

Les bases graphiques : la librairie « JFace » (suite)

LES BASES GRAPHIQUES : LA LIBRAIRIE « JFACE » (SUITE)	1
CONTENU DE LA SECTION	2
GENERALITES (1)	4
<i>les objectifs</i>	4
<i>quelques informations pratiques</i>	4
LES VIEWERS (1).....	5
<i>les objectifs et l'architecture</i>	5
<i>quelques méthodes génériques</i> :.....	5
LES VIEWERS (2).....	6
<i>L'interface IContentProvider</i>	6
LES TREEVIEWERS (1).....	7
<i>quelques caractéristiques de la classe TreeViewer</i>	7
<i>quelques méthodes de la classe TreeViewer</i>	7
LES TREEVIEWERS (2).....	8
<i>l'interface ITreeContentProvider</i>	8
<i>l'interface ILabelProvider</i>	8
LES TREEVIEWERS (3).....	9
<i>un exemple simple : la visualisation du système de fichiers</i>	9
LES TREEVIEWERS (3).....	10
<i>un exemple simple : la visualisation du système de fichiers (suite)</i>	10
<i>quelques compléments</i>	10
LES TABLEVIEWERS (1)	11
<i>quelques caractéristiques de la classe TableView</i>	11
<i>quelques méthodes de la classe TableView</i>	11
LES TABLEVIEWERS (2)	12
<i>l'interface IStructuredContentProvider</i>	12
<i>l'interface ITableLabelProvider</i>	12
LES TABLEVIEWERS (3).....	13
<i>un exemple simple : la visualisation d'un tableau de personnes</i>	13
LES TABLEVIEWERS (4)	14
<i>un exemple simple : la visualisation d'un tableau de personnes (suite)</i>	14
COMPLEMENTS SUR LES VIEWERS	15
<i>le filtrage</i>	15
<i>le tri</i>	15

développement de plugins Eclipse

L'EDITION DES CELLULES D'UN TABLEVIEWER (1).....	16
<i>l'objectif et les principes</i>	16
L'EDITION DES CELLULES D'UN TABLEVIEWER (2).....	17
<i>le modificateur</i>	17
L'EDITION DES CELLULES D'UN TABLEVIEWER (3).....	18
<i>quelques remarques</i>	18
COMPLEMENTS : LES DIALOGS JFACE (1).....	19
<i>les objectifs</i>	19
<i>quelques rappels</i>	19
<i>utilisation générique</i>	19
COMPLEMENTS : LES DIALOGS JFACE (2).....	20
<i>les InputDialog</i>	20
COMPLEMENTS : LES DIALOGS JFACE (3).....	21
<i>les AlertDialog</i>	21
COMPLEMENTS : LES DIALOGS JFACE (3).....	22
<i>un exemple simple</i>	22
COMPLEMENTS : QUELQUES DIALOGS ADDITIONNELS.....	23
<i>quelques autres Dialog</i>	23
COMPLEMENTS : LA FENETRE APPLICATIVE.....	24
<i>la classe ApplicationWindow</i>	24
<i>un exemple simple</i>	24

les objectifs

- librairie complémentaire à SWT
 - abstractions masquant les widgets SWT
 - abstractions MVC (Viewer)
 - widgets additionnels (Dialog JFace, Preference, ...)
 - meilleure utilisation des ressources (Action, Registry, ...)

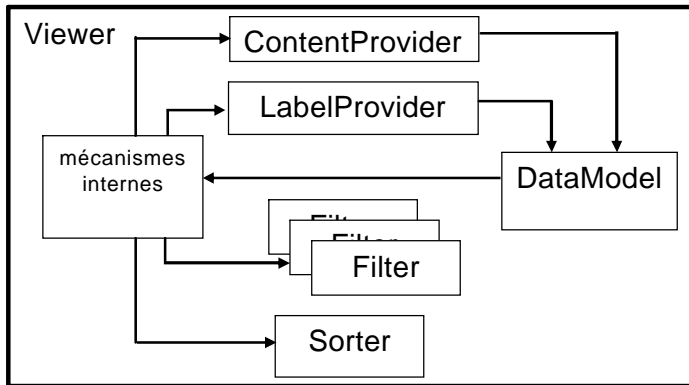
quelques informations pratiques

- pas de distribution bien packagée : → récupérer les .jar
 - jface.jar** dans **org.eclipse.jface_3.0.0**
 - runtime.jar** dans **org.eclipse.core.runtime_3.0.0**
 - osgi.jar** dans **org.eclipse.osgi_3.0.0**
 - jfacetext.jar** dans **org.eclipse.jface.text_3.0.0**
 - text.jar** dans **org.eclipse.text_3.0.0**

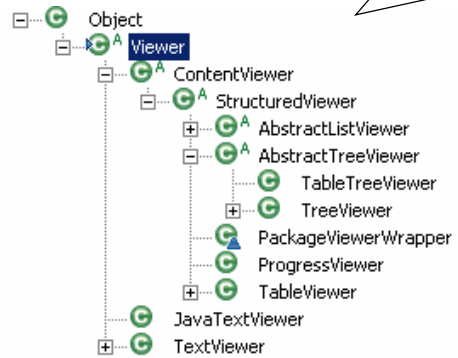
Les Viewers (1)

les objectifs et l'architecture

- séparation du modèle de données et du modèle graphique
- encapsulation des widgets et abstractions SWT (Tree, Table,, Sorter, Filter, ...)



implémentations de visualiseurs usuels



quelques méthodes génériques :

```
void setContentProvider(IContentProvider provider)
void setLabelProvider(ILabelProvider provider)
void setInput(Object input)
```

L'interface IContentProvider

- possède 2 méthodes
void inputChanged(Viewer viewer, Object oldInput, Object newInput) //notification d'un changement du modèle
void dispose()

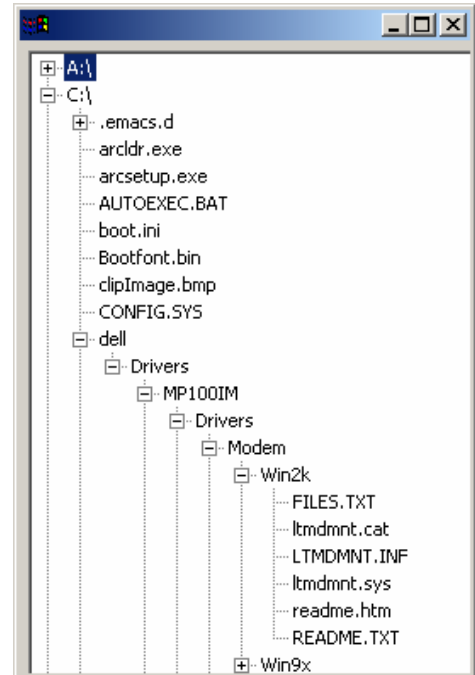
Les TreeViewers (1)

quelques caractéristiques de la classe TreeViewer

- **TreeViewer(Composite parent), TreeViewer(Composite parent, int style), TreeViewer(Tree aTree)**
- utilisation d'une interface spécifique **ITreeContentProvider**

quelques méthodes de la classe TreeViewer

Control getControl()
Tree getTree()



Les TreeViewers (2)

l'interface ITreeContentProvider

```

Object[ ] getChildren(Object node)           //retourne les descendants d'un nœud de l'arbre
Object[ ] getElements(Object input)         //retourne les nœuds racines de l'arbre
Object getParent(Object aNode)
boolean hasChildren(Object node)
void inputChanged(Viewer viewer, Object oldInput, Object newInput) //interface pour notifier un changement de modèle
void dispose()

```

ATTENTION : boucle sur la racine si :

```
Object[ ] getElements(Object input) { return new Object[ ] { input } ;
```

pas de problème si :

```
Object[ ] getElements(Object input) { return (Object[ ]) input; }
```

Avec invocation : `getElements(new Object[] { input });`

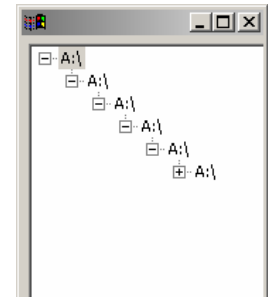
l'interface ILabelProvider

```

String getText(Object aElement)
Image getImage(Object aElement)
boolean isLabelProperty(Object aElement, String aProp)

void addListener(ILabelProviderListener aLn)
void removeListener(ILabelProviderListener aLn)

```



un exemple simple : la visualisation du système de fichiers

```
class FileViewerContentProvider implements ITreeContentProvider {
    public Object[] getChildren(Object aElem) { return ((File) aElem).listFiles(); }
    public Object getParent(Object aElem) { return ((File) aElem).getParentFile(); }

    public boolean hasChildren(Object aElem) {
        Object[] obj = getChildren(aElem);
        return obj != null && obj.length >0;
    }
    public Object[] getElements(Object inputElement) { return File.listRoots(); }
    public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {}
    public void dispose() {}
}

class FileViewerLabelProvider implements ILabelProvider {
    public Image getImage(Object aElem) { return null; }

    public String getText(Object aElem) {
        String txt = ((File) aElem).getName();
        if (txt.length() == 0) txt = ((File) aElem).getPath();
        return txt;
    }
    public void addListener(ILabelProviderListener listener) {}
    public void removeListener(ILabelProviderListener listener) {}
    public void dispose() {}
    public boolean isLabelProperty(Object element, String property) { return false; }
}
```

Les TreeViewers (3)

un exemple simple : la visualisation du système de fichiers (suite)

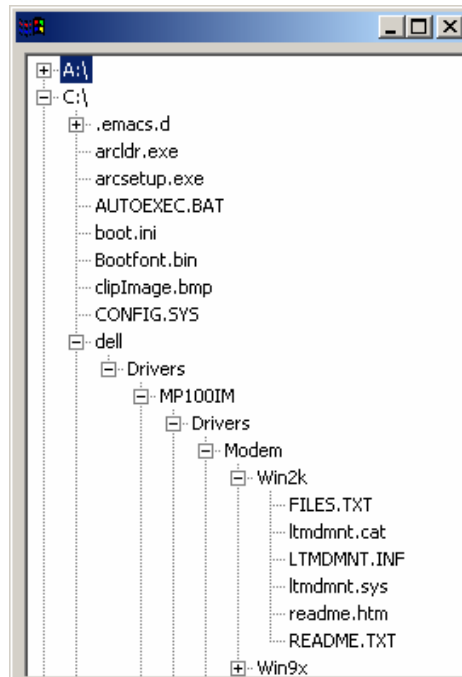
```
public static void main(String[] args) {
    .....
    shell.setLayout(new GridLayout(1, false));
    TreeViewer trV = new TreeViewer(shell);
    trV.setContentProvider(new FileViewerContentProvider());
    trV.setLabelProvider(new FileViewerLabelProvider());
    trV.setInput("Ignored");

    trV.getTree().setLayoutData(new GridData(400, 600));
    .....
}
```

argument nécessaire et nécessairement non nul, même si inutile.

quelques compléments

- possibilité de trier les nœuds (classe ViewerSorter)
- possibilité de filtrer les nœuds (classe ViewerFilter)
- nombreuses méthodes de contrôle de l'arbre (expand/collapse des nœuds, scrollUp/scrollDown)
- association de Listeners



les TableViewers (1)

quelques caractéristiques de la classe TableView

- **TableView(Composite aPar), TableView(Composite aPar, int aStyle), TableView(Table aTable)**
- utilisation d'une interface spécifique ITableLabelProvider

quelques méthodes de la classe TableView

- **Control getControl()**
- **Table getTable()**
- **int getItemHeight()**
- **int getHeaderHeight()**

l'interface IStructuredContentProvider

```
Object[ ] getElements(Object aInput)
void inputChanged(Viewer aViewer, Object oldInput, Object newInput)
void dispose()
```

l'interface ITableLabelProvider

```
String getColumnText(Object aElement, int aColIndex)
Image getColumnImage(Object aElement, , int aColIndex)
boolean isLabelProperty(Object aElement, String aProp)

void addListener(ITableLabelProviderListener aLn)
void removeListener(ITableLabelProviderListener aLn)
```

les TableViews (3)

un exemple simple : la visualisation d'un tableau de personnes

```
class PersonViewerContentProvider implements IStructuredContentProvider {
    public Object[] getElements(Object inputElement) { return ((List) inputElement).toArray(); }
    public void dispose() {}
    public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {}
}

class PersonViewerLabelProvider implements ITableLabelProvider {
    public Image getColumnImage(Object aElem, int alndex) { return null; }

    public String getColumnText(Object aElem, int alndex) {
        switch (alndex) {
            case 0 : return ((Person) aElem).getPrenom();
            case 1 : return ((Person) aElem).getNom();
            case 2 : return "" + ((Person) aElem).getPoids();
            case 3 : return "" + ((Person) aElem).getTaille();
            default : return null;
        }
    }

    public void addListener(ILabelProviderListener listener) {}
    public void removeListener(ILabelProviderListener listener) {}
    public void dispose() {}
    public boolean isLabelProperty(Object element, String property) { return false; }
}
```

```
class Person {
    private String prenom, nom;
    private int poids, taille;

    Person(String aPn, String aN, int aPd, int aT) {
        prenom = aPn;
        nom = aN;
        poids = aPd;
        taille = aT;
    }

    String getPrenom() { return prenom; }
    String getNom() { return nom; }
    int getPoids() { return poids; }
    int getTaille() { return taille; }
}
```

les TableViews (4)

un exemple simple : la visualisation d'un tableau de personnes (suite)

```

public Control createContents(Composite aCompo) {
    System.out.println("createContents");
    Table tbl = getTable();
    tbl.setLayoutData(new GridData(400, 300));
    TableColumn col1 = new TableColumn(tbl, SWT.NONE);
    col1.setText("prenom");
    col1.setWidth(100);
    TableColumn col2 = new TableColumn(tbl, SWT.NONE);
    col2.setText("nom");
    col2.setWidth(100);
    TableColumn col3 = new TableColumn(tbl, SWT.NONE);
    col3.setText("poids");
    col3.setWidth(50);
    TableColumn col4 = new TableColumn(tbl, SWT.NONE);
    col4.setText("taille");
    col4.setWidth(50);
    tbl.setHeaderVisible(true);
    tbl.setLinesVisible(true);
    return null;
}

```

prenom	nom	poids	taille
julien	Durand	75	178
amelie	Dupond	60	167
pierre	Martin	90	175

```

public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setLayout(new GridLayout(1, false));
    SimpleTableViewer tbV = new SimpleTableViewer(shell);
    tbV.createContents(shell);
    tbV.setContentProvider(new PersonViewerContentProvider());
    tbV.setLabelProvider(new PersonViewerLabelProvider());
    tbV.setInput(list);
    shell.pack();
    .....
}

```

le filtrage

- possibilité de filtrer les nœuds (classe ViewerFilter)
- définition de la méthode **boolean select(Viewer viewer, Object parentObject, Object element)**

```
class PersonFilter extends ViewerFilter {
    public boolean select(Viewer viewer, Object parentObject, Object element) {
        return ((Person) aElem).getPoids() > 20 && ((Person) aElem).getTaille() >120;
    }
}
```

le tri

- possibilité de trier les nœuds (classe viewerSorter)
- définition de la méthode **int compare(Viewer viewer, Object obj1, Object obj2)**

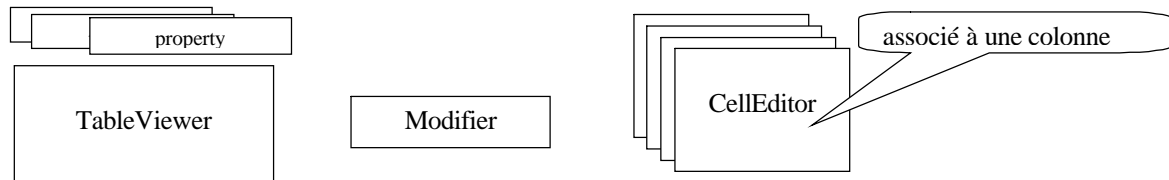
```
class PersonSorter1 extends ViewerSorter {
    .....

    public int compare(Viewer viewer, Object obj1, Object obj2) {
        switch(criterium) {
            case 0 : return ((Person) obj1).getNom().compareTo(((Person) obj2).getNom());
            case 1 : return ((Person) obj1).getPoids() - ((Person) obj2).getPoids();
            case 2 : : return ((Person) obj1).getTaille() - ((Person) obj2).getTaille();
            .....
        }
    }
}
```

L'édition des cellules d'un TableView (1)

l'objectif et les principes

- enrichir un Viewer (entité de visualisation) par des fonctions d'édition
- éditeurs prédéfinis : **TextCellEditor**, **CheckboxCellEditor**, **ComboCellEditor**, **ColorCellEditor**
- mise en oeuvre à l'aide d'éditeurs de cellule et d'un modificateur de cellule (qui établit le lien entre le viewer et l'éditeur)
- association de propriétés aux entités éditables



```

final String[] COLUMN_PROPERTIES = { "nom", "prenom" };
tableView.setColumnProperties(COLUMN_PROPERTIES);    //association de propriétés aux colonnes 0 et 1

tableView.setCellModifier(new PersonCellModifier() );    //association du modificateur (voir suite)

CellEditor[] editors = new CellEditor[2];           //definition des editeurs .....
editors[0] = new TextCellEditor(tableView.getTable()); //editeur de la colonne 0
editors[1] = new TextCellEditor(tableView.getTable()); //editeur de la colonne 1
tableView.setCellEditors(editors);                 //association des editeurs aux colonnes 0 et 1

```

L'édition des cellules d'un TableView (2)

le modificateur

- implémente la classe `ICellModifier`

```
boolean canModify(Object element, String property)
Object getValue(Object element, String property)
void modify(Object element, String property, Object value)
```

```
class PersonCellModifier implements ICellModifier {
    public boolean canModify(Object element, String property) { return true; }

    public Object getValue(Object elem, String prop) {
        Person pers;
        if (elem instanceof TableItem)
            pers = (Person) ((TableItem) elem).getData();
        else pers = (Person) element;

        switch(prop.charAt(0)) {
            case 'n' : return pers.getName();
            case 'p' : return ""+ pers.getFirstName();
        }
        return null;
    }
}

.....
public void modify(Object elem, String prop, Object value) {
    Person pers;
    if (elem instanceof TableItem)
        pers = (Person) ((TableItem) elem).getData();
    else pers = (Person) element;

    switch(prop.charAt(0)) {
        case 'n' : pers.setName((String) value); break;
        case 'p' : pers.First((String) value)); break;
    }
    tableViewer.refresh();
}
}
```

valeur retournée par un `CellEditor`

L'argument passé est parfois de type `Item`

L'édition des cellules d'un TableView (3)

quelques remarques

- Attention : type manipulé par :
 - **ComboBoxCellEditor** : Integer
 - **ColorCellEditor** : RGB
 - **TextCellEditor** : String
 - **BooleanCellEditor** : Boolean
- type transmis à **getValue** et **modify** est souvent **Item** (obtention de l'objet via **getData**)

Compléments : les Dialogs JFace (1)

les objectifs

- les Dialogs JFace ne remplacent PAS mais complètent les Dialogs SWT :
- **InputDialog, MessageDialog** : nouvelles classes
- **ProgressMonitorDialog, ErrorDialog** : classes fortement liées à Eclipse
- **TitleAreaDialog, IconAndMessageDialog** : classes de base pour la création de Dialog personnalisé

Les Dialogs WT sont :
**MessageBox, ColorSelectionDialog,
DirectorySelectionDialog,
FileSelectionDialog, PrintDialog**

quelques rappels

- s'utilisent globalement comme les Dialog SWT
- le parent d'un dialogue est toujours un shell
- méthode générique : **void open()**

utilisation générique

- comme les Dialog SWT :

```
<DialogType> dlg = new <DialogType>(shell) ;  
dlg.setSomeData(.....);  
<ReturnType> res = dlg.open();  
if (res == null) { /*l'utilisateur a cliqué le bouton CANCEL */  
} else { /*traiter la réponse de l'utilisateur */ }
```

Compléments : les Dialogs JFace (2)

les InputDialog

- forme particulière d'ErrorDialog pour la saisie texte
- retourne soit **Window.OK** soit **Window.CANCEL**
- possibilité d'associer un valideur (interface **IInputValidator**)

Retourne null si OK ou un message d'erreur à afficher

```
public class JFaceDialogExample0 {
```

```
    public static void main(String[] args) {
```

```
        Display display = new Display();
```

```
        final Shell shell = new Shell(display);
```

```
        shell.setLayout(new GridLayout(1, false));
```

```
        final Label label = new Label(shell, SWT.NONE);
```

```
        InputDialog dg = new InputDialog(shell, "", "entrer un texte de moins de 10 car.", "", new LValid());
```

```
        if (dg.open() != Window.OK) {
```

```
            label.setText("label ignoree !!");
```

```
        } else {
```

```
            label.setText(dg.getValue());
```

```
        }
```

```
        shell.pack();
```

```
        shell.open();
```

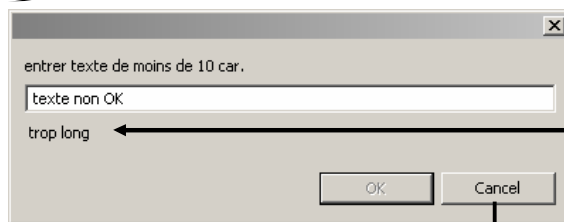
```
        while ( !shell.isDisposed() ) {
```

```
            if ( !display.readAndDispatch() ) display.sleep();
```

```
        }
```

```
        display.dispose();
```

```
    }
```



```
static class LValid implements IInputValidator {
    public String isValid(String input) {
        if (input.length(> 9) return "trop long";
        return null;
    }
}
```

les ErrorDialog

définis dans `org.eclipse.core.runtime`

- repose sur l'utilisation d'un **IStatus** (implémenté dans 2 classes eclipse : **Status** et **MultiStatus**)

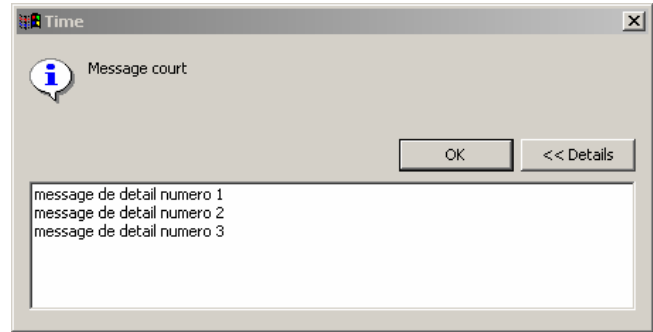
IStatus.CANCEL, IStatus.ERROR, IStatus.INFO, IStatus.OK, IStatus.Warning

- **Status(int severity, String pluginId, int code, String message, Throwable exception)**
-
- quelques méthodes de **IStatus**
IStatus[] getChildren()
int getCode(), Throwable getException(), String getMessage(), String getPlugin()
- l'introduction d'une zone "Detail" via un **MultiStatus**

les **ErrorDialog** sont utilisés pour l'envoi de messages quelconques si une zone Details est souhaité

un exemple simple

```
public class JFaceDialogExample1 {  
  
    public static void main(String[] args) {  
        Display display = new Display();  
        Shell shell = new Shell(display);  
        shell.setLayout(new GridLayout(2, false));  
  
        String[] messages = new String[] {  
            "Message court",  
            "message de detail numero 1",  
            "message de detail numero 2", "message de detail numero 3"  
        };  
  
        final String pluginId = "org.fho.tools.bidon";  
        MultiStatus info = new MultiStatus(pluginId, 1, messages[0], null);  
        for (int i = 1; i < 4; i++) info.add(new Status(IStatus.INFO, pluginId, 1, messages[i], null));  
        ErrorDialog.openError(shell, "Time", null, info);  
  
        shell.pack();  
        shell.open();  
        while (!shell.isDisposed()) { if (!display.readAndDispatch()) display.sleep(); }  
        display.dispose();  
    }  
}
```



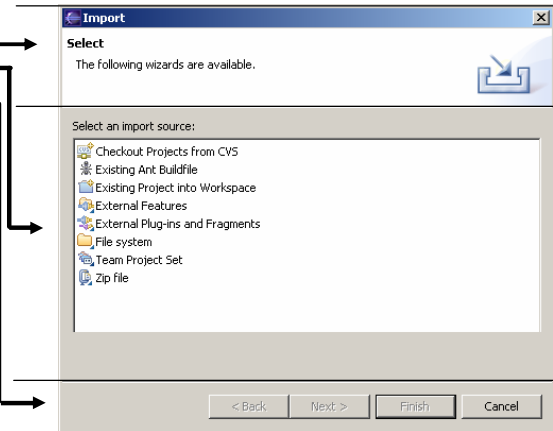
Compléments : quelques Dialogs additionnels

quelques autres Dialog

- les **TitleAreaDialog** (Dialog personnalisables)
sont formés de 3 zones :
une zone blanche (titre + image + message)
une zone grise de dialogue
une bande contenant les boutons

personnalisation par redéfinition des méthodes :
createDialogArea() et **createButtonBar()**

- les **IconAndMessageDialog** (Dialog personnalisables)
- les **ProgressMonitorDialog**
titre et message personnalisables + icône information
+ barre de progression + bouton Cancel



la classe `ApplicationWindow`

- une fenêtre applicative
- possède un shell comme parent (soit passé comme argument du ctor, soit créé automatiquement)
- les extensions doivent définir la méthode **`Control createContents(Composite parent)`**
- utilise **`ApplicationWindowLayout`**

un exemple simple

```
class JFaceExample extends ApplicationWindow() {
    public JfaceExample() { super(null) ; }
    public Control createContents(Composite parent) {
        Label lab = new Label(parent, SWT.CENTER);
        lab.setText("bonjour !!");
        return lab;
    }

    public static void main(String[] args) {
        JFaceExample aw = new JFaceExample();
        aw.setBlockOnOpen(true);
        aw.open();
        Display.getCurrent().dispose();
    }
}
```