méthode abstraite
        // dans ActionListener
    }
}

MonPanel panel = new MonPanel();
JButton button = new JButton("OK");// Crée un bouton qui est source
// d'événements de type ActionEvent
button.addActionListener(panel) // Associer un écouteur
// d'événements panel au bouton
```



Gestion d'événements

Mise en œuvre – Sélection dans une JTable

```
class GereTables implements ListSelectionListener
{
    public void valueChanged(ListSelectionEvent e) {
        if (e.getValueIsAdjusting())
            return;
        ListSelectionModel lsm;
        lsm = (ListSelectionModel) e.getSource();
        if(lsm.isSelectionEmpty()){
            // Que fait si aucune ligne n'a été sélectionnée?
        } else {
            // La ligne sélectionnée est :
            int i = lsm.getMinSelectionIndex();
            // Que faire avec cette information ?
        }
    }
}
```

Gestion d'événements

Mise en œuvre – Associer la source et l'écouteur

- En implémentant ce qui précède dans la classe concernée, on rend alors la classe en question capable de prendre en charge les événements du type correspondant
- Il reste à associer chaque source potentielle d'événements (la fenêtre principale, chaque élément de menu, chaque bouton, chaque liste déroulante et chaque table) à son objet écouteur, instance de la classe correspondante
- Cela se fait en général pendant ou juste après la construction de l'objet source au moyen d'une méthode
« addXXXXXXListener(objetEcouteur) »

Gestion d'événements

Mise en œuvre – Associer la source et l'écouteur

- Pour informer la fenêtre qu'elle est son propre écouteur:
`this.addWindowListener(this);`
- Pour associer un élément de menu à son écouteur :
`clients = new JMenuItem("Liste des clients");`
`clients.addActionListener(new GereMenus());`
- Pour associer un bouton à son écouteur :
`bouton = new JButton("Sauver la réservation");`
`bouton.addActionListener(new GereMenus());`
- Pour associer une table à son écouteur :
`table = new JTable(dataTable,headersTable);`
`table.setSelectionModel(ListSelectionModel.SINGLE_SELECTION);`
`table.getSelectionModel().addListSelectionListener(new GereTables());`

PROJET

- 3 types d'événements (ou actions) doivent être traités :
 - La fermeture de la fenêtre (pour qu'elle entraîne la fin du programme)
 - Les sélections dans les menus et les clics sur les boutons (pour déclencher les actions requises)
 - La sélection d'une ligne dans la table (pour savoir quelle réservation traiter)

Classe Source	Evénements	Interface	Classe Ecouteur
FenetrePrincipale	Fermeture de la fenêtre	WindowListener	FenetrePrincipale
JMenuItem	Choix dans un menu	ActionListener	GereMenus
JButton	Clic sur un bouton	ActionListener	GereMenus
JTable	Sélection d'une ligne	ListSelectionListener	GereTables

- Il faut donc :
 - Rendre la classe FenetrePrincipale capable de se gérer elle-même
 - Créer 2 classes d'écouteurs spécifiques : GereMenus, GereTables (code fourni)
 - Instancier ces 2 classes
 - Associer les écouteurs à la source d'événements qu'ils doivent gérer

Connecter l'interface graphique à l'application OO

- Pour « relier » l'interface graphique à l'application OO, il suffit d'en utiliser des objets dans les classes graphiques
- Tout dépend naturellement des fonctionnalités requises dans l'interface graphique et donc des besoins de celle-ci en matière d'objets
- Par exemple, pour afficher les clients, l'interface graphique aura besoin d'une collection contenant les objets clients générés au départ de la base de données
- L'interface graphique devra donc pouvoir obtenir ces objets, ce qui suppose qu'elle puisse interagir avec la base de données...

Graphisme 2D Fonctionnement

- Tout ce qui est nécessaire aux graphismes en 2D est fourni par la classe **Graphics**
- Chaque composant est toujours doté d'un objet de ce type
 - Graphics g
- On peut donc invoquer, sur cet objet graphique « g », toutes les méthodes de graphisme:
 - g.drawLine(int, int, int, int)
 - g.drawRect(int, int, int, int)
 - g.fillOval(int, int, int, int)
 - g.setColor(Color)
 - Etc.

Graphisme 2D

Classe java.awt.Rectangle

- **Classe fournie dans le package java.awt**
- **Idéale pour servir de squelette à la plupart des objets graphiques**
 - On peut inscrire toutes les formes dans un rectangle
- **Attributs publics:**
 - int x,y, width, height
- **Méthodes publiques**
 - contains(int x, int y): boolean
 - Renvoie *true* si les coordonnées (x,y) sont contenues dans le rectangle
 - contains(Rectangle r): boolean
 - Renvoie *true* si le rectangle r est contenu dans le rectangle en cours
 - intersects(Rectangle r): boolean
 - Renvoie *true* si le rectangle r touche le rectangle en cours
 - ...

Exercice

EX 7.3

Paint (1/4)

- L'objectif du projet est la réalisation d'un éditeur graphique permettant de dessiner des segments de droites colorés (une version BETA de Paint 1.0)
- La réalisation de l'éditeur graphique commencera par la construction d'une fenêtre (Frame), support de l'application, que nous personnalisons grâce à divers éléments : boutons (Button), listes déroulantes (Choice), zone de dessin (Canvas), etc.
- Ensuite nous gérerons les événements générés par l'utilisateur (clic de souris, ...) pour donner vie à cette application
- Enfin en utilisant le contexte graphique fourni par la JVM, nous aborderons les mécanismes de construction graphique en 2D

Introduction à Java Annexes: GUI, Multithreading & JDBC

B. Multithreading



Survol du chapitre

- **Introduction**
 - Définition
 - Raison d'être
- **Création de Thread**
 - Par implémentation
 - Par héritage
- **Gestion de Thread**
 - Méthodes de gestion
 - Diagrammes d'état

Introduction

Qu'est-ce qu'un Thread?

- **Un ordinateur qui exécute un programme :**
 - Possède un CPU
 - Stocke le programme à exécuter
 - Possède une mémoire manipulée par le programme
 - Multitasking géré par l'OS
- **Un thread (« file ») a ces mêmes capacités**
 - A accès au CPU
 - Gère un processus
 - A accès à la mémoire, qu'il partage avec d'autres files
 - Multithreading géré par la JVM

Introduction

Pourquoi le multithreading?

- **Un programme moderne est composé de**
 - Une interface graphique
 - Quelques composantes pouvant agir de manière autonome
- **Sans multithreading**
 - Les composantes ne pourraient agir que lorsque l'interface est suspendue
- **Plus généralement, le multithreading**
 - Permet de réaliser plusieurs processus indépendants en parallèle
 - Permet de gérer les files d'attente
 - Permet au besoin de synchroniser les différents processus entre eux

Création de Thread

Mise en œuvre (1/3)

• Par implémentation de l'interface

- Usage
 - `public void MaClasse implements Runnable`
- Avantages et inconvénients
 - ☺ Meilleur sur le plan orienté objet
 - ☺ La classe peut hériter d'une autre classe
 - ☺ Consistance

• Par héritage de la classe Thread elle-même

- Usage
 - `public void MaClasse extends Thread`
- Avantages et inconvénients
 - ☺ Code simple (l'objet est un Thread lui-même)
 - ☹ La classe ne peut plus hériter d'une autre classe

Création de Thread

Mise en œuvre (2/3)

```
public class MaFile implements Runnable {
    public void run(){
        byte[] buffer=new byte[512];
        int i=0;
        while(true){
            if(i++%10==0)System.out.println(""+i+" est divisible par 10");
            if (i>101) break;
        }
    }
}

public class LanceFile {
    public static void main(String[]arg){
        Thread t=new Thread(new MaFile());
        t.start();
    }
}
```

Le constructeur de la classe Thread attend un objet Runnable en argument

Création de Thread

Mise en œuvre (3/3)

```
public class MyThread extends Thread {
    public void run(){
        byte[] buffer=new byte[512];
        int i=0;
        while(true){
            if(i++%10==0) System.out.println(""+i+" est divisible par 10");
            if(i>101) break;
        }
    }
}

public class LaunchThread{
    public static void main(String[] larg){
        MyThread t=new MyThread();
        t.start();
    }
}
```

Grâce à l'héritage, un objet de type MyThread est lui-même Runnable, on peut donc appeler un constructeur sans argument

Gestion des Thread

Méthodes de gestion (1/4)

- `t.start()`
 - Appeler cette méthode place le thread dans l'état "runnable"
→ Eligible par le CPU
- `t.yield()` throws `InterruptedException`
 - La VM arrête la file active et la place dans un ensemble de files activables. (runnable state)
 - La VM prend une file activable et la place dans l'état actif (running state)
- `t.sleep(int millis)` throws `InterruptedException`
 - La VM bloque la file pour un temps spécifié (état « d'attente »)
- `t.join()` throws `InterruptedException`
 - Met la file en attente jusqu'au moment où la file t est terminée (a fini sa méthode `run()`). Le thread appelant redevient alors activable.

Gestion des Thread Méthodes de gestion (2/4)

- `Thread.yield()` throws `InterruptedException`
 - La VM arrête la file active et la place dans un ensemble de files activables. (runnable state)
 - La VM prend une file activable et la place dans l'état actif (running state)
- `Thread.sleep(int millis)` throws `InterruptedException`
 - La VM bloque la file pour un temps spécifié (état « d'attente »)
- `t.join()` throws `InterruptedException`
 - Met la file en attente jusqu'au moment où la file t est terminée (a fini sa méthode `run()`). Le thread appelant redevient alors activable.

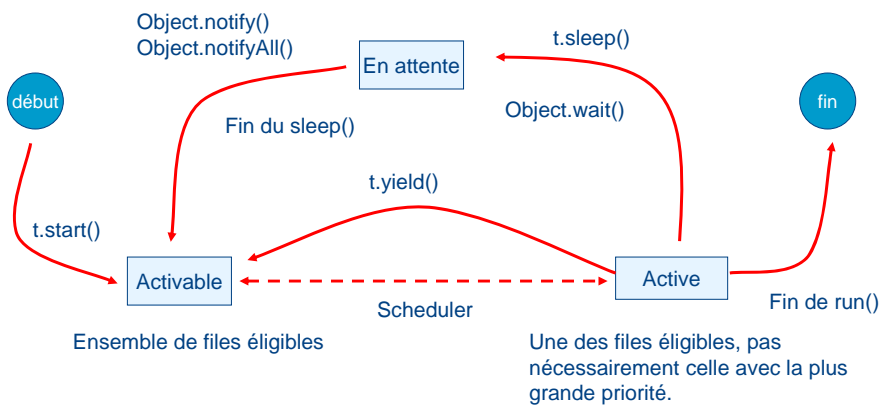
Gestion des Thread Méthodes de gestion (3/4)

```
public class ExJoin {
    public static void main(String[] arg){
        Thread t=new Thread(new FileSecondaire());
        t.start();
        for(int i=0;i<20;i++){
            System.out.println("File principale en cours d'exécution "+i);
            try{
                Thread.sleep(10);
            } catch(InterruptedException ie){}
        }
        try {
            t.join();
        } catch (InterruptedException ie) {}
        System.out.println("t termine son exécution, fin programme");
    }
}
```

Gestion des Thread Méthodes de gestion (4/4)

```
class FileSecondaire implements Runnable{
    public void run(){
        for(int i=0;i<40;i++){
            System.out.println("File secondaire en execution "+i);
            try{
                Thread.sleep(10);
            } catch (InterruptedException ie) {}
        }
        System.out.println("Fin de file secondaire");
    }
}
```

Gestion des Thread Diagrammes d'état



Exercice

EX 10.1

Paint (4/4)

- Créer une classe « Balle » héritant de « Thread »
- Définir ses variables et ses méthodes d'affichage *paint(Graphics g)* et *deplaceToi()*
- Redéfinir la méthode *run()* contenant une boucle infinie dans laquelle le déplacement de la balle est provoqué
- Instancier deux balles dans la zone graphique et lancer leurs *Thread* associés

- Implémenter l'interface « Runnable » dans la classe *ZoneGraphique* afin que l'écran soit automatiquement rafraîchi à intervalles réguliers.
- Redéfinir la méthode *run()* et lancer le *Thread* dans le constructeur de la classe

Introduction à Java Annexes: GUI, Multithreading & JDBC

C. Connexions aux bases de données avec JDBC

Connexion aux bases de données en Java

- Le défi
- La solution: JDBC
- Mise en œuvre

Connexion aux bases de données en Java

Le défi

- **Une application informatique a le plus souvent (surtout en informatique d'entreprise) besoin d'interagir avec une ou plusieurs bases de données**
- **Exemples:**
 - L'application qui permet à un agent bancaire d'effectuer des opérations sur le compte de ses clients doit obtenir et enregistrer des données sur la base de données de la banque
 - Les terminaux Bancontact ou Visa doivent envoyer une opération à la banque de données des comptes ou cartes et obtenir une confirmation
 - Un call center dans un service à la clientèle doit pouvoir obtenir et modifier les données des clients, de leurs factures, etc.

Connexion aux bases de données en Java

Le défi

- **Or les bases de données sont en général centralisées et servent un grand nombre de « clients »**
 - Comment stocker l'information à un seul endroit et permettre à de multiples utilisateurs d'y accéder en même temps et de différentes façons?
- **Les bases de données se contentent donc en général de stocker l'information, éventuellement de la valider, mais pas de la traiter**
- **Et elles se mettent à la disposition des applications pour leur fournir l'information demandée ou pour stocker leurs modifications (en SQL)**

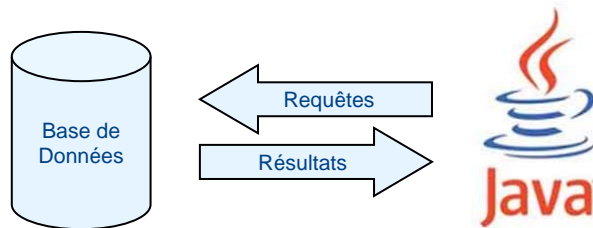
Connexion aux bases de données en Java

Le défi

- **Comment une application Java peut-elle interagir avec une base de données externe (par exemple MS Access)?**
 - Logiquement, en SQL
- **Mais comment permettre à l'application de se connecter à la base de données, de lui envoyer des requêtes et d'en récupérer le résultat?**
 - En rendant la base de données Access disponible pour interrogation par des applications externes (cf. procédure ODBC)
 - Et en établissant une connexion entre l'application et cette source ODBC

Connexion aux bases de données en Java

Le défi



Connexion aux bases de données en Java

La solution: JDBC

- **Dans le monde Windows, il existe une plateforme baptisée ODBC**
 - ODBC = Open DataBase Connectivity
- **Il s'agit d'une interface standard permettant de mettre des bases de données à disposition des applications Windows**
- **Chaque PC équipé de Windows possède une liste des "DataSources" ODBC qui pointent chacune vers une base de données**
- **ODBC "connaît" chaque type de base de données Windows et sait "dans quel langage" lui parler**
- **Autrement dit, il existe un "pilote ODBC" pour la plupart des bases de données qui tournent sous Windows.**
 - Microsoft Access en fait partie
- **Pour qu'un programme puisse accéder à une base de données, il suffit donc que celle-ci soit déclarée dans la liste des sources ODBC**

PROJET

- **Déclarez votre base de données MS Access comme une source de données ODBC**
 1. Ouvrez le menu « Démarrer » puis « Data Sources ODBC »
 2. Cliquez sur « Ajouter »
 3. Choisissez « Microsoft Access Driver (*.mdb) » dans la liste
 4. Cliquez sur Finish
 5. Donnez un nom (simple) à votre source de données
 6. Cliquez sur « Select... »
 7. Repérez et sélectionnez votre base de données et cliquez sur OK (Notez que votre BD pourrait très bien se trouver sur un autre ordinateur)
 8. Cliquez sur OK pour fermer le panneau de contrôle ODBC
- **Votre base de données MS Access est maintenant reconnue comme une source accessible via ODBC**

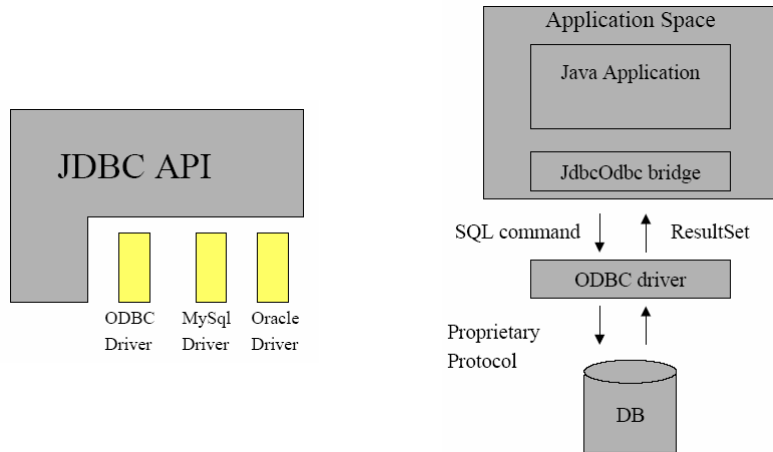
Connexion aux bases de données en Java

La solution: JDBC

- **Le problème: Java n'est pas une plateforme Windows**
 - Java n'est donc pas lié en soi à ODBC
 - Les applications Java doivent en effet pouvoir dialoguer avec n'importe quelle base de données
- **C'est pourquoi Java possède son propre système de communication avec les bases de données: JDBC (Java DataBase Connectivity)**
- **JDBC est un ensemble de classes prédéfinies pour chaque type de BD**
- **A l'instar d'ODBC, JDBC peut dialoguer avec un grand nombre de bases de données différentes (mais pas Access en tant que tel)**
- **En particulier, JDBC comprend un pilote qui lui permet de dialoguer avec ODBC**
- **Pour accéder à une base de données ODBC, un programme Java doit donc établir une connexion vers ODBC en précisant la DataSource dont il a besoin**
- **On parle en général d'un « Pont JDBC-ODBC »**

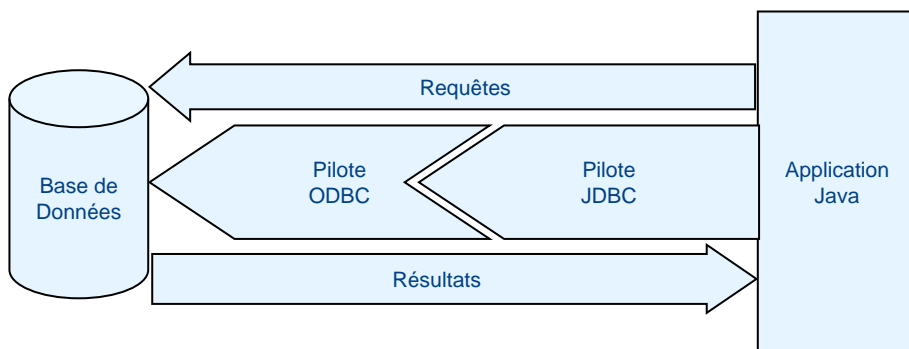
Connexion aux bases de données en Java

La solution: JDBC



Connexion aux bases de données en Java

La solution: JDBC



Connexion aux bases de données en Java

Mise en œuvre de JDBC

- **Pour établir une connexion JDBC, quelques lignes de code suffisent:**

- `public static final String jdbcdriver = "sun.jdbc.odbc.JdbcOdbcDriver";`
- `public static final String database = "jdbc:odbc:NomDeLaDataSourceODBC";`
- `private Connection conn;`
- `try{`
 - `Class.forName(jdbcdriver);`
 - `conn = DriverManager.getConnection(database);``} catch(Exception e) {`
 - `e.printStackTrace();``}`

- **Pour envoyer des requêtes à la BD une fois la connexion établie, il faut créer un « Statement »;**

- `Statement stmt = conn.createStatement();`

Connexion aux bases de données en Java

Mise en œuvre de JDBC

- **Une fois le Statement créé, on peut alors lui envoyer les requêtes SQL sous la forme de chaînes de caractères:**

- `String requete = "DELETE * FROM clients";`
- `stmt.executeUpdate(requete);`

- **Mais dans le cas de requêtes SELECT, il faut récupérer le résultat**

- **Les données issues de la requête sont rassemblées dans un jeu de résultats, appelé « ResultSet »:**

- `String requete = "SELECT * FROM client"`
- `ResultSet rs = stmt.executeQuery(requete);`

Connexion aux bases de données en Java

Mise en œuvre de JDBC

- On peut alors accéder aux enregistrements renvoyés un à un en demandant chaque fois au ResultSet de nous renvoyer la ligne suivante:

```
while(rs.next()){
    int c_id = rs.getInt("client_id");
    String c_n = rs.getString("client_name");
    String c_fn = rs.getString("client_firstname");
    String c_ad = rs.getString("client_address");
    String c_ph = rs.getString("client_phone");
    int c_ab = rs.getInt("client_abonne");
    double c_c = rs.getDouble("client_credit");
    int c_am = rs.getInt("client_anneemembre");
}
```

- Notez que l'on récupère la valeur de chaque champ en utilisant la méthode correspondant au type de données du champ (getInt, getString, getDate, etc.)

Connexion aux bases de données en Java

Mise en œuvre de JDBC

- Ce faisant, on a alors récupéré les données correspondant à chaque enregistrement
- Il nous reste donc à utiliser ces données pour créer un objet du type souhaité:

```
Client c = new Client(c_id,c_n,c_fn,c_ad,c_ph,c_c,c_am);
```

- Nous pouvons alors stocker ces clients un à un dans l'ArrayList:

```
clients.add(c); // A supposer que nous ayons une variable membre clients de
// type ArrayList<Client> dans la classe
}
```

- Il faut enfin toujours veiller à fermer les connexions:

```
rs.close();
stmt.close();
conn.close();
```

PROJET

1. Créez une nouvelle classe « `DataAccessObject` » qui sera chargée de communiquer avec la base de données. Le diagramme UML de cette classe vous est proposé dans l'énoncé (remplacez-y "`LocalServer`" par le nom de votre Source DSN ODBC)
2. Une fois cette nouvelle classe réalisée, vous remplacerez le code de votre ancienne méthode « `main` » (qui créait des objets artificiellement) par un code permettant de récupérer les objets de la classe « `DataAccessObject` » ci-dessus et vous demanderez à tous ces objets de s'afficher à l'écran sur la console. De cette façon, votre programme, en se lançant, chargera toutes les données de la BD et fera défiler tout son contenu sur la console. Vous y appellerez également une fois chacune des 4 méthodes transactionnelles (celles qui modifient des données).