

**Introduction to  
Borland JBuilder™ 2.00 Client/Server Suite**  
With DataGateway Enterprise

**Simple exercises - examples**



**Institution för Data- och systemvetenskap**

**Stockholms Universitet - KTH**

**nikos dimitrakas**

## Table of Contents

<b>1 INTRODUCTION.....</b>	<b>3</b>
<b>2 JBUILDER'S INTEGRATED DEVELOPMENT ENVIRONMENT (IDE).....</b>	<b>3</b>
2.1 STARTING JBUILDER .....	3
2.2 THE MAIN WINDOW .....	3
2.2.1 <i>The menu bar</i> .....	4
2.3 THE APPBROWSER .....	5
2.3.1 <i>The Content pane</i> .....	6
2.3.2 <i>The Navigation pane</i> .....	7
2.3.3 <i>The Structure pane</i> .....	7
2.3.4 <i>AppBrowser modes</i> .....	8
2.3.5 <i>Drilling down into other classes and interfaces</i> .....	9
2.3.6 <i>Browse Symbol at Cursor</i> .....	10
2.3.7 <i>Removing tabs from the AppBrowser</i> .....	10
2.4 THE OBJECT GALLERY .....	10
2.5 WIZARDS .....	12
2.6 USING THE HELP VIEWER.....	12
<b>3 CREATING AND MANAGING PROJECTS.....</b>	<b>13</b>
3.1 WHAT IS A PROJECT? .....	13
3.2 DISPLAYING A PROJECT .....	13
3.3 CREATING A NEW PROJECT .....	14
3.3.1 <i>Creating a new project with the Project Wizard</i> .....	14
3.4 OPENING AN EXISTING PROJECT .....	14
<b>4 BASIC EXERCISES .....</b>	<b>15</b>
4.1 USING SIMPLE GUI CONTROLS .....	15
4.2 USING ADDITIONAL CONTROLS.....	24
4.3 USING CONTAINERS AND LAYOUTS.....	27
4.3.1 <i>Further practice</i> .....	33
4.4 USING DIALOGS AND MENUS .....	33
4.5 WORKING WITH FRAMES .....	40
4.5.1 <i>Further practice</i> .....	42
<b>5 INTRODUCTION TO DATAGATEWAY.....</b>	<b>42</b>
<b>6 DATABASE EXERCISES .....</b>	<b>42</b>
6.1 MAKING A DATABASE AVAILABLE .....	43
6.1.1 <i>Through DataGateway</i> .....	43
6.1.2 <i>Through ODBC</i> .....	44
6.2 SHOW AND UPDATE A SINGLE TABLE .....	44
6.3 MASTER-DETAIL .....	49
6.4 FILTERING DATA - CALCULATED FIELDS .....	52
6.5 USING PARAMETERIZED QUERIES .....	55
<b>7 EPILOG .....</b>	<b>57</b>

# 1 Introduction

This compendium is designed for people that have prior knowledge of programming and Java. It does not require on the other hand knowledge of Rapid Application Development (RAD) tools. This compendium is not covering JBuilder to the full extend. The goal is to introduce to the reader to basic concepts of Rapid Application Development, make the reader familiar with working in an Integrated Development Environment, and even to present basic features of JBuilder as well as database development possibilities that JBuilder offers<sup>1</sup>. In addition to all that we will briefly present how to create applets with JBuilder.

## 2 JBuilder's Integrated Development Environment (IDE)

This chapter introduces you to the JBuilder integrated development environment (IDE).

This chapter provides you with overviews of:

- **Starting JBuilder**  
Describes what you see when you open JBuilder.
- **The main window**  
Describes the parts of JBuilder's main window, including the menu bar, the toolbar, and the Component Palette.
- **The AppBrowser**  
Describes the AppBrowser and the AppBrowser modes, including the Project Browser mode, the Opened Files Browser mode, and the Directory Browser mode. Describes the AppBrowser's Navigation pane, Content pane, and Structure pane. Explains how to navigate your .java file and drill down into ancestor classes.
- **The Object Gallery**  
Describes the Object Gallery, a repository of shortcuts that create skeletal instances of many objects.
- **Wizards**  
Provides a brief description what each wizard accomplishes.
- **Using Help**  
Describes the JBuilder Help Viewer and the different ways to search in the help files

### 2.1 Starting JBuilder

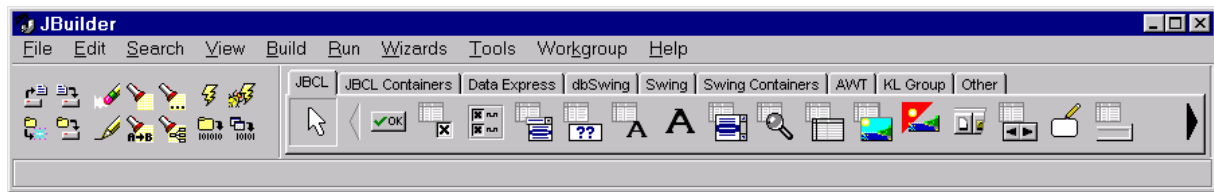
When you first open JBuilder, you see JBuilder's main window and AppBrowser.

### 2.2 The main window

The main window is at the top of the screen when you open JBuilder. It contains the menu bar, the toolbar and the Component Palette.

---

<sup>1</sup> JBuilder can work with databases in different ways. ODBC is the standard driver that is normally available, but it gives limited possibilities. DataGateway is a collection of JDBC drivers that allow Java applications (and applets) to access databases of various types. In this compendium DataGateway is presented briefly in order to establish the database connections needed for the exercises.



- The menu bar is where you choose menu commands.
- The toolbar displays buttons that are shortcuts for commonly performed tasks. Place the mouse pointer over a toolbar button, without clicking, to see a description of what the button does.
- The Component Palette displays components available in the JBuilder's component library. See Components delivered on the Palette for more information.
- The status bar displays compilation progress and file save messages.

### 2.2.1 The menu bar

The menu bar is at the top of the main window. You can select a menu command, then press F1 to display context-sensitive help for that command.

The following table provides brief descriptions of menu commands.

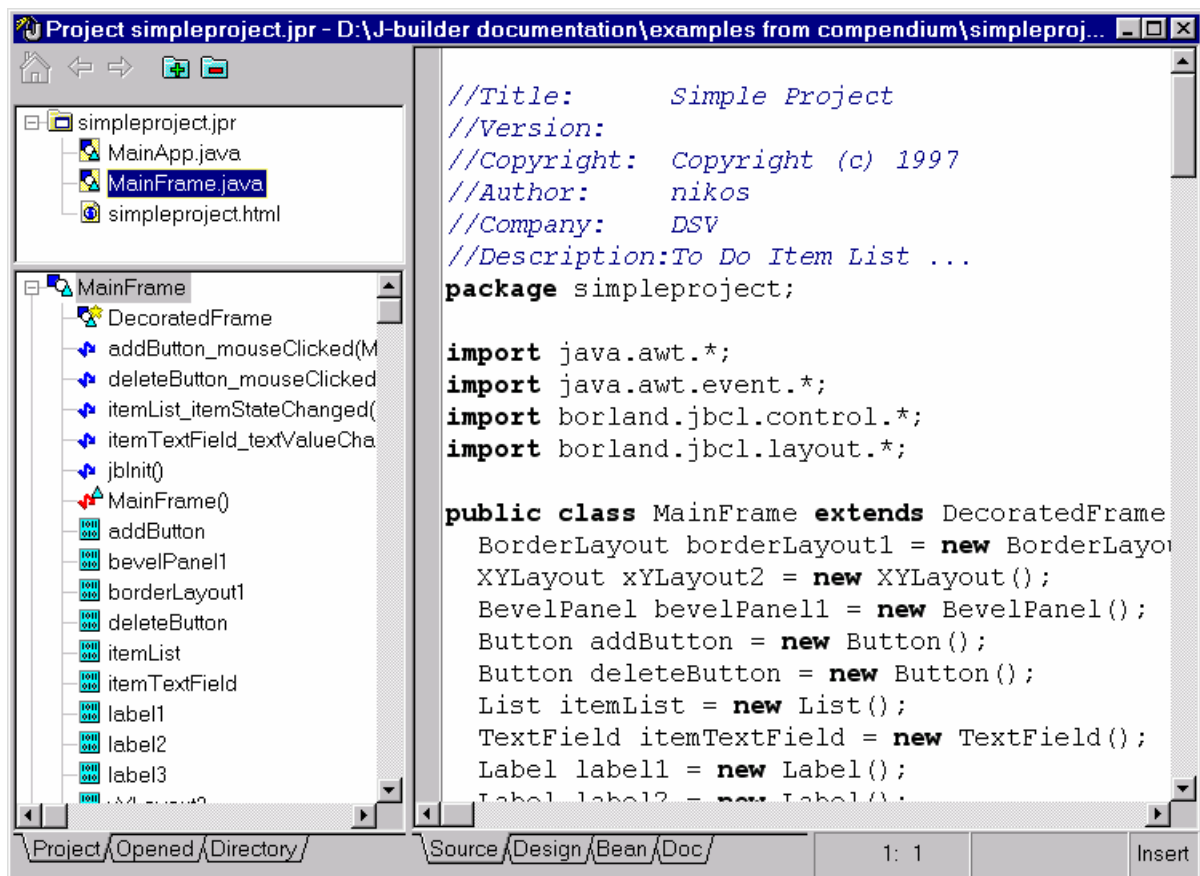
Menu	Commands for...
File menu	Creating, opening, closing, renaming and saving files and projects; removing files from projects; configuring printers; printing files.
Edit menu	Copying, pasting, deleting and selecting text; undoing and redoing actions.
Search menu	Finding and replacing text; searching for text incrementally and by line number; searching for text across a source path; searching for a symbol.
View menu	Viewing Debugger windows, a new AppBrowser, the next or previous error message, the toolbar, the Component Palette, or other currently open AppBrowsers.
Build menu	Making or building the selected node.
Run menu	Running the application or applet; stepping over or tracing into code; running to the end of a selected method; pausing the program; setting watches or breakpoints; inspecting, evaluating and modifying.
Wizards menu	Running utility wizards for tasks such as implementing an interface, overriding a method, bundling resources, and wrapping an applet.
Tools menu	Displaying the Environment Options dialog; invoking the Windows Notepad and Calculator; invoking remote object creation tools; and opening database tools.
Workgroup menu	Setting up version control and managing workflow.
Help menu	Displaying documentation, such as the Help system, the BeansExpress tutorial, the JDK API Reference, and the JBCL Reference. Also for viewing the Inprise Online web site in your default web browser, loading the Welcome project for experimenting, and seeing information about this

release of JBuilder.

## 2.3 The AppBrowser

JBuilder is structured to increase your productivity as a Java developer. Because Java projects use many files, and because the various development tasks (such as editing, debugging, and browsing for information) have traditionally used multiple windows, it can be difficult to find the window you need.

To simplify your job, JBuilder introduces a new concept in user interfaces for development environments: a single AppBrowser that is used to perform all the usual development functions. The JBuilder AppBrowser lets you to explore, edit, design, and debug all in one unified window.



The JBuilder AppBrowser usually fills the screen area below the main window. If you are running JBuilder, and don't have an AppBrowser displayed, choose Help|Welcome Project now to view it.

The AppBrowser usually contains three panes:

- The Navigation pane, on the upper left
- The Content pane, on the right side
- The Structure pane on the bottom left
- The Inspector on the right, visible only in specific modes.

Although you will usually have only one AppBrowser visible, you can open additional AppBrowsers as needed, for example, to view more than one project at a time. Use View|Windows to switch between AppBrowsers.

### 2.3.1 The Content pane

The Content pane displays the detailed content of the file selected in the Navigation pane. The editor or viewer used is determined by the file's extension.

File Type	Editor(s) or Viewer(s) available in the Content pane
Text files	If you select a text file in the Navigation pane (a file with an extension such as .txt or .bat), a single editor, identified by the Source tab, is available in the Content pane.
Image files	If you select a .GIF, .JPG, or .BMP image file in the Navigation pane, an image viewer, identified by the View tab, is available in the Content pane.
HTML files	<p>If you select an HTML file in the Navigation pane, two tabs are displayed at the bottom of the Content pane, labeled View and Source.</p> <p>View tab The View tab selects an HTML viewer. This viewer lets you to see the rendered HTML file, as you would see it in a web browser.</p> <p>Source tab The Source tab selects an Editor that lets you see and edit the file as raw HTML source.</p>
.java files	<p>If you select a .java file in the Navigation pane, you will see three tabs labeled Source, Doc, and Design.</p> <p>Source tab If you select the Source tab when viewing a .java file, you will see the JBuilder Java Source Code Editor. This is a full-featured, syntax-highlighted programming editor.</p> <p>Doc tab If you select the Doc tab when viewing a .java file, you will see the corresponding reference doc for that .java file, if there is one.</p> <p>Design tab If you select the Design tab when viewing a .java file, you will see the JBuilder visual design tools for that class. For example, if you select the WelcomeFrame.java class in the Welcome project (or a Frame class in your own project), you will see the Jbuilder UI Designer in the Content pane.</p> <p>Bean tab When you select the Bean tab, you see the BeansExpress designers. The Bean tab exposes the BeansExpress Property, Event, BeanInfo, and Property Editor designers. Use them to add properties and events to your bean, choose what properties are exposed, and create custom property editors.</p>

You can expand the Content pane to fill the entire AppBrowser window. You simply toggle it in and out of full window mode with the View|Toggle Curtain menu command. You can also use the mouse to resize the window or any of its panes by dragging the pane boundaries.

Note that when you activate the design tab of the content pane, the Inspector appears on the right side of the content pane. The Inspector shows common properties and events for the selected bean (component). It is also possible to mark two or more components at once, then the Inspector shows only the common properties for the marked components.

Some properties can be changed manually (When changing properties on the Inspector, it is recommendable to use the Enter key. Not using the Enter key may sometimes lead to inconsistencies between the code and what is shown on the Inspector.), while others can be changed only by JBuilder itself.

On the events tab things work differently. First you have to click once on the event you want to edit.



Then you can either write the name of the event handler you want to call when that event occurs, or double click on the field to the right of the event name. When you've done that the source tab of the content pane is activated and a method header is created. The name of this method is automatically entered in the event handler field of the chosen event. In this method you can write your Java code to be executed when this event occurs.

### 2.3.2 The Navigation pane

The Navigation pane is the top left pane of the AppBrowser. This pane shows a list of one or more files. In the case of the Project Browser, you will see the project (.jpr) file first. Attached to that is a list of the files in the project. The list may include .java, .html, text, or image files.

You can select a file in the Navigation pane by clicking it. The Content pane and the Structure pane display information about the selected file. As you select different files in the Navigation pane, each one will be represented in the Content and Structure panes.

### 2.3.3 The Structure pane

The lower left pane of the AppBrowser shows you a structural analysis of the file that you have selected in the Navigation pane.

For example, if you select a .java file, the Structure pane will show you structural information about the java code in that file, such as

- Imported packages.
- The classes and/or interfaces in the file.
- Any ancestor classes and/or interfaces.
- Variables and methods.

This structural analysis is in the form of a hierarchical tree. You can think of the Structure pane as a table of contents for the file.

### ***2.3.3.1 Navigating .java files using the Structure pane***

Not only does the Structure pane show you the structure of your file, you can also use it as a quick navigation tool to the various structural elements in the file. For example, if you have selected a .java file, you see classes, variables, and methods in the Structure pane. You can then click on any of those elements in the Structure pane and the Content pane will move to and highlight that element in the source code. You can also use the Structure pane for drilling down into other ancestor classes and interfaces.

### ***2.3.3.2 The Component Tree***

When you have selected a .java file and then select the Design tab at the bottom of the Content pane, the Structure pane will display the designable objects in the file, and how they are nested and interrelated. This view is called the Component Tree. For more information, see Using the Component Tree.

## **2.3.4 AppBrowser modes**

The AppBrowser has two sets of tabs to control the type of view you see. One set is at the bottom of the Structure pane and the other set is at the bottom of the Content pane. Different tabs appear, depending on what you are doing.

- The tabs below the Structure pane control which mode the browser is in (what kind of browser it is). Examples are Project Browser mode (Project tab), Opened Files Browser mode (Opened tab), and Directory Browser mode (Directory tab).
- The tabs below the Content pane control the kind of viewer or editor used in the Content pane. Examples are the Editor (Source tab), the UI Designer (Design tab), and Documentation Viewer (Doc tab). Other available viewing modes of the Content pane are HTML Browser (View tab) and Image Viewer (Viewer tab).

The browser as a whole is called the AppBrowser. When you switch the AppBrowser into a different mode, such as Project mode, you can think of it as the Project Browser.

The AppBrowser can display the following modes:

- Project Browser
- Directory Browser
- Opened Files Browser
- Debugger
- Class Hierarchy Browser
- Search Results Browser

To return to the starting state and see your code, click the Project tab, then the Home button.

### ***2.3.4.1 Project Browser***

The Project Browser mode of the AppBrowser manipulates files in a project. To put the AppBrowser into Project Browser mode, choose the Project tab on the lower left tabset, below the Structure pane. In this mode, the Navigation pane shows the project (.jpr) file and a list of the files in that project.



If you find a file you want to open, select it in the Navigation pane. File details are displayed in the Content pane and file structure is displayed in the Structure pane.

#### ***2.3.4.2 Directory Browser***

If you select the Directory tab below the Structure pane, you will switch the AppBrowser from browsing your project to browsing your file system directory. In this mode, you can browse through directories to locate files.

If you find a file you want to open, select it in the Navigation pane. File details are displayed in the Content pane. File structure is displayed in the Structure pane just as if it were a file in your project.

You can quickly add a file to your project from the Directory Browser by clicking it and dragging it to the Project tab.

#### ***2.3.4.3 Opened Files Browser***

The Opened tab gives you a list of your currently active or open files. Your open files are

- Files you have edited in this AppBrowser session.
- Files you have explicitly dropped onto the Opened tab from the Directory or Project tabs.

#### ***2.3.4.4 Debugger***

When you run the JBuilder Debugger, it will add two more tabs to the AppBrowser, Debug and Watch. The Debug tab displays the main Debugger view and the Watch tab displays the Watch view.

While debugging, you can switch to any of the other tabs to review files in your project, browse directories, and look up reference information without disturbing your debugging session or cluttering your screen.

#### ***2.3.4.5 Class Hierarchy Browser***

To see the class hierarchy for a particular .java file, select it in the Navigation pane, right-click, and from the popup menu choose Class Hierarchy. This will add a new AppBrowser tab for Class Hierarchy viewing. When you click on this tab, the Navigation pane will show you the inheritance tree for your file. You can then navigate to any file in the inheritance tree by selecting it in the Navigation pane.

#### ***2.3.4.6 Search Results Browser***

If you use Search|Search Source Path to look for a search string across directories, JBuilder adds a Search Results tab to the AppBrowser. When you click this tab, you will see a list of the file(s) that contain the selected text.

### **2.3.5 Drilling down into other classes and interfaces**

Often you will need to look at .java files that are not part of your project, but which are referred to in the class you are editing. This might be an ancestor class or the class of some instance variable.

There are several traditional ways you can navigate to the file you need. If you know its full package and class name, you can

- Open it in a separate browser (File|Open/Create)
- Browse for it using Search|Browse Symbol
- Use the Directory Browser to look for it on the disk

With the Structure pane, JBuilder provides a much faster way. To see the .java file for an ancestor class, an interface, or the type of a variable shown in the Structure pane, just double-click it. The AppBrowser will go to that file, showing it in all three panes.

Follow these steps for an example of drilling down:

- Use Help|Welcome Project to open the Welcome Project.
- Select WelcomeFrame.java in the Navigation pane.
- In the Structure pane, note that WelcomeFrame extends DecoratedFrame.
- Double-click DecoratedFrame in the Structure pane.
- The AppBrowser will show you DecoratedFrame.
- Double-click Frame to drill down another level into the Frame parent class of DecoratedFrame. Note that you can not only see the source of the Frame class, but you can also click on the Doc tab and read the reference doc for the class as well.
- If you navigate to a file for which no source code is found, JBuilder will synthesize a temporary source file to show in the Source pane. This temporary file contains method stubs for the public methods.

### **2.3.6 Browse Symbol at Cursor**

In addition to drilling down into ancestor or variable classes in the Structure pane, you can also use the popup menu in the Source view of a .java file to browse to a symbol in the source.

- Place your cursor on the symbol (interface or class name) in the source.
- Right-click.
- Choose Browse Symbol at Cursor.
- JBuilder will locate the file for that Java class or interface and show it in the AppBrowser.

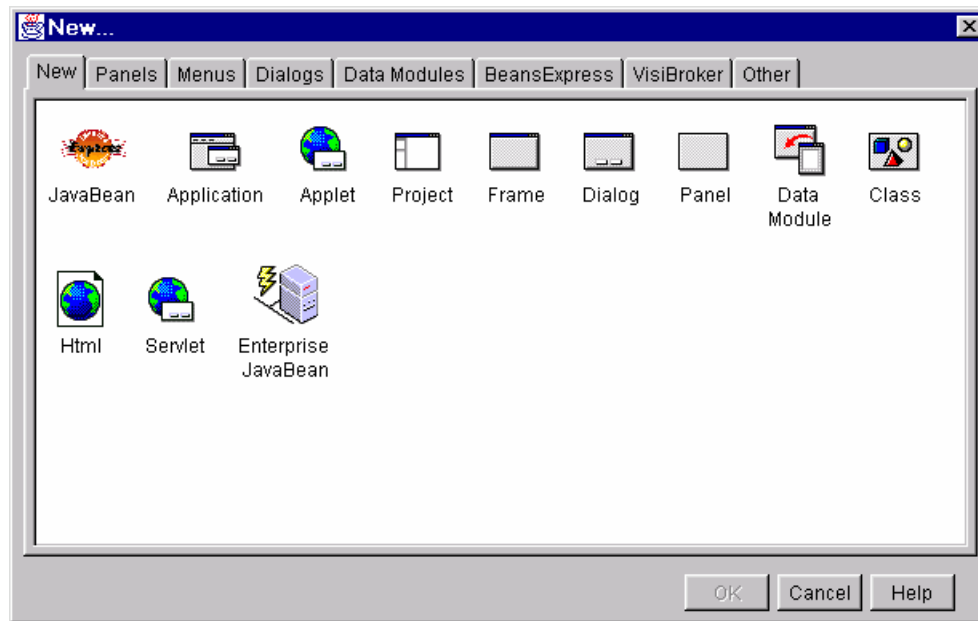
### **2.3.7 Removing tabs from the AppBrowser**

To remove a tab, such as Search Results or Class Hierarchy, that has been added to the AppBrowser,

- Right-click it.
- Choose the Drop command for the name of the tab you want to drop.
- JBuilder removes the tab.

## **2.4 The Object Gallery**

The Object Gallery contains shortcuts that create skeletal instances of many objects. To display the Object Gallery, choose File|New.



To use a shortcut or template, click the icon. JBuilder creates the skeletal code in a .java file and adds the file to your project.

- The New page contains shortcuts for creating applications, applets, projects, frames, dialogs, panels, data modules, classes and HTML files.
- The Panels page contains shortcuts for creating
  - A tabbed Pages panel, a multi-paged dialog box with three tabbed pages
  - A dual list box with a Source and Destination list
- The Menus page contains a shortcut for creating a standard menu, which includes a File, Edit and Help menu. Each menu contains standard menu items.
- The Dialogs page contains shortcuts for creating:
  - An About Box
  - Two standard dialogs, each with a BevelPanel and an OK and Cancel button
  - A Password dialog
- The DataModule page contains a shortcut for creating an employee database.
- The BeansExpress page contains shortcuts for creating JavaBeans.
- The VisiBroker page contains a shortcut for creating a CORBA server.
- The Other page contains a shortcut for creating an example snippet.

To add your own files to the Object Gallery:

- Select the page of the Object Gallery on which you want the file to appear.
- Right-click and choose Add Snippet.
- The Add Snippets dialog is displayed.
- In the Object Gallery description field, enter a brief description of the snippet.
- Specify the name of the snippet file.
- Specify the name of the image to display in the Object Gallery and click OK.

## 2.5 Wizards

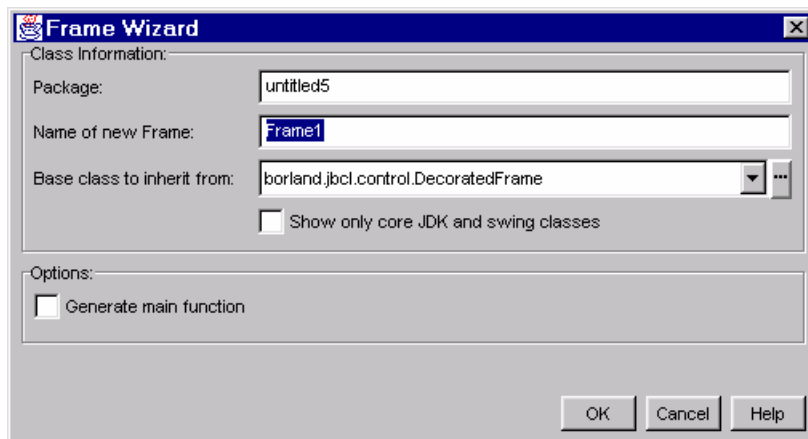
To guide you through the major tasks necessary to create Java programs, JBuilder comes equipped with wizards. These are tools that guide you through the necessary steps for a specific function or task.

There are three types of wizards:

- File wizards are wizards that create new files. These wizards are available in the Object Gallery.
- Utility wizards modify existing files and are available on the Wizards menu.
- Remote Objects wizards are found on the Tools and Wizards menus and in the Object Gallery.

Example:

New Frame Wizard - creates a new frame:



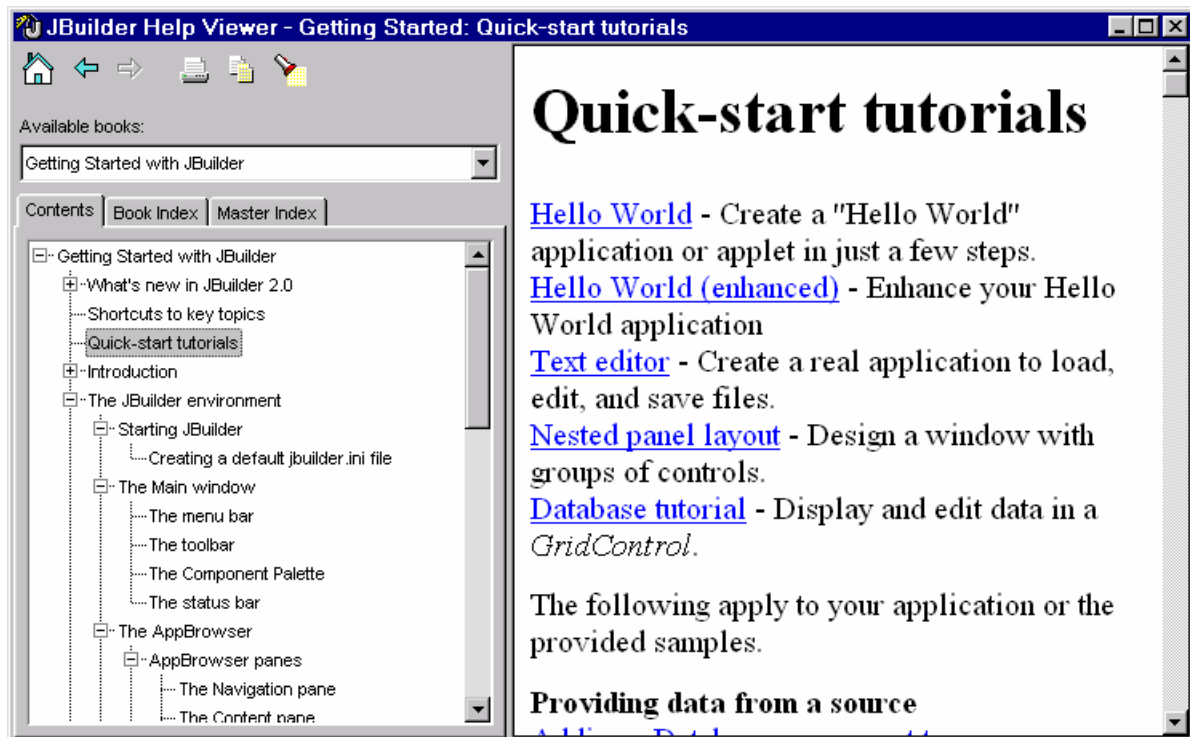
## 2.6 Using the Help Viewer

The Help Viewer provides access to various books with helpful information and examples about:

- JBuilder
- Java
- DataGateway
- InterClient
- Other useful topics concerning development with JBuilder

The Help Viewer is very much like the AppBrowser. There is the navigation pane, the content pane and some navigation buttons. The navigation pane has three modes (tabs):

- Contents This option allows you to see the contents of the book selected in the “available books” drop down list. Clicking on a chapter in the navigation pane shows this chapter in the content pane of the viewer.
- Book Index Here you can search for a particular title in a specific book.
- Master Index Here you can search for a particular title in all the books at once.



You can at all times search for a certain word, by clicking on the search icon, but this searches only in the currently shown document.  
By using the next and previous buttons you can easily go back and forth among the different topics that you have visited.

## 3 Creating and managing projects

To develop programs in the JBuilder environment, you must first create a project.

### 3.1 What is a project?

In its simplest form, a project is just a holder of files. Generally, a project is used to "hold" the files that together make up a JBuilder application or applet. These files can be in any directory. The project ties them all together.

The information about each JBuilder project is stored in a project file that has a .jpr file extension. This project file contains a list of all the files in the project and the project settings and properties. JBuilder uses the information in the project file when you load, save or build a project and sometimes when you use a wizard. You don't edit a project file directly, but it is modified whenever you use the JBuilder development environment to add and remove files and set project options. You can see the project file as a node at the top of the project tree in the Navigation pane of the AppBrowser.

### 3.2 Displaying a project

JBuilder displays each project within its own AppBrowser. Therefore, if you have more than one project open, JBuilder opens an AppBrowser for each.

### **3.3 Creating a new project**

JBuilder gives you several ways for starting a new project. The method you choose to begin a project depends on what your intended goal is for the project.

You might want to start a new project by creating only the project file and adding files later. Or if you intend to design an application or applet, you can use one of JBuilder's wizards to automatically generate the framework for your application or applet. The wizard automatically generates a project file and imports the basic files if no other project is already open.

#### **3.3.1 Creating a new project with the Project Wizard**

JBuilder provides the Project Wizard to assist in creating a new project. The Project Wizard automatically sets up the framework for the project and gives you the opportunity to enter informative data about the project such as its location on disk, the author of the project, and a description.

The Application and Applet Wizards both have a shortcut to the Project Wizard. If either one of those wizards is launched when a project is not open, the Project Wizard will be launched first and a new project will be created. Then, control returns to the wizard that was first invoked. The new application or applet will automatically be added to the newly created project. (Of course, the Application and Applet Wizards can be run at any time during the development of a project and the appropriate files will be added to the project).

To create a new project with the Project Wizard,

- Choose File|New Project... (or File|New... and then Project)
- The Project Wizard is displayed.
- Enter the name of the project file. Make sure the file extension is .jpr.
- Fill in the title, author, company, and description fields. These fields are optional.
- Click Finish when you're done.
- The new project is displayed in its own AppBrowser.

### **3.4 Opening an existing project**

There are three ways to open an existing project. You can either use the File|Open/Create dialog box, the File|Reopen menu command, or the Directory Browser.

To open a project using the File|Open/Create dialog box,

- Choose File|Open/Create to display the File Open/Create dialog box.
- In the Files of Type box, select Java Projects to display only JBuilder project files.
- Navigate to the directory that contains the file to be imported.
- Select the .jpr file you want to open.
- Choose Open.

To open a project with the File|Reopen command,

- Choose File|Reopen. A list of previously opened projects is displayed.
- Choose the project you want to open.

To open a project using the Directory Browser,

- Choose the Directory tab. The Directory Browser appears in the AppBrowser.
- In the Directory Browser, double-click directories to navigate to the directory that holds the project you want to open.
- Double-click the project file. An AppBrowser appears, displaying the new project node in the Project Browser.

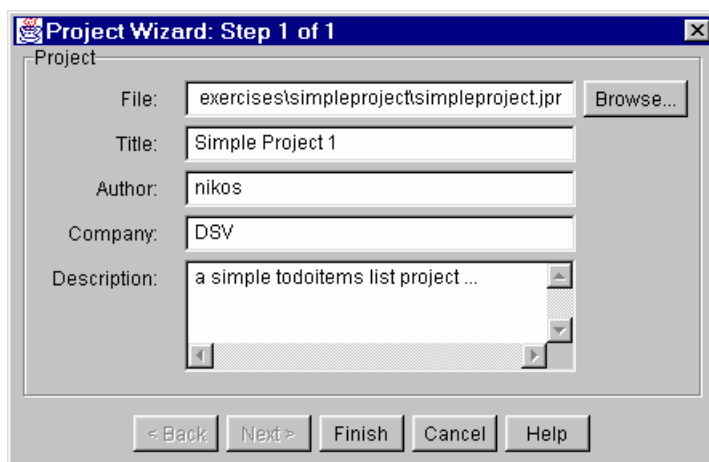
## 4 Basic exercises

The exercises that follow concentrate on common components that JBuilder provides. We will notice that actions that have been done in an exercise are often not described in detail in a later exercise. If you feel that you don't know how to do something in one of the later exercises, you can then go back to an earlier exercise, where this something was described in detail.

### 4.1 Using simple GUI controls

This exercise concentrates mostly in using components found under the AWT (Abstract Windowing Toolkit<sup>2</sup>) tab on the Component Palette. In this exercise we will create a simple project. We will let the user create a list with items to do. The user will be able to add items to and remove items from the list. To do that we need to create a project, add a frame to this project, add a few GUI controls and implement some functionality to these controls by writing some Java code. To do that, follow the following steps:

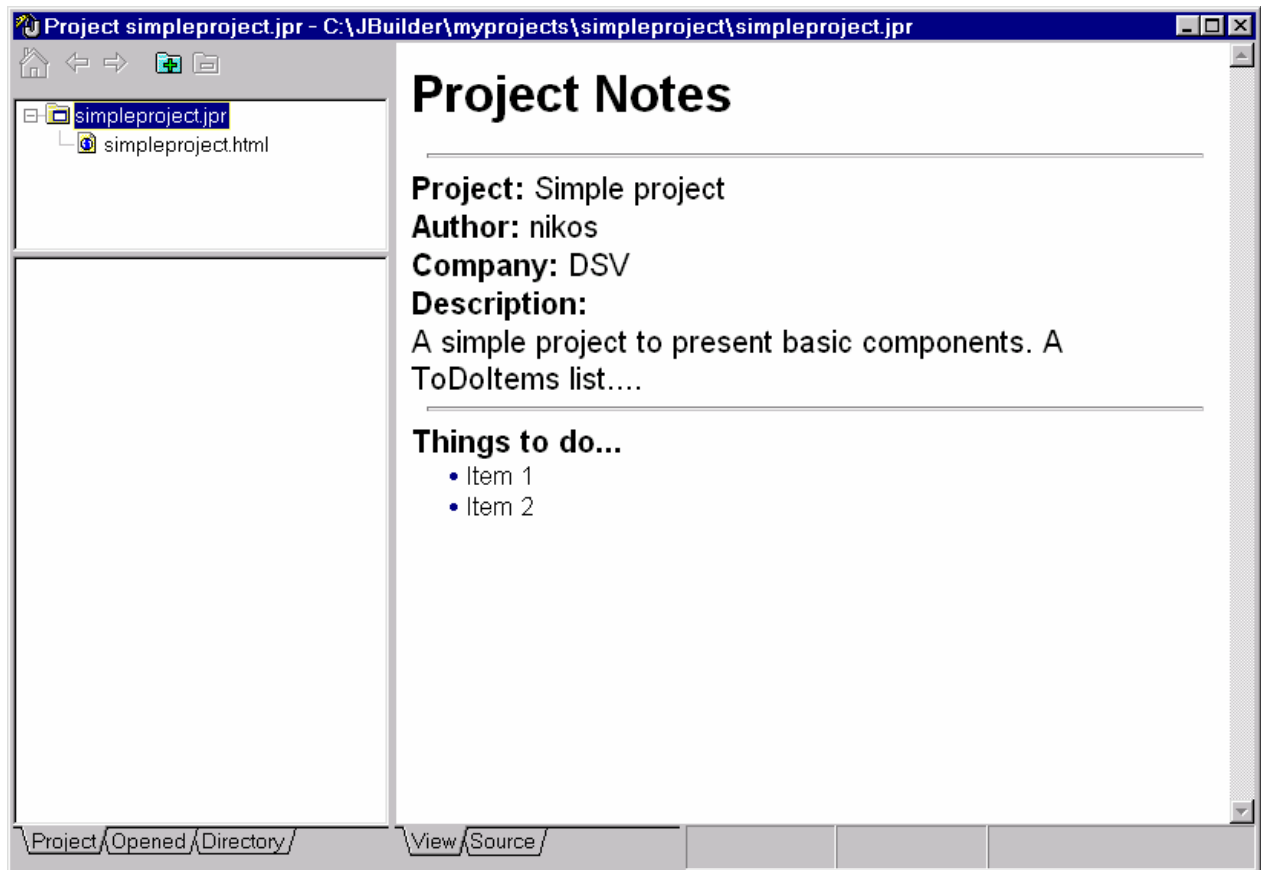
- Start JBuilder and close all open projects. (Only the main window should be visible)
- Start a new project
- When the Project wizard appears set the file field to:  
*Your home directory*\JBuilder exercises\simpleproject\simpleproject.jpr  
(You can also fill the other fields if you want)



<sup>2</sup> The java.awt package contains components that are simple GUI controls such as checkboxes, labels, radio buttons, and text boxes.

- Click the finish button

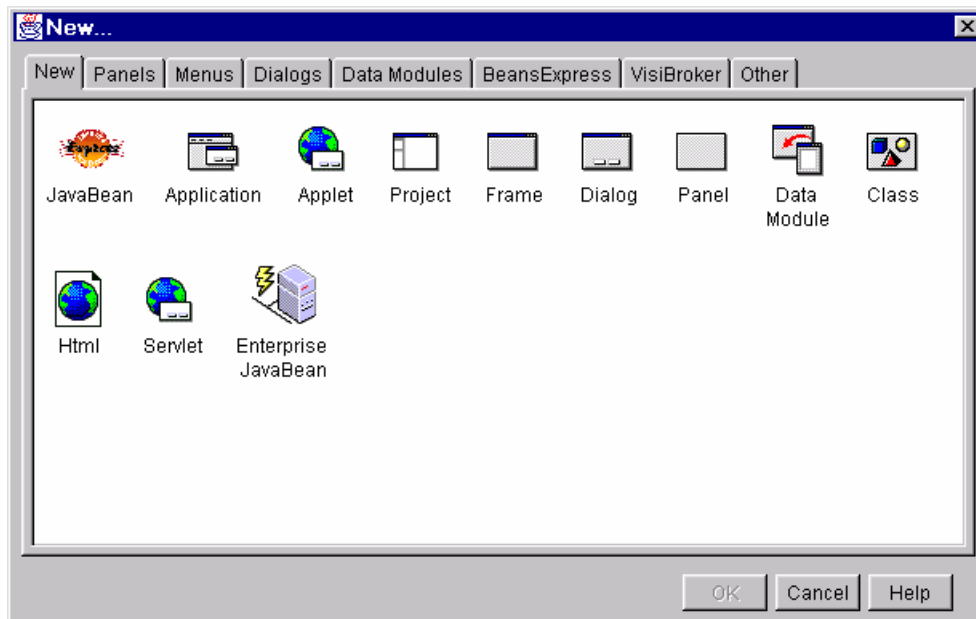
A new AppBrowser appears:



- Choose File|New... from the menu bar

The Object Gallery is displayed





- Choose Application and click the OK button

The new Application Wizard appears



The package field contains by default the name of the .jpr file, namely simpleproject.

- Fill in the class name: **MainApp**

The File name is automatically filled by JBuilder

- Check the Generate header comments checkbox
- Click the Next button

The second window of the Application Wizard appears

- Fill in the Class field and Title field

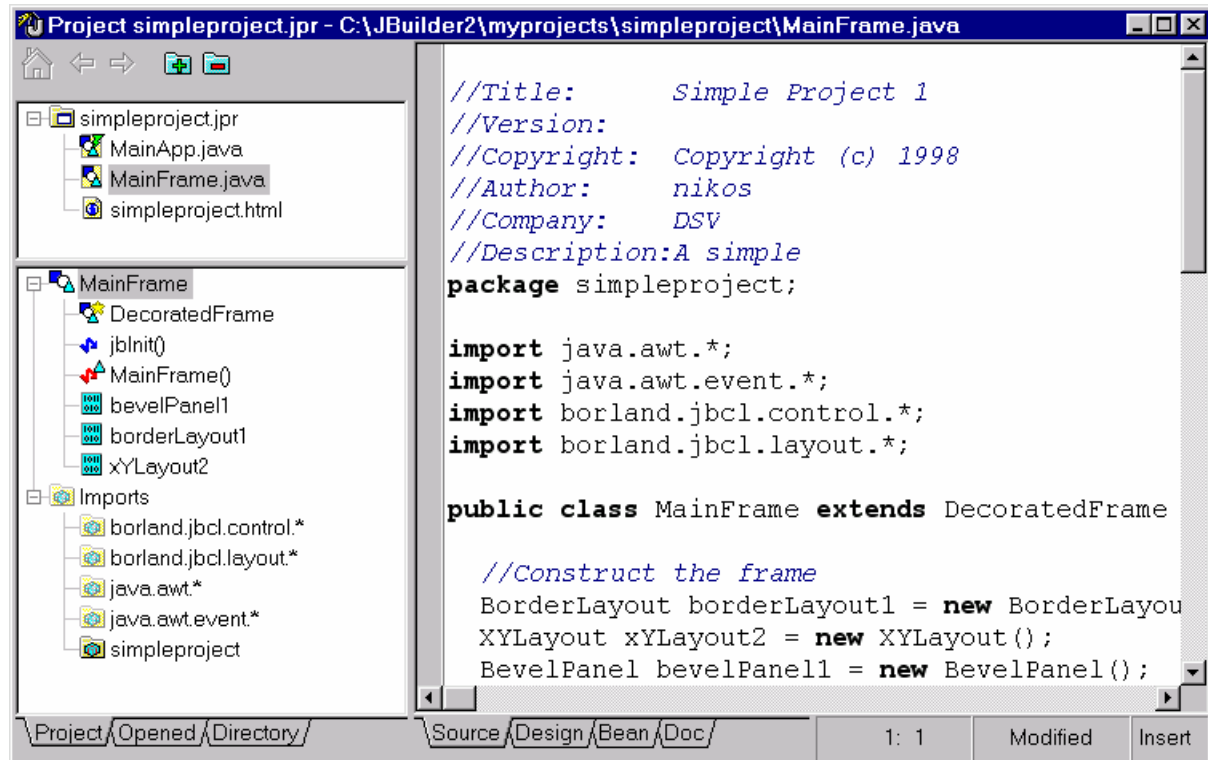
class: **MainFrame**

title: **Main Frame for Simple Project**

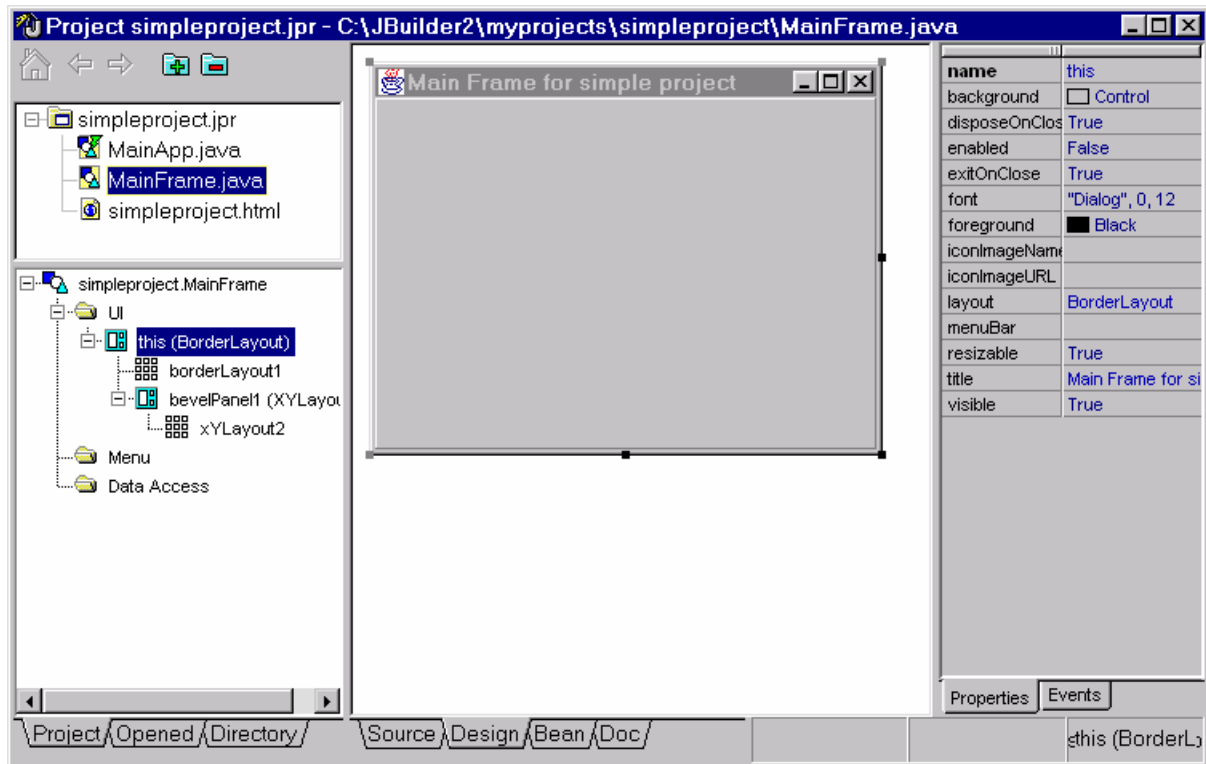


- Press the Finish button

Two new files are now included to your project: MainFrame.java and MainApp.java



- Mark the MainFrame.java in the Navigation pane
- Activate the design tab in the Content pane



The Inspector is also visible now (on the right side of the AppBrowser)

- Activate the AWT tab in the Component Palette



- Add two Button components, a List component, a Textfield component and three Label components to the Frame<sup>3</sup>
- Edit in the inspector the following properties of the components that you added<sup>4</sup>:

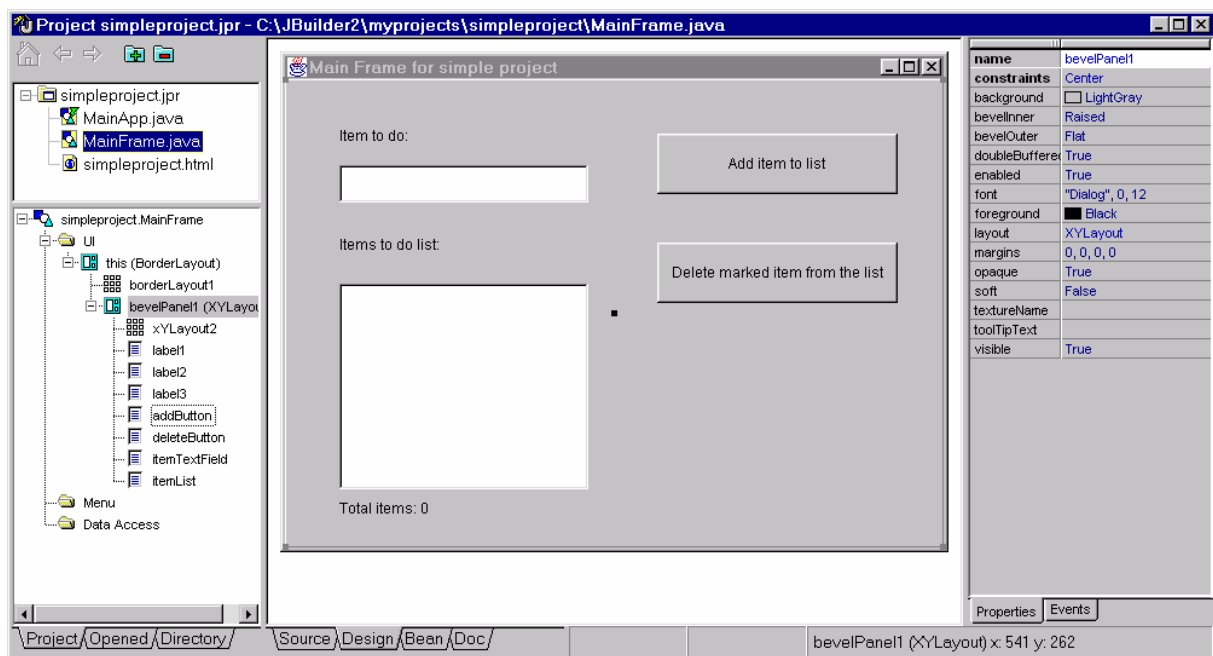
Component	Property	Value
label1	text	Item to do:

<sup>3</sup> You can add a component to the frame by first choosing the component you wish to add in your frame on the Component Palette and then clicking on the frame once at the position you want the component to be. After that you can easily resize the component.

<sup>4</sup> You can choose the component that you want to edit the properties of, by either clicking on it on the content pane (while in design state) or by choosing the component on the Structure pane (while the design tab is activated in the content pane).

label2	text	Items to do list:
label3	text	Total items: 0
button1	label	Add item to list
	name	addButton
button2	label	Delete marked item from the list
	name	deleteButton
textField1	name	itemTextField
	text	
list1	name	itemList
	background	White

The AppBrowser should now look something like this:

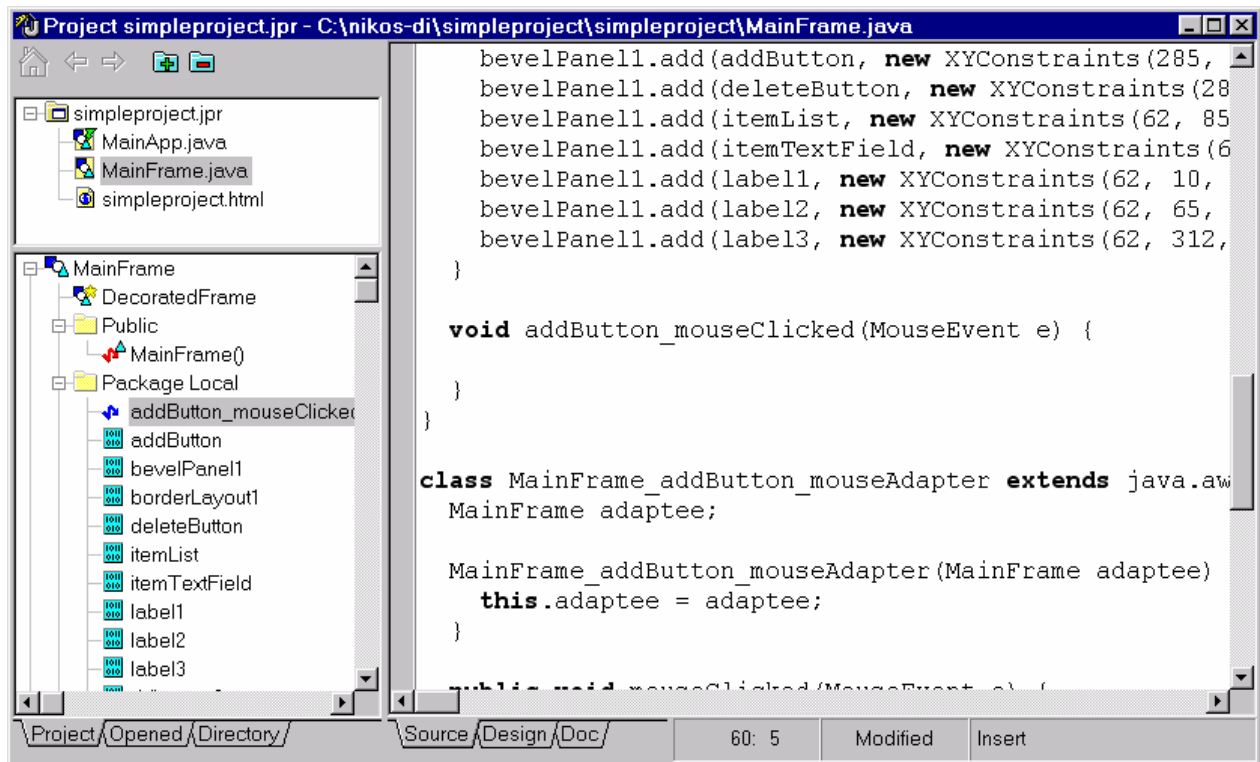


Now that the interface is ready we can start adding some functionality

Pressing the addButton should take the content of the itemTextField and add a new row in the itemList. Once this will happen every time the addButton is pressed, it is the addButton's mouseClicked event we should work with.

- Activate the addButton and show the events tab on the Inspector
- Click once on the mouseClicked event to activate it
- Doubleclick on the empty field next to mouseClicked event

JBuilder swaps automatically to the Source tab in the content pane and it creates a new method header



➤ Write the following code:

```
itemList.addItem(itemTextField.getText());
```

add a new item to the list, the new item has the current text of the itemTextField.

```
label3.setText("Total Items: " + itemList.getItemCount());
```

update the third label so that it shows the correct amount of items in the list.

The method should now look like this:

```
void addButton_mouseClicked(MouseEvent e) {
    itemList.addItem(itemTextField.getText());
    label3.setText("Total Items: " + itemList.getItemCount());
}
```

➤ Do the same with the deleteButton but write instead, the following code:

```
itemList.remove(itemList.getSelectedIndex())
```

remove from the list the item that is selected

```
label3.setText("Total Items: " + itemList.getItemCount());
```

update the third label so that it shows the correct amount of items in the list.

The method should now look like this:

```
void deleteButton_mouseClicked(MouseEvent e) {
    itemList.remove(itemList.getSelectedIndex());
    label3.setText("Total Items: " + itemList.getItemCount());
}
```

As it is now, the user may add an item with no text or try to remove an item while no item is selected. To avoid those cases we can enable and disable the add and delete buttons according to the current status of the itemTextField and the itemList. We can start by making the default status of both buttons to disabled. that can be done easily by selecting the buttons and changing their enabled property on the Inspector to False.

The addButton should be enabled every time the text in the itemTextField is not empty (its length is greater than 0), and disabled when the text is empty. The text of the itemTextField can also be cleared after the addButton has been pressed.

To do that we can use the textValueChanged event of the itemTextfield and the mouseClicked event of the addButton.

➤ Add the following line to the textValueChanged event of the itemTextfield:

```
if (itemTextField.getText().length()>0){    If the length of the text in the itemTextField is greater than 0
    addButton.setEnabled(true);           then enable the addButton
}
else {                                     else
    addButton.setEnabled(false);         disable the addButton
}
```

The method should now look like this:

```
void itemTextField_textValueChanged(TextEvent e) {
    if (itemTextField.getText().length()>0){
        addButton.setEnabled(true);
    }
    else {
        addButton.setEnabled(false);
    }
}
```

➤ Add the following line of code to the mouseClicked event of the addButton:

```
itemTextField.setText("")    clear the text of the itemTextField
```

The method should now look like this:

```
void addButton_mouseClicked(MouseEvent e) {
    itemList.addItem(itemTextField.getText());
    label3.setText("Total Items: " + itemList.getItemCount());
    itemTextField.setText("");
}
```

Similarly the deleteButton should be enabled when an item on the list is selected and disabled when no item is selected (or after the deleteButton has been pressed). To do that we can use the itemStateChanged event of the itemList and the mouseClicked event of the deleteButton.

- Write the following code in the itemStateChanged event of the itemList:

```
if (itemList.getSelectedIndex() != -1) {    If an item is selected in the itemList (-1 indicates no selected item)
    deleteButton.setEnabled(true);        then enable the deleteButton
}
```

- Add the following code to the mouseClicked event of the deleteButton:

```
deleteButton.setEnabled(false);    disable the deleteButton
```

The entire method should now look like this:

```
void deleteButton_mouseClicked(MouseEvent e) {
    itemList.remove(itemList.getSelectedIndex());
    label3.setText("Total Items: " + itemList.getItemCount());
    deleteButton.setEnabled(false);
}
```

- You can now save and run the application!

The only problem now is that both buttons are enabled by default when we start the application. To change that we can use an event that occurs when the application starts. The componentShown event of the mainFrame (“this”) is such an event.

- Add the following code to the componentShown event of the “this”:

```
addButton.setEnabled(false);    disable the addButton
deleteButton.setEnabled(false);  disable the deleteButton
```

The entire method should now look like this:

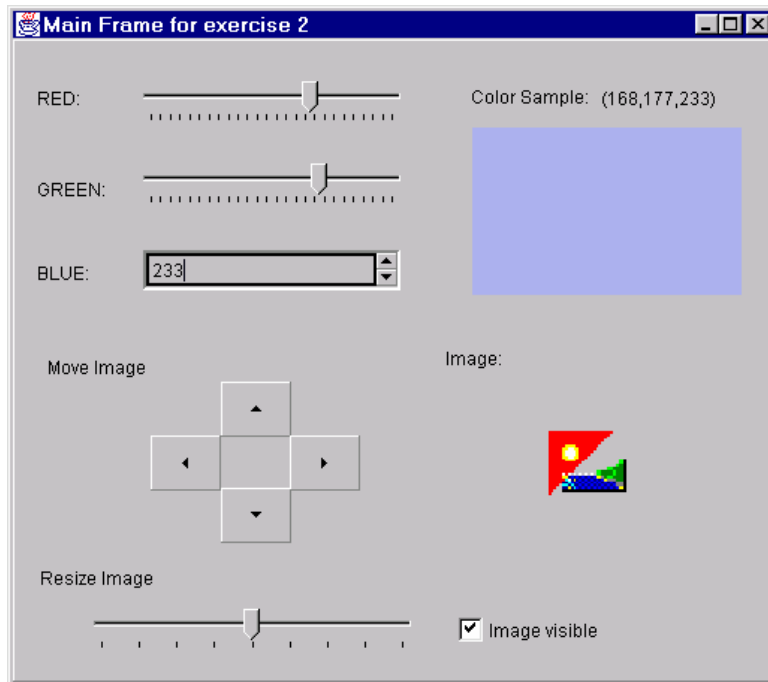
```
void this_componentShown(ComponentEvent e) {
    addButton.setEnabled(false);
    deleteButton.setEnabled(false);
}
```

- Save and run the application!

## 4.2 Using additional controls

In this exercise we will work with components from the KL Group tab, the AWT tab and the JBCL tab<sup>5</sup>. We will play with colors and images...

The application will look like this when we are done:



In the upper half of the frame we experiment with colors. Adjusting red, green and blue with the components on the left side affect the color shown on the right. In the lower half we have an image which we can move in different directions and change the size of.

- Start JBuilder and close all open projects
- Create a new application according to the following:
- When the Project wizard appears set the file field to:  
***Your home directory\JBuilder exercises\exercise2\exercise2.jpr***
- Press the finish button
- When the Application wizard appears set the class field to:  
**MainApp**
- Press the next button
- Fill in the Class field and Title field (for the frame)

---

<sup>5</sup> I had thought about using a tabManager component for this exercise but once they require the use of containers and layouts we will wait until the next exercise.



class: **MainFrame**

title: **Main Frame for Simple Project**

- Press the finish button

The AppBrowser appears

- Select the MainFrame.java and activate the design tab
- Add the following components to the upper half of the frame (you may need to resize the frame):

component name	components tab	property name	property value
label1	AWT	text	<b>RED:</b>
label2	AWT	text	<b>GREEN:</b>
label3	AWT	text	<b>BLUE:</b>
label4	AWT	text	<b>Color Sample:</b>
label5	AWT	text	<b>(0,0,0)</b>
jCSlider1	KL Group	maximum	<b>255</b>
jCSlider2	KL Group	maximum	<b>255</b>
jCSpinBox1	KL Group	maximum	<b>255</b>
panel1	AWT		

The upper part of the frame should now look like this:



When we start the application the color of the panel should be (0,0,0) (meaning black). To initiate the panel to black we have to write the following line of code in a method that is going to be executed before the frame is visible. The componentShown method of the frame (this) is such.

- Select the frame and enter the componentShown event from the inspector
- Write the following code:

```
panel1.setBackground(new Color(0,0,0))    set the background color of the panel to black
```

Now we can write some code to connect the sliders and the spinBox to the panel's color:

The sliders have an `adjustmentValueChanged` event, the `spinBox` has a `spinBoxChangeEnd` event.

➤ In those events write the following code:

```
panel1.setBackground(new Color(jCSlider1.getValue(),jCSlider2.getValue(),jCSpinBox1.getIntValue()));
label5.setText(""+jCSlider1.getValue()+" "+jCSlider2.getValue()+" "+jCSpinBox1.getIntValue()+"");
```

These lines update the color of the panel by gathering the values of the sliders and the `spinBox` and creating the matching color. The label that shows the numbers that define the color is also updated.

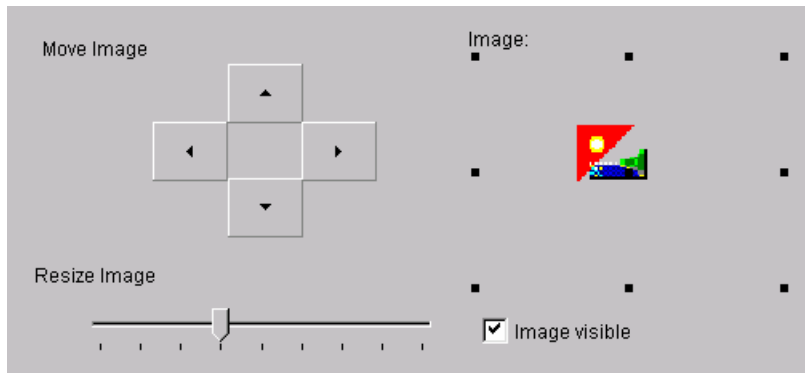
This half of the exercise is now finished. You can test the application by saving and running it.

➤ Add the following components in the lower part of the frame:

component name	components tab	property name	property value
label6	AWT	text	<b>Move Image</b>
label7	AWT	text	<b>Resize Image</b>
label8	AWT	text	<b>Image:</b>
jCArrowButton1	KL Group	orientation	<b>RIGHT</b>
jCArrowButton2	KL Group	orientation	<b>UP</b>
jCArrowButton3	KL Group	orientation	<b>DOWN</b>
jCArrowButton4	KL Group	orientation	<b>LEFT</b>
jCSlider3	KL Group	maximum	<b>100</b>
		minimum	<b>20</b>
		value	<b>50</b>
checkbox1	AWT	label	<b>Image visible</b>
		state	<b>true</b>
panel2	AWT	layout	<b>XYLayout</b>
transparentImage1	JBCL	imageName	<b>C:\JBuilder\lib\image32\trimage.gif</b>
		alignment	<b>HStretch+VStretch</b>

Make sure to place the `transparentImage1` on the `panel2`. We do that in order to prevent the image from moving over the entire frame. If the image is placed on a panel then it cannot move outside the panel's borders.

The lower part of the frame should now look like this:



Now to add some functionality:

The checkbox1 has an event called `itemStateChanged`.

➤ Add to this event the following code:

```
transparentImage 1.setVisible(checkbox 1.getState());
```

set the visible property of the `transparentImage 1` to the same as the `checkbox1`'s state property

➤ Add the following code to the `adjustmentValueChanged` event of the `jCSlider3`:

```
transparentImage 1.setSize(jCSlider3.getValue(),jCSlider3.getValue());
```

set the size of the `Image` to current value of the `jCSlider`

➤ Add to the `jCArrowButtons`' `mouseClicked` events the following code:

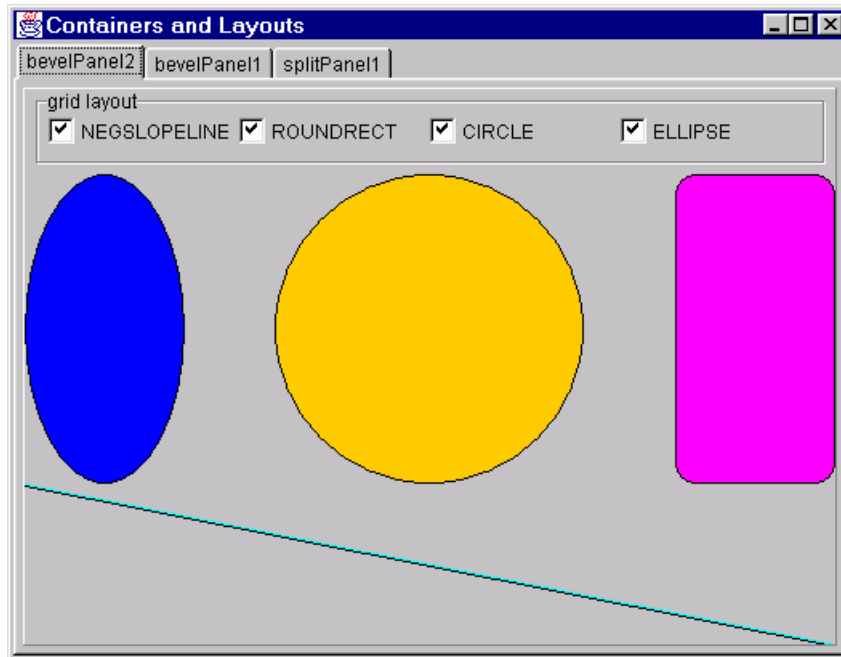
```
transparentImage 1.setLocation(transparentImage 1.getLocation().x,transparentImage 1.getLocation().y-1);    UP  
transparentImage 1.setLocation(transparentImage 1.getLocation().x,transparentImage 1.getLocation().y+1);    DOWN  
transparentImage 1.setLocation(transparentImage 1.getLocation().x-1,transparentImage 1.getLocation().y);    LEFT  
transparentImage 1.setLocation(transparentImage 1.getLocation().x+1,transparentImage 1.getLocation().y);    RIGHT
```

We simply get the location of the image, alter it and set it again

➤ Save and run the application

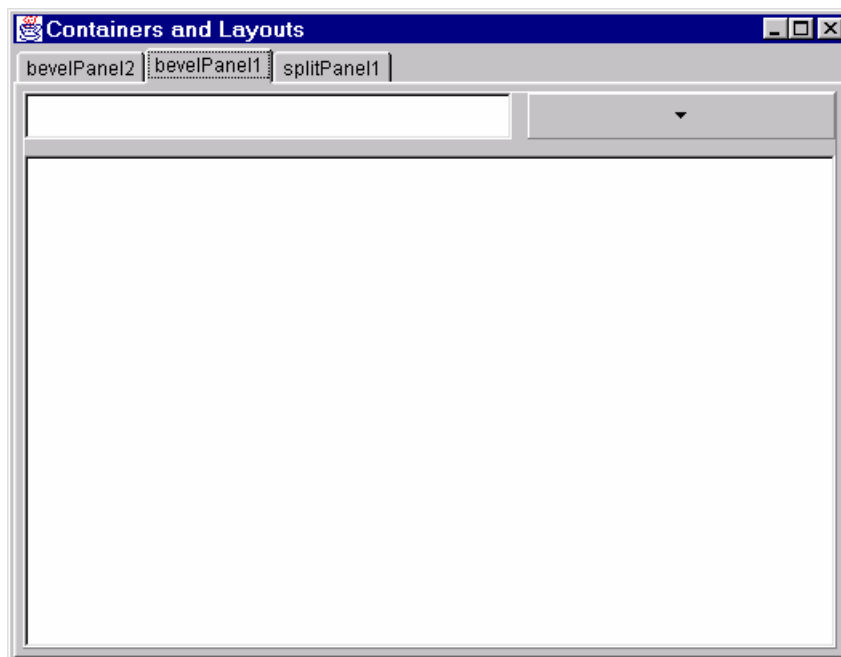
### 4.3 Using containers and layouts

A Container component is a component that contains other components. A container can be used to group other components and to arrange them in such way so that their layout can be maintained when the size of the container is altered. In this exercise there is no functionality implemented. This exercise consists of one frame which contains one `tabSetPanel` component with 3 tabs. The first tab is designed with border layout and it looks like this:

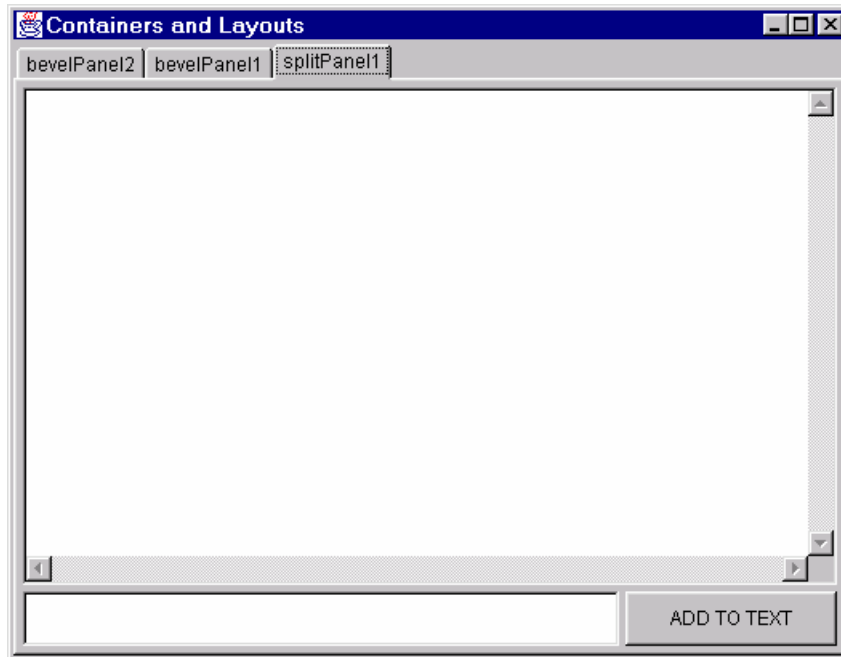


This tab contains a groupbox (which is also a container) which is using Grid layout.

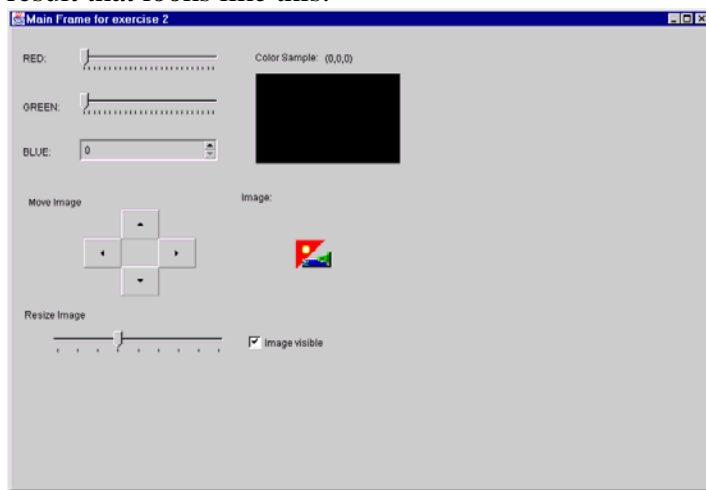
The second tab is designed with pane layout and looks like this:



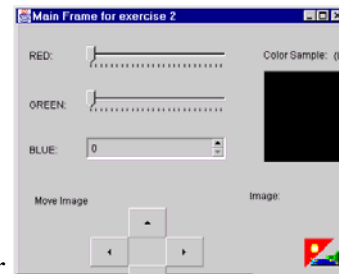
The third tab uses a splitPanel component, which is a container that uses pane layout. This tab looks like this:



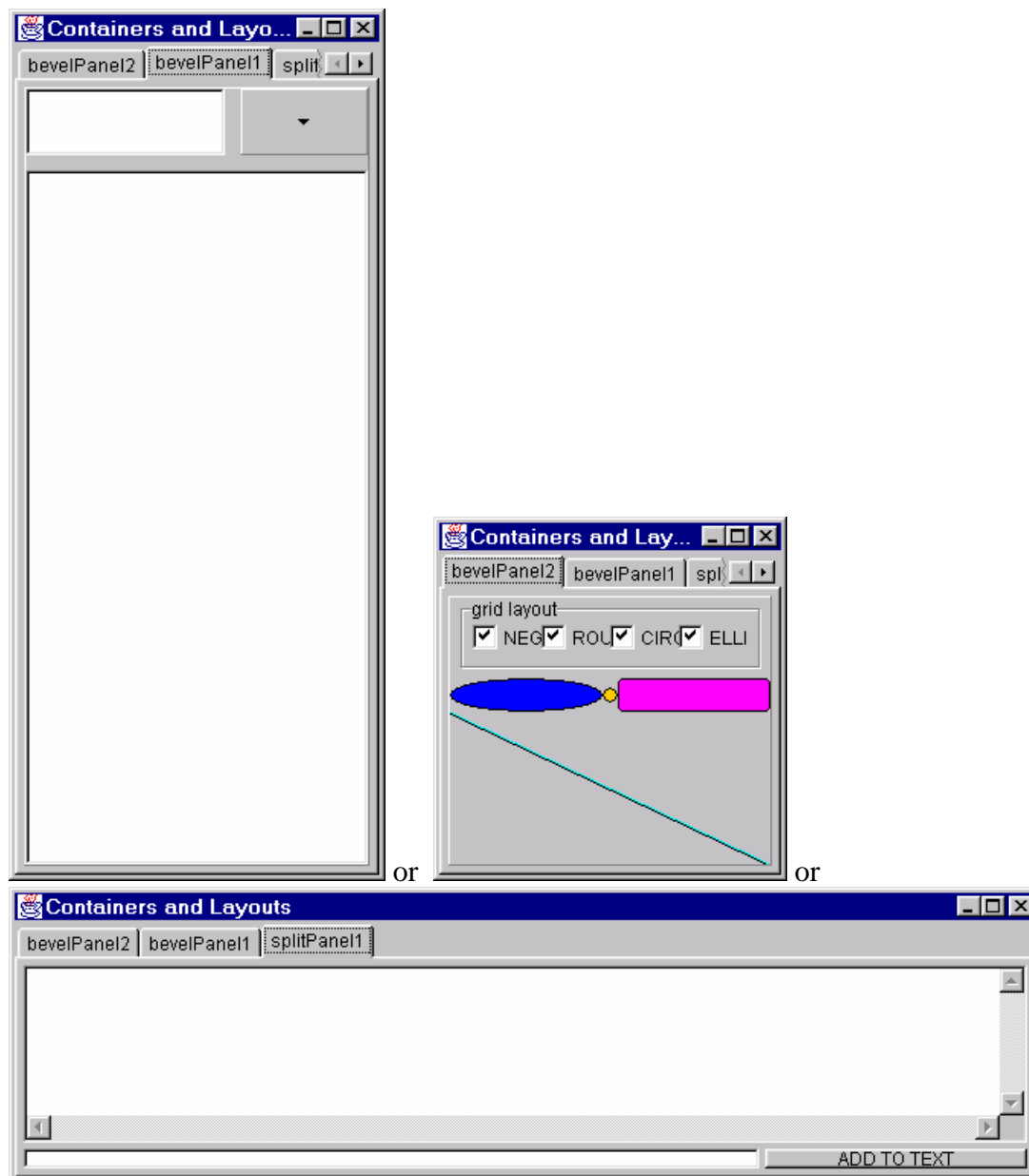
In the previous exercise we used a frame that used XYlayout. If we resize that frame we get a result that looks like this:



or



But if we resize the frame of this exercise all the objects on the frame are resized and repositioned. The result is like this:

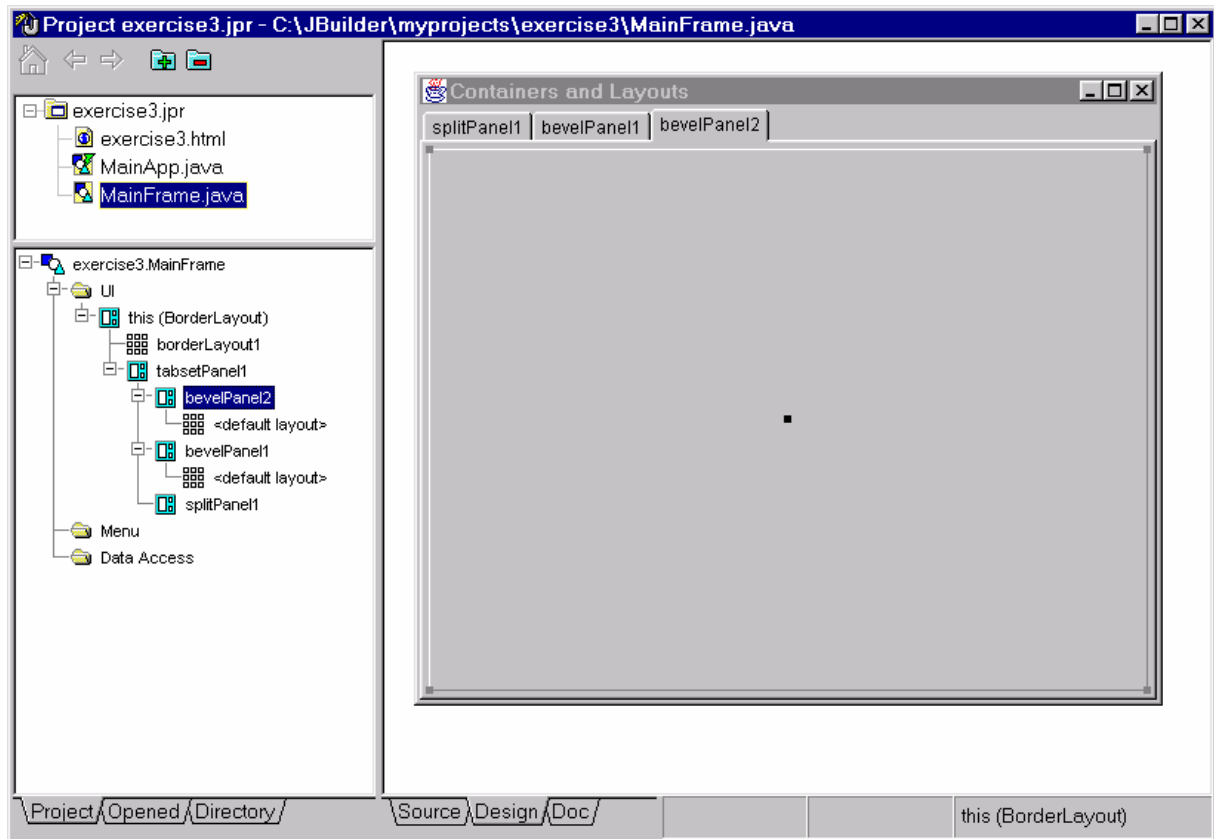


- Start by creating a new application
- Name the project *Your home directory*\JBuilder exercises\exercise3\exercise3.jpr
- Name the application **MainApp**
- Name the frame **MainFrame**
- Show the MainFrame on the design tab of the content pane
- Remove the bevelPanel that is automatically created on the frame
- Add a tabsetPanel to the frame

- Add a splitPanel and two bevelPanels to the tabsetPanel<sup>6</sup>

This should create three tabs on your tabsetPanel

When you have done that, your AppBrowser should look like this:



To change between tabs you can use the `selectedIndex` property of the `tabsetPanel1`<sup>7</sup> or select the respective panel on the structure pane

- Activate the tab named `bevelPanel1`
- Select the `bevelPanel1` component that rests on this tab
- Change the layout property of the `bevelPanel1` to **paneLayout**
- Do likewise with `bevelPanel2` and set its layout property to **borderLayout**

`SplitPanel1` has no layout property! It is by default of `paneLayout`

On the `bevelPanel1` add the following components:

---

<sup>6</sup> To make sure that the panels are placed on the `tabsetPanel` and not on each other you can add them to the `tabsetPanel` on the Structure Pane.

<sup>7</sup> The first tab has index 0

component	property	value
list		
jCArrowButton	orientation	<b>DOWN</b>
textField		

To add a small gap between the components change the gap property of the paneLayout1 that is directly after the bevelPanel1 on the Structure Pane

- Activate the bevelPanel2 tab and add to the bevelPanel2 four shapeControl components and one groupBox component
- Set their properties to match the following:

component	property	value
shapeControl1	type	<b>NegSlopeLine</b>
	foreground	<b>Cyan</b>
	constraints	<b>South</b>
shapeControl2	type	<b>Ellipse</b>
	foreground	<b>blue</b>
	constraints	<b>Center</b>
shapeControl3	type	<b>RoundRenc</b>
	foreground	<b>Magenta</b>
	constraints	<b>East</b>
shapeControl4	type	<b>Circle</b>
	foreground	<b>Orange</b>
	constraints	<b>West</b>
groupBbox1	constraints	<b>North</b>
	layout	<b>GridLayout</b>

To change the gap between the components change the properties of the borderLayout1 that is directly after the bevelPanel2 on the Structure Pane

- Add four checkboxes to the groupBox1:

component	property	value
checkbox1	label	<b>RoundRenc</b>
	state	<b>true</b>
checkbox2	label	<b>Circle</b>
	state	<b>true</b>
checkbox3	label	<b>Ellipse</b>
	state	<b>true</b>
checkbox4	label	<b>NegSlopeLine</b>
	state	<b>true</b>



You can change the way the checkboxes are placed within the groupBox1 by editing the properties of the gridLayout1 that is directly after the groupBox1 on the Structure Pane

- Activate the splitPanel1 tab and add the following components to the splitPanel1:

component	property	value
textField		
textArea		
button	label	<b>Add to text</b>

Once the splitPanel1 is always of paneLayout it has the gap property of the paneLayout itself

- Save and run the application
- Try resizing the frame at runtime

#### 4.3.1 Further practice

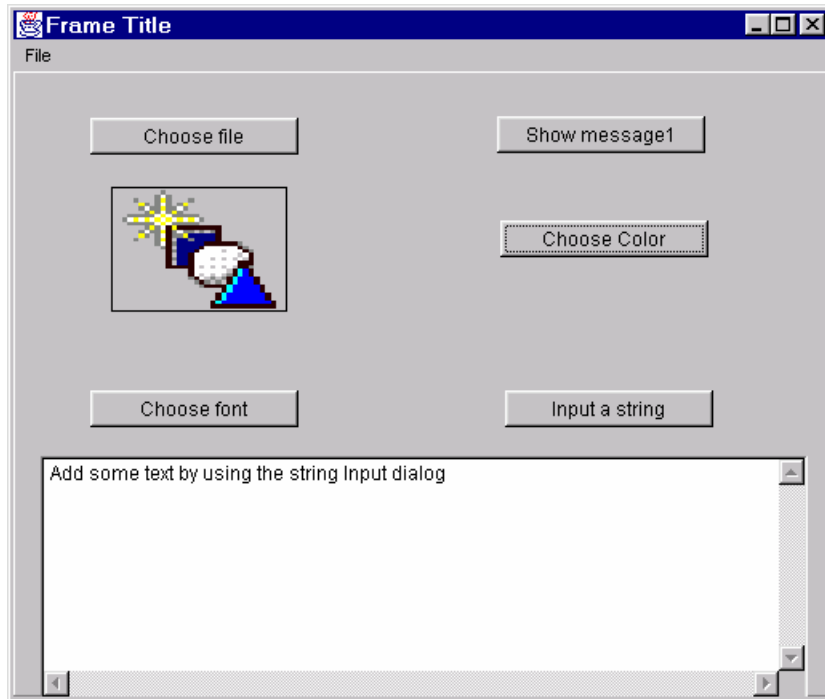
- Connect the checkboxes to show and hide the shapeControls
- Make the button and the jCArrowButton move the text from the textFields to the textArea and the list

### 4.4 Using Dialogs and Menus

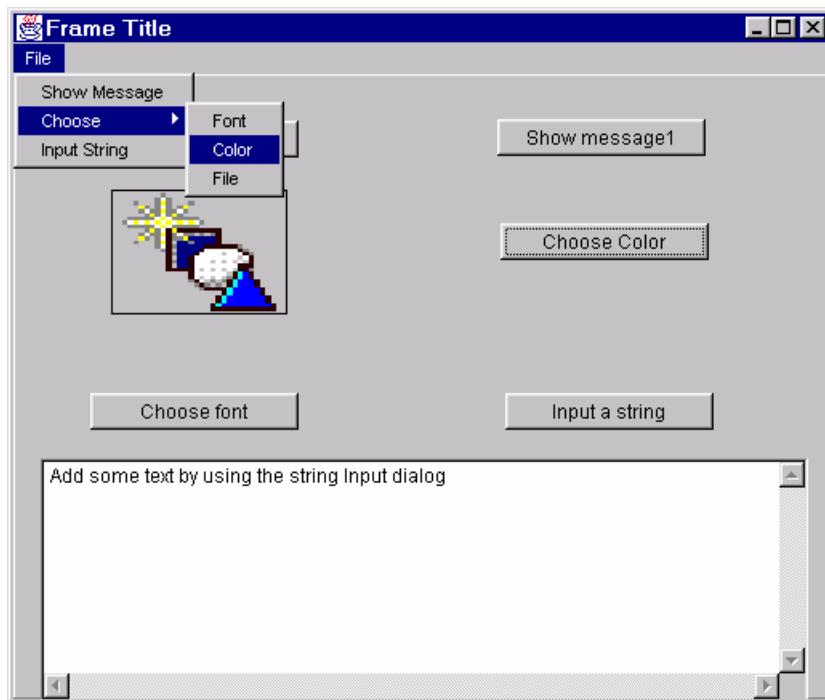
JBuilder provides two types of dialogs. There are the ones that can be found in the Object Gallery (which we are not going to use) and the ones that rest on the Dialogs tab of the Component Palette. The dialogs in the Object Gallery can be adjusted to the programmers needs but they require a certain amount of extra code in order to work properly. The dialogs of the Component Palette are standard dialogs that cannot be changed so much. The advantage with them is that they can be used directly in your project without any extra code.

The menus that come with JBuilder are not stable. In this exercise a menubar component from the AWT tab is used.

This exercise contains one frame, which can call different types of dialogs by either clicking buttons or using menus. The main frame of the exercise looks like this:



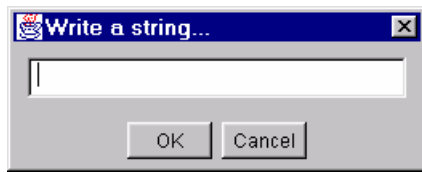
The five buttons call five different dialogs. The same can be done from the menus:



Show message shows the following message dialog:

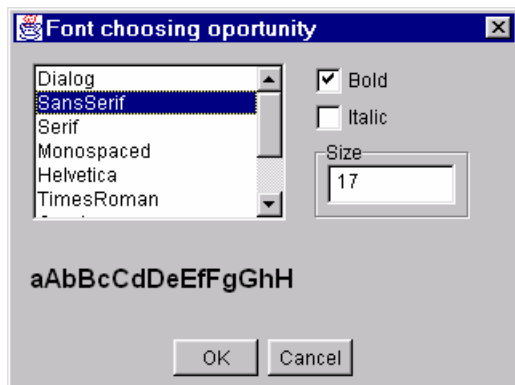


Input String activates the following dialog:



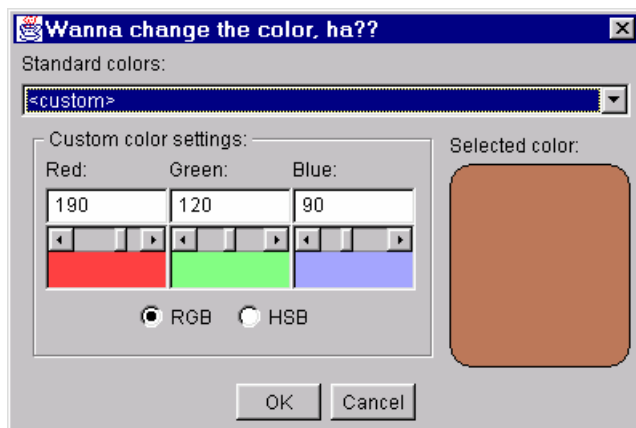
Use this dialog to add strings to the text area

Choose Font shows this dialog:



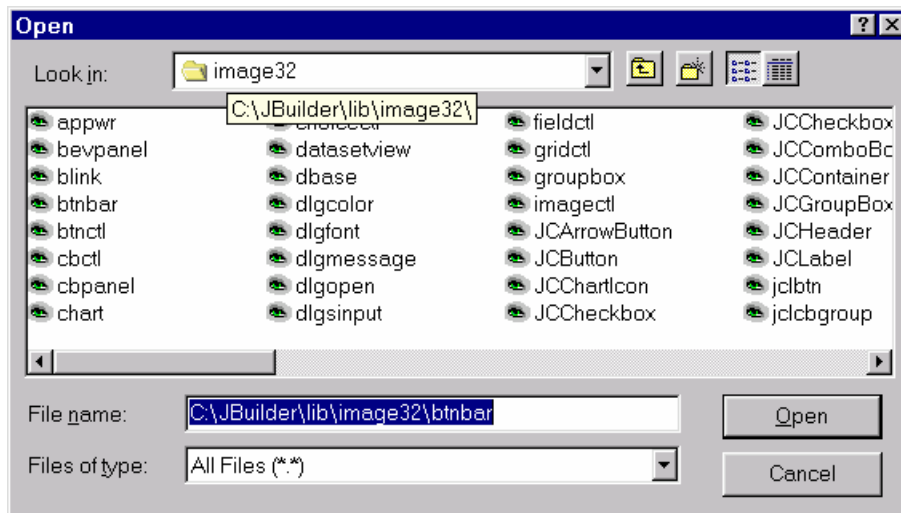
With this dialog you can choose the font that is used in the text area

Choose Color brings up the following dialog:



This dialog is used in this exercise for picking the background color of the frame

Choose file activates the following dialog:



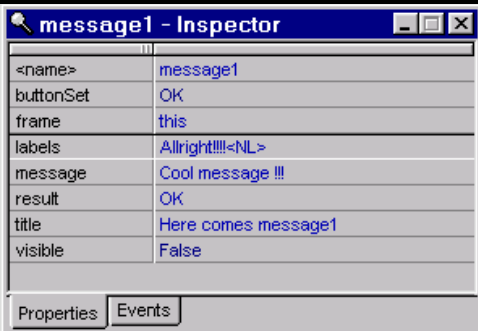

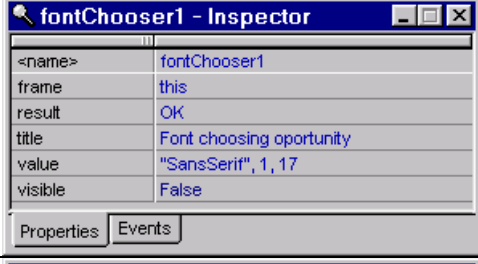
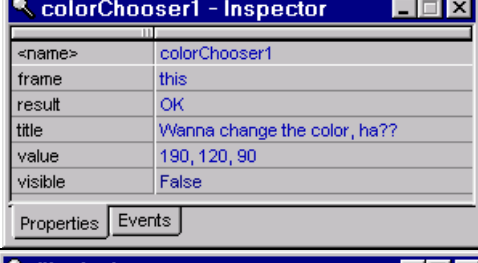
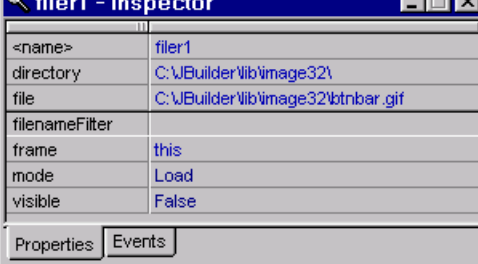
The file you choose is then connected to the image component.

To do this exercise, do the following:

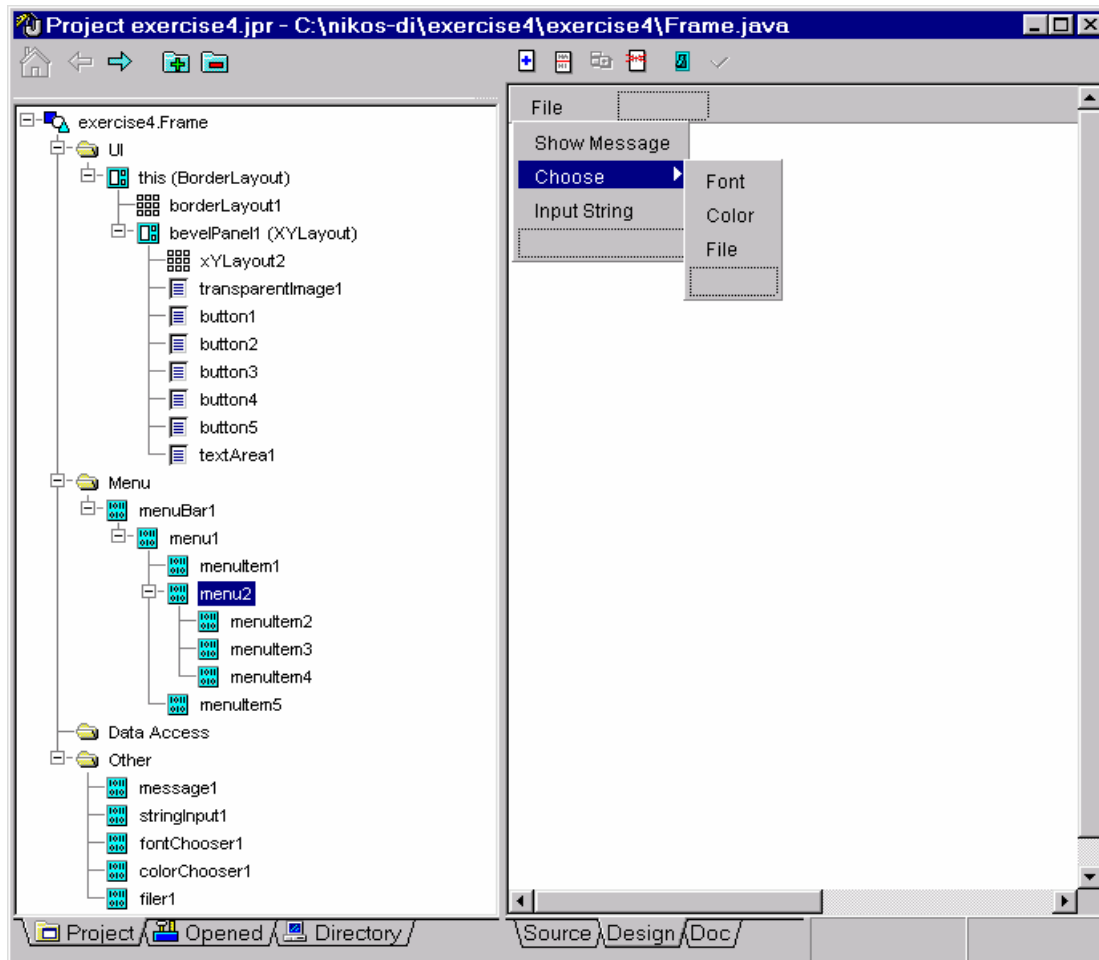
- Create a new application
- Name the project *Your home directory*\JBuilder exercises\exercise4\exercise4.jpr
- Name the application **MainApp**
- Name the frame **MainFrame**
- Show the MainFrame on the design tab of the content pane
- Add the following components with the following properties to the frame<sup>8</sup>:

component	property	value
button1	label	<b>Choose Color</b>
button2	label	<b>Show Message 1</b>
button3	label	<b>Choose File</b>
button4	label	<b>Input a string</b>
button5	label	<b>Choose a font</b>
textArea1	text	<b>Add some text by using the string Input dialog</b>
transparentImage1	imageName	<b>C:\JBuilder\lib\image32\appwr.gif</b>

<sup>8</sup> Some of the components are not visible on the frame during the design, to activate them for changing their properties you can select them on the structure pane

message1 <sup>9</sup>	
stringInput1	
fontChooser1	
colorChooser1	
filer1	
menuBar1	you can design the menus graphically, make the menus as they are in the image that follows

<sup>9</sup> A message has a property butonSet and a property labels. Normally the amount of labels should be the same with the amount of buttons. If there is not enough labels then JBuilder assigns default labels to the buttons that do not have a label. If the labels property is empty then all the buttons should get the default labels. Unfortunately JBuilder has a bug and sometimes creates the following line: message1.setLabels(null);, which has to be removed in order to run the application.



Once all this is done we can connect the buttons and the menuitems to the dialogs

All the dialogs have a show() method. By just calling this method the dialog is shown

You can then use methods like getResult() or getValue()

- Write the following code in the actionPerformed event method of the menuItem that is supposed to activate message1:

```
message1.show();
```

- Write the same code for the mouseClicked event of the correct button
- Write the following code in the actionPerformed event method of the menuItem that is supposed to activate colorChooser1:

```
colorChooser1.show();  
if (colorChooser1.getResult() == Message.OK) {  
    bevelPanel1.setBackground(colorChooser1.getValue());  
}
```

shows the dialog  
If the OK button was pressed then  
set the panels background to the color that was  
chosen

- Write the same code for the mouseClicked event of the correct button
- Write the following code in the actionPerformed event method of the menuItem that is supposed to activate filer1:

```
filer1.show();           show the dialog
try {                   try to the open file that was selected
    transparentImage1.setImageName(filer1.getDirectory()+filer1.getFile()); by the dialog as an image
}
catch (Exception exc){ if exception then do nothing
}
```

- Write the same code for the mouseClicked event of the correct button
- Write the following code in the actionPerformed event method of the menuItem that is supposed to activate fontChooser1:

```
fontChooser1.show();   show the dialog
if (fontChooser1.getResult() == Message.OK) { If the ok button was pressed then
    textArea1.setFont(fontChooser1.getValue()); set the font of the textArea to the font that was chosen
}
```

- Write the same code for the mouseClicked event of the correct button
- Write the following code in the actionPerformed event method of the menuItem that is supposed to activate stringInput1:

```
stringInput1.show();   show the dialog
if (stringInput1.getResult() == Message.OK) { If the OK button was pressed
    textArea1.append("\n"+stringInput1.getValue()); append a line break plus the input string to the
} textArea
```

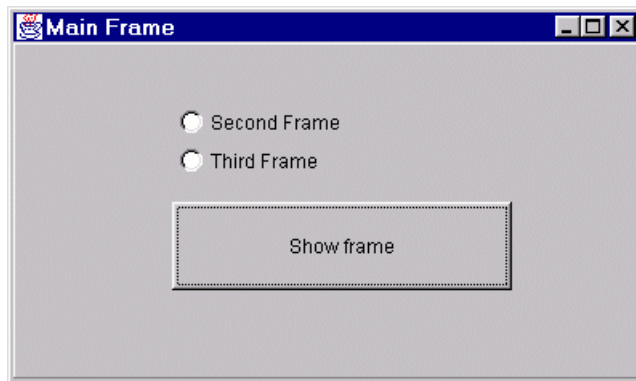
- Write the same code for the mouseClicked event of the correct button
- Save and run the application<sup>10</sup>

---

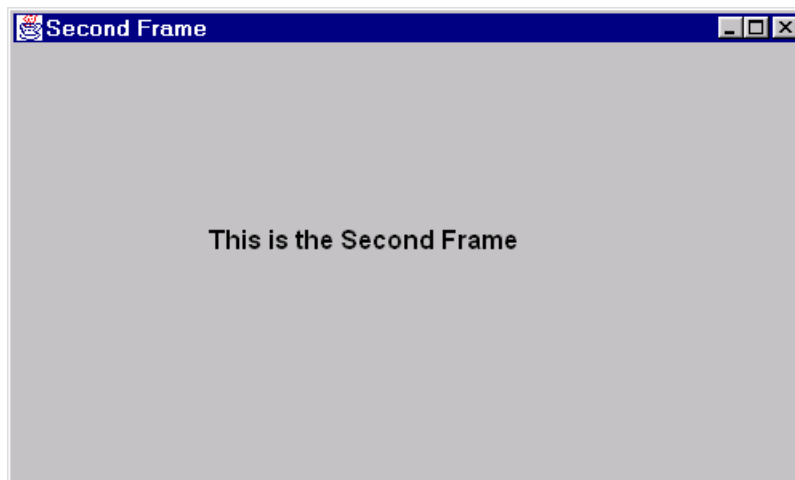
<sup>10</sup> JBuilder forgets sometimes to repaint the frame after you open a new image. The image is actually there but for some reason the bevelPanel covers it. The picture will appear if you, for example, resize the frame.

## 4.5 Working with Frames

In this exercise we will work with an application which consists of three frames. The one frame appears when we run the application, the other frames are shown when we press a button which rests on the main frame of the application. To make things more interesting we use `checkboxGroup`<sup>11</sup> to decide which of the frames will be displayed when the button is clicked. The main frame of the application shall look like this:



The two other frames look like this:



---

<sup>11</sup> A `checkboxGroup` behaves as a group of radio buttons but it is built with checkboxes and a `checkboxGroup`.





- Create a new application (as usual)
- Add two new frames (file|new – frame)
- Add two checkboxes and a checkboxGroup components to the main frame
- Set the checkboxGroup property of the two checkboxes to the checkboxGroup1
- Add a button component to the main frame
- Add the following line of code to the mouseClicked event of the button1:

```
if (checkbox1.getState()){           if checkbox1 is checked then
    secondFrame.show();         show the second frame
} else {                          else
    thirdFrame.show();         show the third frame
}
```

- Add the following code to the componentShown event of the “this” (of the main frame):

```
checkboxGroup1.setCurrent(checkbox1);    enable the first checkbox
```

- Change the exitOnClose property to False for the second and third frame<sup>12</sup>
- Add the following code to the windowClosing event of the second and third frame:

---

<sup>12</sup> Preventing a Frame from closing the Application by changing the Frame's exitOnClose property to False prevents the frame from closing. To restore the behavior of the close icon of the frame, the following code has to be added to the windowClosing event of the frame: `this.dispose();`

The method `dispose()` sets the frame's visible property to false. The frame is, however, still available, the frame is not destroyed.

`this.dispose();`

#### 4.5.1 Further practice

Write some code to count how many times you have shown each frame and display this number on the frame when you show it.

## 5 Introduction to DataGateway

Borland DataGateway is a collection of JDBC drivers that allow Java applications and applets on any platform to access data sources such as dBase, Paradox, Microsoft Access, Informix, IBM DB/2 and other. An alternative way to connect a database to a Java application or applet is by using an ODBC driver. In order to make a database available through DataGateway, the database has to be registered in the Borland Database Engine (BDE). Similarly, a database has to be registered in the ODBC Data Source Administrator, in order to be available through ODBC for a Java application.

## 6 Database exercises

In this chapter we present some basic database functionality that is available for JBuilder application. The techniques that are used here may not be the fastest or the optimum in all aspects.

In the following exercises, a sample database, created with MSAccess 97, is used. This sample database consists of 4 tables: Courses, Teachers, Students and StudentCourse. A course has a teacher who is responsible for this course. Every student has attended many courses and a course may be attended by lots of students. In order to break this many to many relationship, we add the studentcourse table. This model is extremely simplified, but is suitable for the exercises that follow.

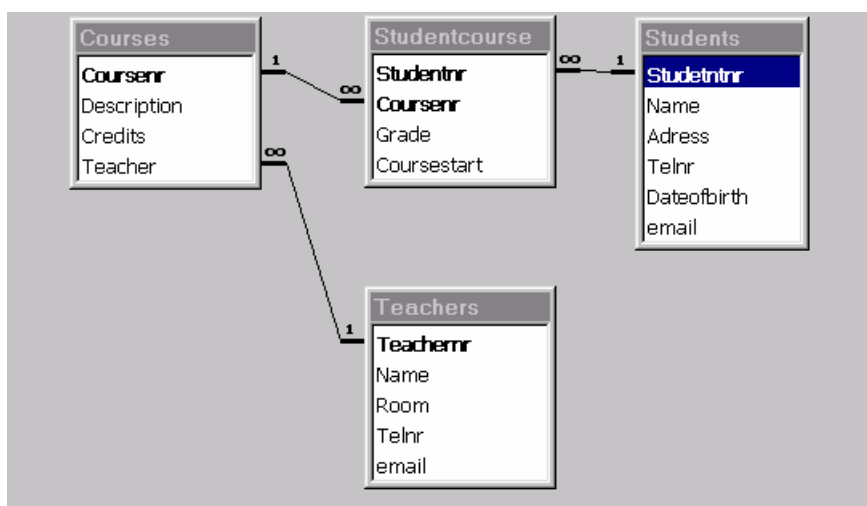


Figure 1 Relationships between tables in the sample database

This database is available at [//goofy/prog-1/kurser/2i-1100/sample.mdb](http://goofy/prog-1/kurser/2i-1100/sample.mdb) or can be downloaded from <http://L238.dsv.su.se/courses/2i-1100>. Copy the database file to your account and register it in the ODBC Data Source Administrator, as it is described later. You can use any alias name you like, but the alias used later in the exercises is “school”.

## 6.1 Making a database available

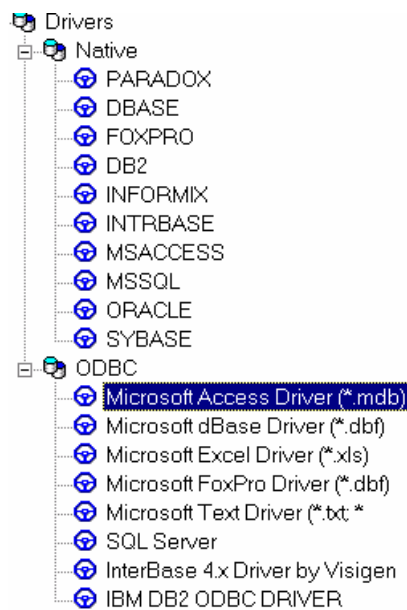
JBuilder provides different ways for connecting to a database. DataGateway and ODBC are two common ways. When the connection to the database has been established then both ways behave similarly. ODBC seems to be more stable and that is why all the exercises that follow build on ODBC connections.

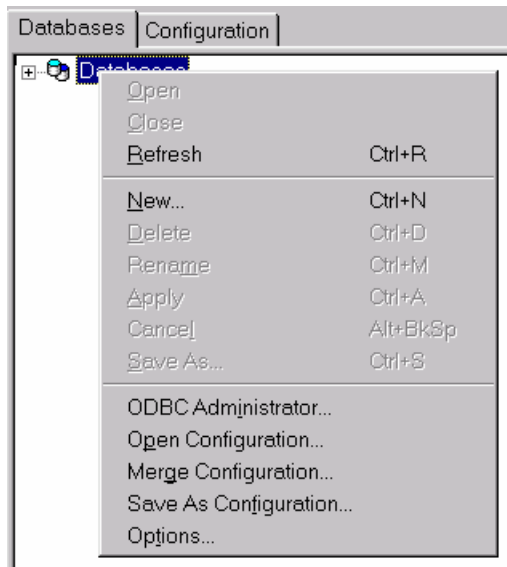
In any way both ways to make a database available for a JBuilder application are presented.

### 6.1.1 Through DataGateway

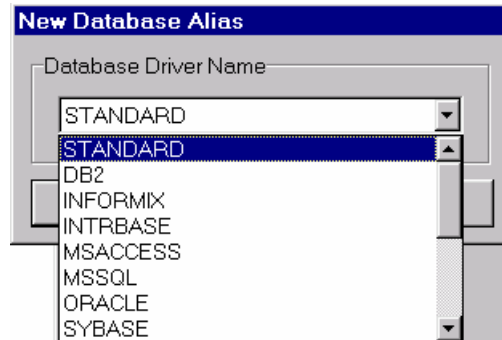
To make a database available through DataGateway, the database has to be registered in the BDE Administrator. The BDE Administrator is available in the control panel and under the Start-menu | Programs | Borland DataGateway. The BDE Administrator provider both ODBC and native drivers for connecting to the most common types of databases. In order to be able to connect to the database with DataGateway, the database has to be registered with a native driver.


Drivers available in the BDE Administrator. More drivers can be added on demand. The drivers shown here are the ones that come with the original installation of the BDE Administrator.





- By right clicking on the database and choosing New... (or pressing Ctrl+N) you can add a new database alias. The following window appears:



- Choose the native driver that matches the type of your database!
- Name your alias and define the properties in the right pane of the BDE Administrator so that your alias points at your database. Depending on the driver (and type of database) different properties may be available.
- When you are done with this, choose Object|Apply or press .

Now the database is available through DataGateway.

### 6.1.2 Through ODBC

To register a database with an ODBC driver you can use the ODBC Data Source Administrator, which can be found in the control panel. In the User DSN tab you can add your personal aliases to point to your databases.

To add a new alias do the following:

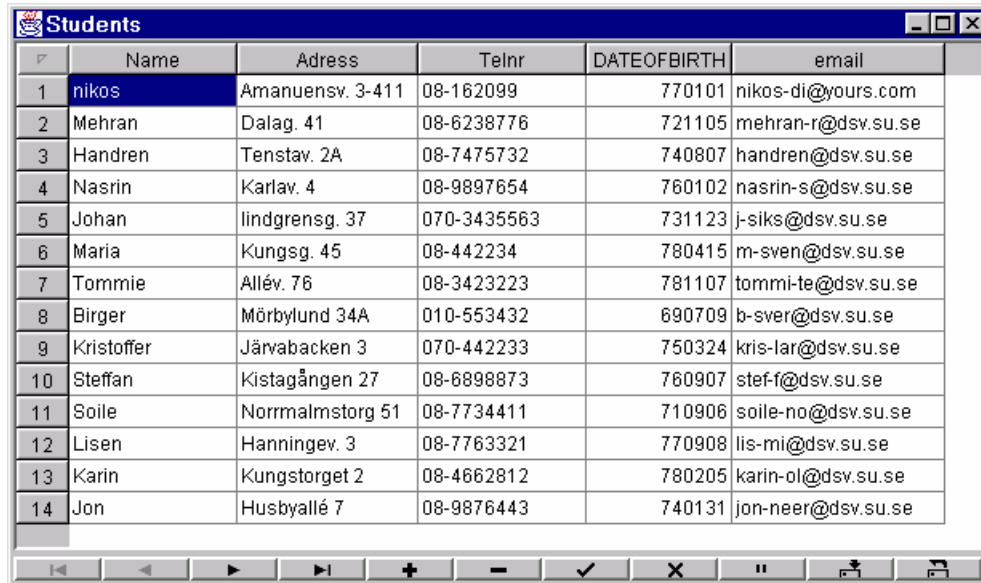
- Start the ODBC Data Source Administrator!
- Activate the User DSN tab!
- Press the Add... button!
- Choose the suitable driver!
- Press the Finish/Slutför button!
- Give a name to your alias, a description and the database to be associated with the alias!
- Press the OK button!

Now your database is available for JBuilder through an ODBC driver.

## 6.2 Show and update a single table

In this exercise we will just work with one table only. Because of referential integrity between the tables the table we will use is the table *Students*. To see and manipulate the data in the table we will just use a gridControl and a navigatorControl:

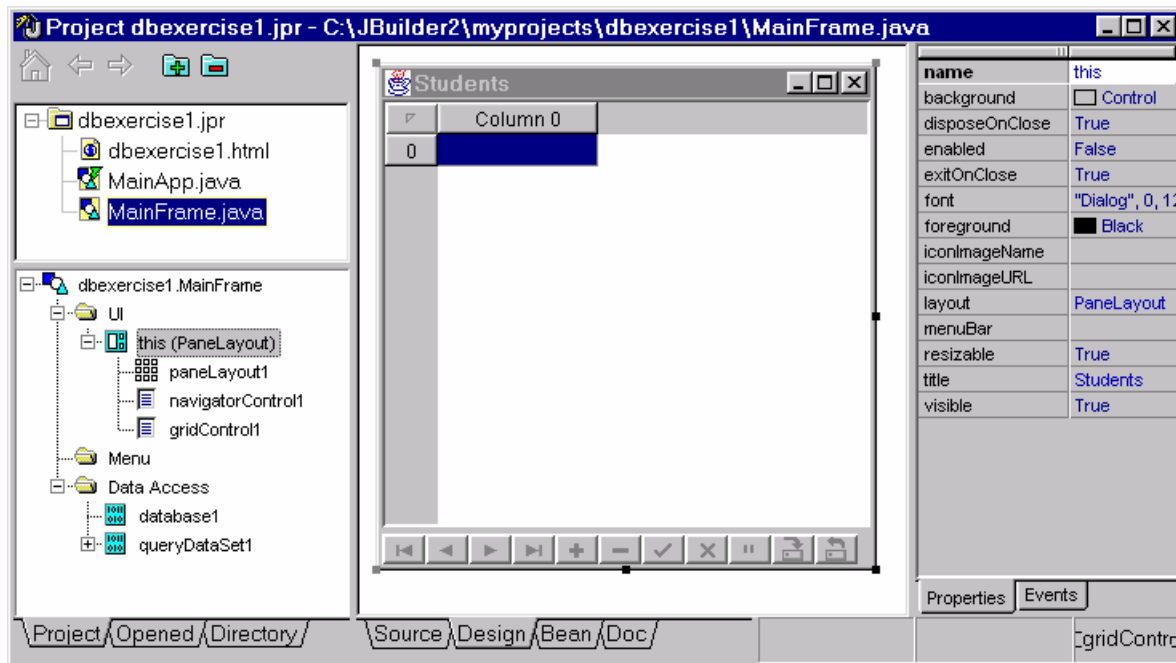
The application will consist of one frame that shall look like this:



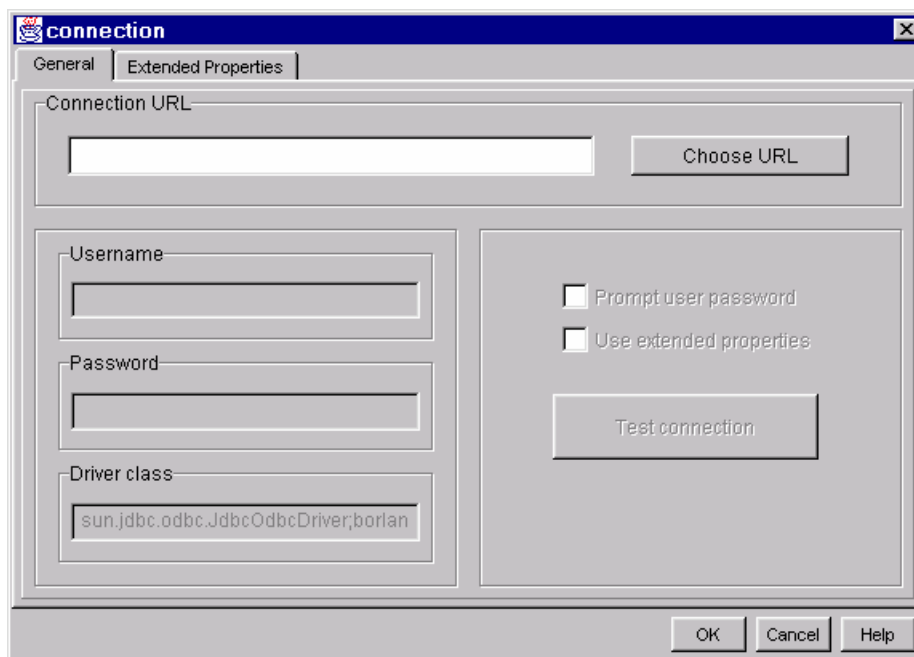
P	Name	Adress	Telnr	DATEOFBIRTH	email
1	nikos	Amanuensv. 3-411	08-162099	770101	nikos-di@yours.com
2	Mehran	Dalag. 41	08-6238776	721105	mehran-r@dsv.su.se
3	Handren	Tenstav. 2A	08-7475732	740807	handren@dsv.su.se
4	Nasrin	Karlav. 4	08-9897654	760102	nasrin-s@dsv.su.se
5	Johan	lindgrensg. 37	070-3435563	731123	j-siks@dsv.su.se
6	Maria	Kungsg. 45	08-442234	780415	m-sven@dsv.su.se
7	Tommie	Allév. 76	08-3423223	781107	tommi-te@dsv.su.se
8	Birger	Mörbylund 34A	010-553432	690709	b-sver@dsv.su.se
9	Kristoffer	Järvabacken 3	070-442233	750324	kris-lar@dsv.su.se
10	Steffan	Kistagången 27	08-6898873	760907	stef-f@dsv.su.se
11	Soile	Normalmstorg 51	08-7734411	710906	soile-no@dsv.su.se
12	Lisen	Hanningev. 3	08-7763321	770908	lis-mi@dsv.su.se
13	Karin	Kungstorget 2	08-4662812	780205	karin-ol@dsv.su.se
14	Jon	Husbyallé 7	08-9876443	740131	jon-neer@dsv.su.se

- Create a new application
- Name the project *Your home directory*\JBuilder exercises\dbexercise 1\dbexercise 1.jpr
- Name the application **MainApp**
- Name the frame **MainFrame**
- Show the MainFrame on the design tab of the content pane
- Remove the bevelPanel from the frame
- Change the layout property of the frame (this) to **PanelLayout**
- Add a GridControl and a NavigatorControl (they are in the JBCL-tab) to the frame
- Add a Database component and a QueryDataSet component (Data Express tab) to the frame

Your AppBrowser should look like this now:

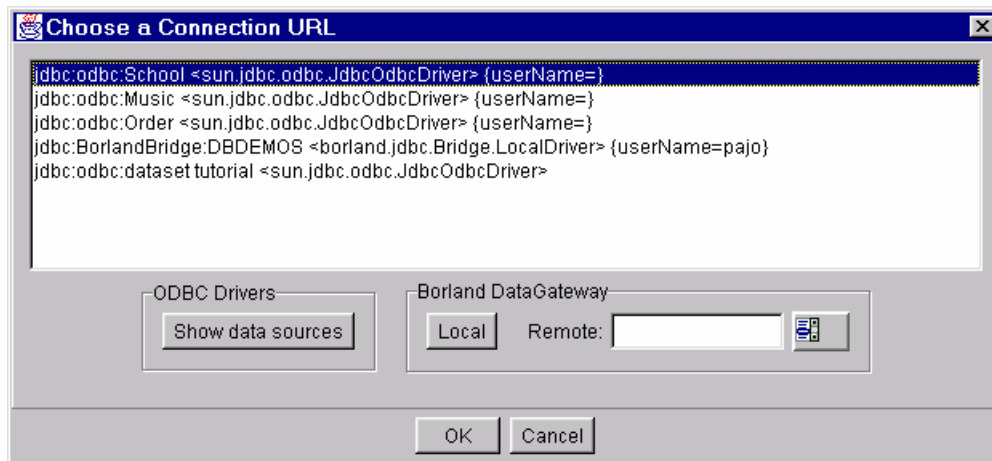


- Activate the database1 on the component tree
- Click on the connection property and make the connection wizard visible

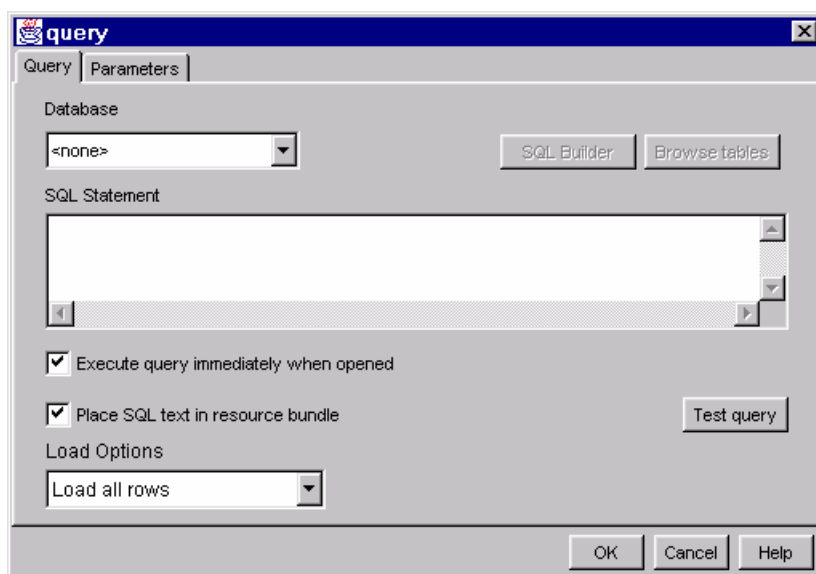


- Click on the Choose URL button

A new window comes up:

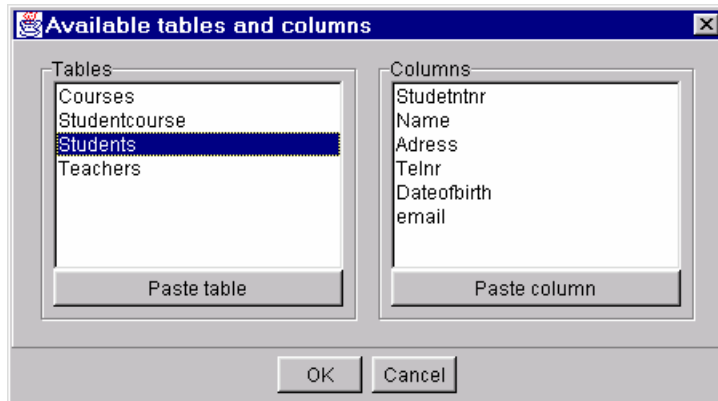


- Click on the Show data sources to show all the ODBC connections
- Choose the connection that points to the sample.mdb (the alias that you created in the ODBC Data Source Administrator)
- Click OK
- You can now test the connection by clicking on the Test connection button.
- If the connection is successful then click OK. If the connection fails it may depend that the database is open by another application.
- It is recommendable to save the application every few steps, just in case.
- Activate the queryDataSet1 on the component tree
- Click on the query property and make the query wizard visible



- Choose the **database1** as Database

(The Browse tables button is now active. Clicking it will show the following window:



This can be helpful if you are not sure of the names of the tables and columns in the tables.)

- Write the following SQL statement:

**SELECT \* FROM students**

- Click the Test query button to test the SQL
- Choose Cancel at the Create Resource Bundle window
- Select the **queryDataSet1** as the DataSet for both the gridControl1 and the navigatorControl1
- Change the metaDataUpdate property of the queryDataSet1 to **none**
- Change the tableName property of the queryDataSet1 to **students**
- Set the rowId property of the STUDENTNR column of the queryDataSet1 to **true**
- You can also edit the editMask and displayMask properties of for example the DATEOFBIRTH column, so that the date is displayed in a certain way
- Save and run the application

As you may have noticed the field STUDENTNR is a counter which does not have to be field by the user. We can therefore either make this column read only or we can remove it from the view of the table.

- To do that we just need to change the visible property of the STUDENTNR column of the queryDatSet1 to **true**, or alternatively the property readOnly to **true**

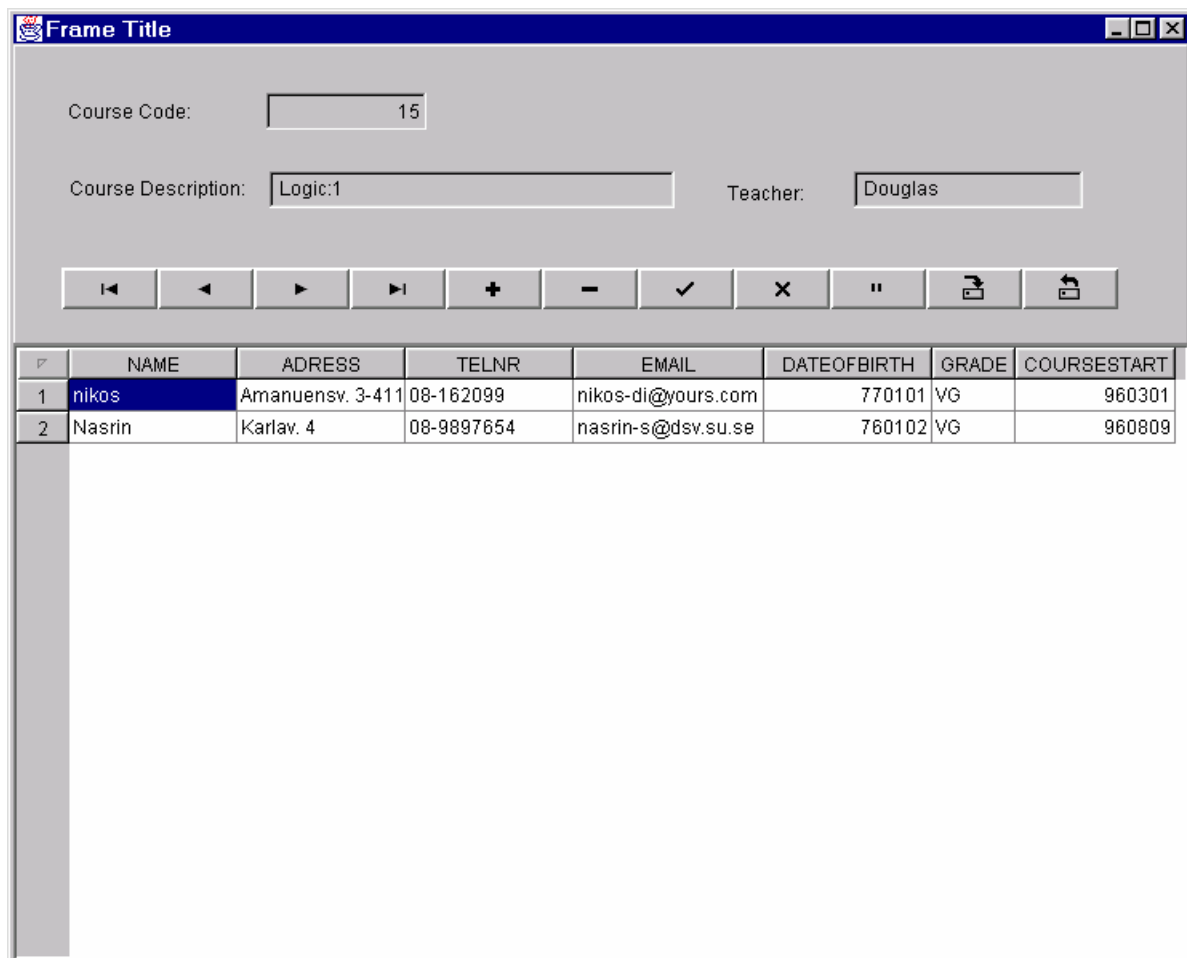


## 6.3 Master-Detail

In this exercise we will use all the tables of our database. We will show for one course at a time:

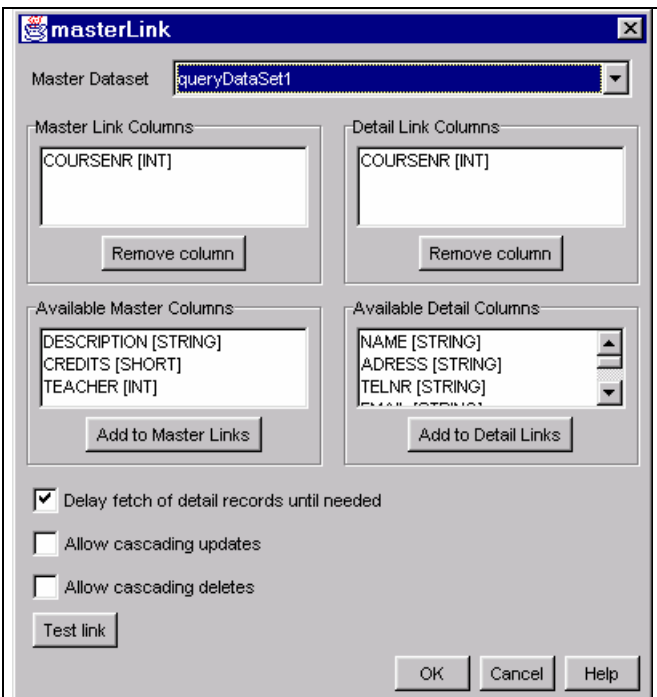
- the course's code and description (courses.coursenr and courses.description)
- the course's teacher's name (teachers.name)
- all the students that have attended the course (students.\*)

Our frame will look like this:



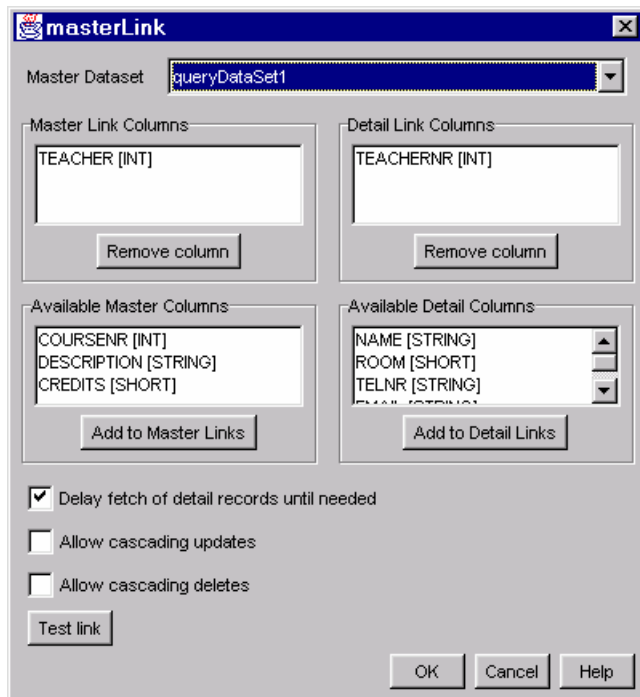
- Create a new application
- Place a gridControl, a navigatorControl, textControls and fieldControls on the frame
- Add a database and three queryDataSets to the frame
- Connect the database to the ODBC:school connection (see exercise 6.2)
- First queryDataSet's SQL: **Select \* from courses**

- Second queryDataSet's SQL: **Select \* from studentcourse as sc, students as s where s.studentnr = sc.studentnr**
- Third QueryDataSet's SQL: **Select \* from teachers**
- Activate the second queryDataSet and show the masterLink wizard by clicking the ellipsis next to the masterLink property. Set the values as in the figure that follows:



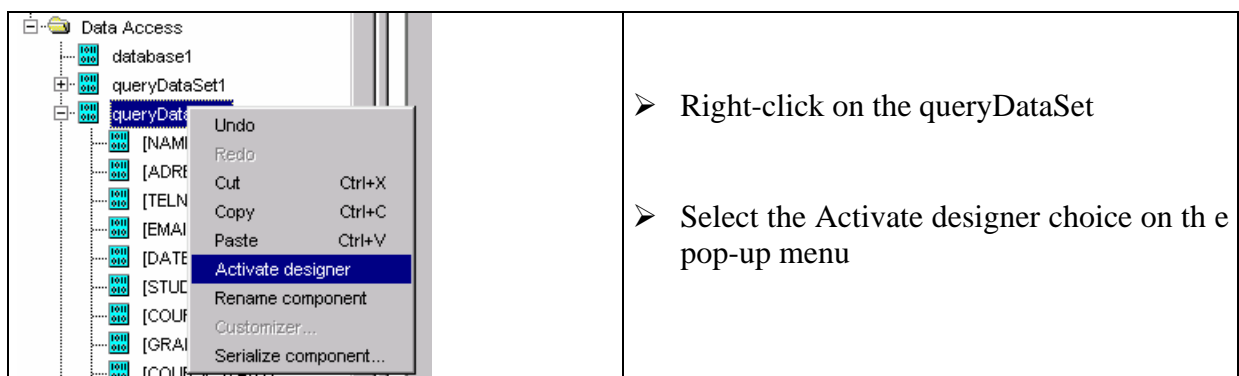
- Select the first QueryDataSet as the Master Dataset
- Select the **Coursenr** as the link column
- Test the link by clicking the test link button
- Press OK

- Do the same for the third queryDataSet according to the next figure

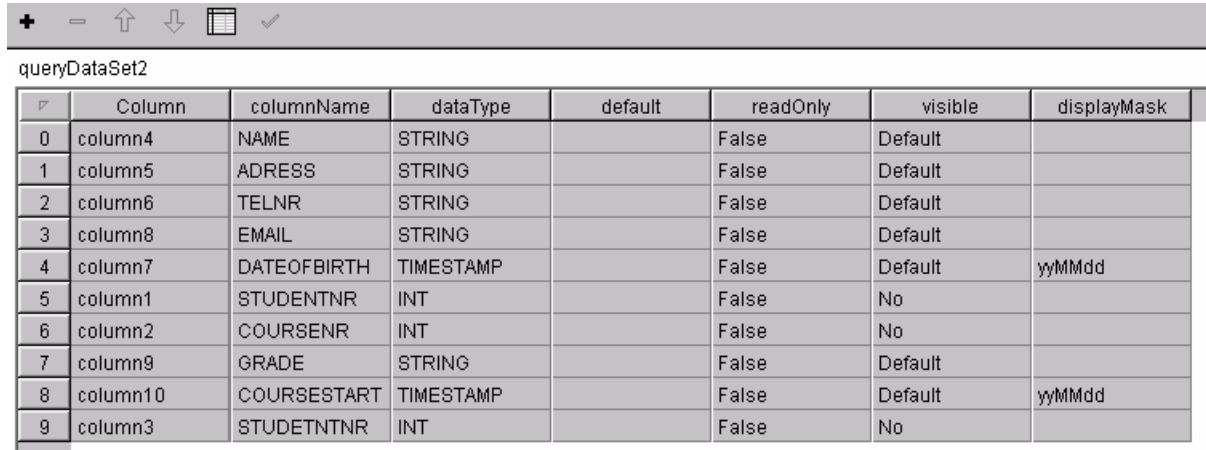


- Connect the gridControl to the second queryDataSet (set the dataSet property of the gridControl)
- Connect the navigator control to the first queryDataSet
- Connect the fieldControls to the course's code and description (courses.coursenr and courses.description) and the course's teacher's name (teachers.name) by setting the dataSet property and the columnName property.
- To make all the fieldControls and queryDataSets read only set their readOnly property to **true**



Some of the columns of the second queryDataSet (that is shown in the gridControl) may not be important to be shown in the gridControl. It may also be needed to change the order that the columns are displayed or even the way the data is displayed (for example a date may be displayed with different masks). To define all these properties you can use the "designer" for the queryDataSet that you want to manipulate:



The content pane shows now a table with all the columns of the queryDataSet and their properties (the selected ones):



	Column	columnName	dataType	default	readOnly	visible	displayMask
0	column4	NAME	STRING		False	Default	
1	column5	ADRESS	STRING		False	Default	
2	column6	TELNR	STRING		False	Default	
3	column8	EMAIL	STRING		False	Default	
4	column7	DATEOFBIRTH	TIMESTAMP		False	Default	yyMMdd
5	column1	STUDENTNR	INT		False	No	
6	column2	COURSENR	INT		False	No	
7	column9	GRADE	STRING		False	Default	
8	column10	COURSESTART	TIMESTAMP		False	Default	yyMMdd
9	column3	STUDETNTNR	INT		False	No	

- Use the  and  buttons to change the order of the columns
- Write the following code in the navigated event of the first queryDataSet<sup>13</sup>:

```
try {  
    queryDataSet2.refresh();  
}  
catch (Exception ex) {  
}
```

- Save and run the application

## 6.4 Filtering Data - Calculated Fields

In this exercise we will use calculated fields, a calculated field is created at run time by the program and it is not part of the database. A calculated field appears for example as an extra column in a queryDataSet. In this exercise we will create a calculated field which will be a concatenation of three other fields with some extra characters.

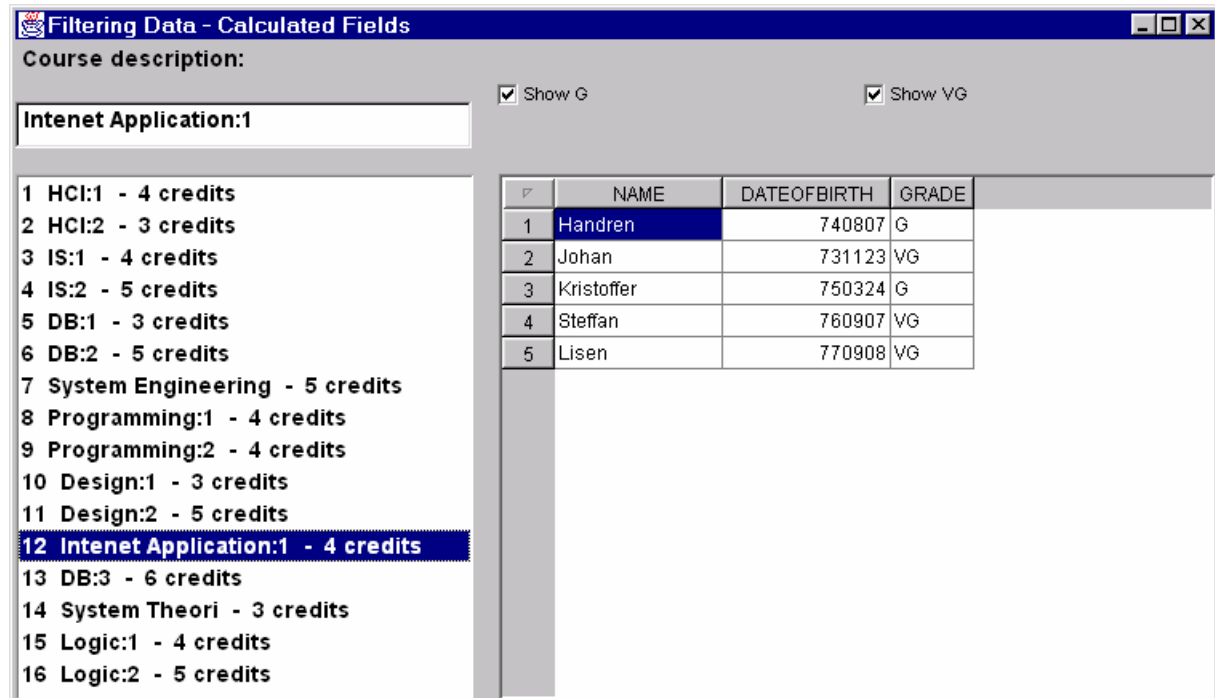
Further more we will try to navigate through a queryDataSet by using a locatorControl component. A locatorControl is similar to a textField but it is connected to a dataSet and to a column of the dataSet. A locatorControl performs a best-match locate (within the specified column) after each character is entered.

In addition to all that we will try to filter the result of a queryDataSet by using a couple of checkBoxes and the filterRow event of the queryDataSet.

---

<sup>13</sup> These lines of code are just to ensure that every row of the query is shown once. If you try to run the application without this code it is possible that you see the same row of the query multiple times in the gridControl.

The frame will hopefully look like this:



- Start by creating a new application
- Add a database component and two queryDataSet components to the frame
- Connect the database to the school database
- QueryDataSet1 SQL: **Select \* from courses**
- QueryDataSet2 SQL: **Select \* from studentcourse as sc, students as s where s.studentnr = sc.studentnr**
- Set the queryDataSet1 to be the master of the queryDataSet2 with coursennr as the link
- Set both queryDataSets to be read only
- Activate the designer of the queryDataSet1
- Add a new column and edit the properties of the columns to be like this (the column "Column" is filled by JBuilder and it not significant to us):

queryDataSet1

☐	Column	columnName	dataType	visible	readOnly	calcType
0	column11	COURSENR	INT	Default	True	not calculated
1	column12	DESCRIPTION	STRING	Default	True	not calculated
2	column13	CREDITS	SHORT	Default	True	not calculated
3	column14	TEACHER	INT	Default	True	not calculated
4	column15	ALLINFO	STRING	Default	True	calculated

- Write the following code at the calcFields event of the queryDataSet1:

```
calcRow.setString("ALLINFO", changedRow.getInt("COURSENR")+ " " +
    changedRow.getString("DESCRIPTION")+ " . " +
    changedRow.getShort("CREDITS")+ " credits");
```

This code calculates, for each row of the queryDataSet1, the value of the ALLINFO column<sup>14</sup>.

- Activate the designer of the queryDataSet2
- Edit the properties of the columns (and even their order) to be like this:

queryDataSet2

☐	Column	columnName	dataType	visible	readOnly	displayMask
0	column6	NAME	STRING	Yes	True	
1	column9	DATEOFBIRTH	TIMESTAMP	Yes	True	yyMMdd
2	column4	GRADE	STRING	Yes	True	
3	column1	STUDENTCOURSE.STUDENTNR	INT	No	True	
4	column2	COURSENR	INT	No	True	
5	column3	COURSESTART	TIMESTAMP	No	True	
6	column5	STUDENTS.STUDENTNR	INT	No	True	
7	column7	ADRESS	STRING	No	True	
8	column8	TELNR	STRING	No	True	
9	column10	EMAIL	STRING	No	True	

- Add the following GUI components with properties to the frame:

component name	property name	property value
textControl1	text	<b>Course description:</b>
locatorControl1	dataSet	<b>queryDataSet 1</b>
	columnName	<b>DESCRIPTION</b>
listControl1	dataSet	<b>queryDataSet 1</b>
	columnName	<b>ALLINFO</b>
	readOnly	<b>True</b>
jCheckBox1	text	<b>Show G</b>
	selected	<b>True</b>

<sup>14</sup> A column with that name and of calcType:calculated has to be predefined for the queryDataSet.

jCheckBox2	text	Show VG
	selected	True
gridControl1	dataSet	queryDataSet 1
	readOnly	True

- To connect the jCheckBox1 and jCheckBox2 to the queryDataSet2 add the following code to the filterRow event of the queryDataSet2:

```
if ((row.getString("grade").equals("G") && jCheckBox1.isSelected()) |
    (row.getString("grade").equals("VG") && jCheckBox2.isSelected()))
{
    response.add();
}
```

if (the row's grade = "G" AND  
jCheckBox1 is selected) OR  
(the row's grade = "VG" AND  
jCheckBox2 is selected)  
then  
add this row to the result

- To initiate a refiltering of the records every time the jCheckBoxes are selected or deselected add the following code to the itemStateChanged event of the jCheckBoxes:

```
try {
    queryDataSet1.refilter();
    queryDataSet2.refresh();
}
catch (Exception ex) {
}
```

- Add the following code to the selectionChanged event of the listControl1:

```
try {
    queryDataSet2.refresh();
}
catch (Exception ex) {
}
```

- Run the application

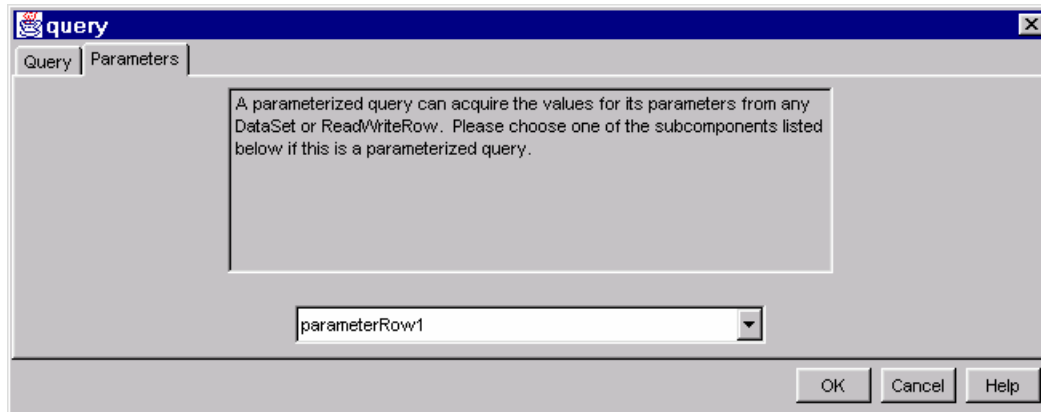
## 6.5 Using parameterized queries

This exercise, even though the last, is very small and simple. The only thing we will do is connect an SQL parameter to a textField.

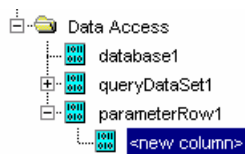
- Start a new application
- Add a database, queryDataSet and a parameterRow to the frame
- Edit the query's SQL and write the following:

**Select name, adress, telnr, email from students where (name=:name)**

- In the Parameters tab select the parameterRow1:



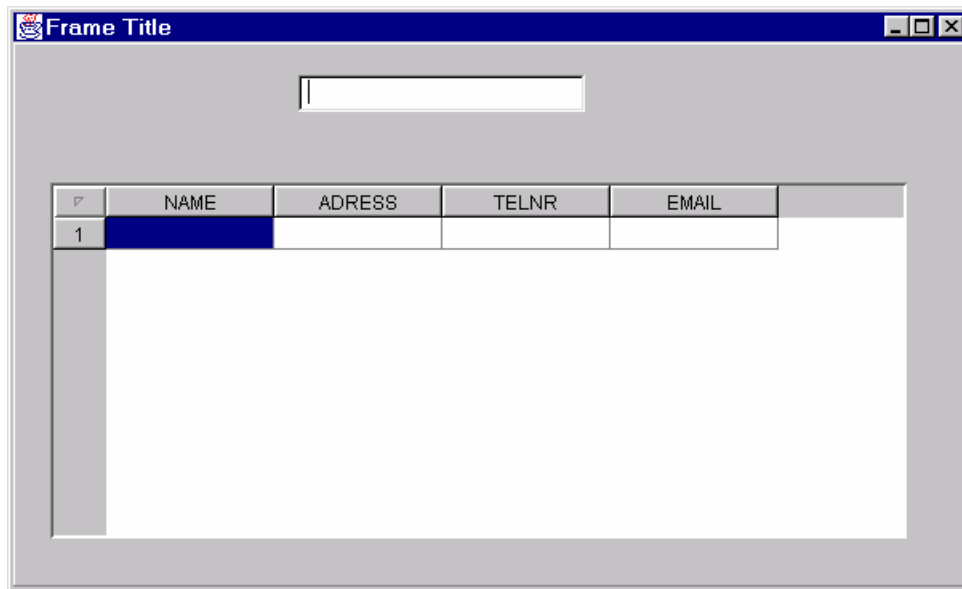
- Press OK to close the dialog
- Activate the <new column> under the ParameterRow1 on the Component tree:



- Change its columnName property to “name”
- Now add a gridControl and a textfield to the frame
- Make the gridControl1 read only
- Connect the gridControl1 to the queryDataSet1

The frame should look like this:





The only thing missing is the connection between the `textField1` and the `name`-column of the `parameterRow1`.

- Add the following code at the `keyPressed` event of the `textField1`

```
if (e.getKeyCode() == KeyEvent.VK_ENTER) {           If ENTER was pressed then
    try {
        parameterRow1.setString("name", textField1.getText());  set the parameter "name" to the current text of
        queryDataSet1.refresh();                                the textField1
    }                                                         refresh the queryDataSet1
    catch (Exception ex) {
    }
}
```

- Run the application

## 7 Epilog

JBuilder 2 has many more possibilities. There are more examples and solutions to various problems in the help files of JBuilder. Other useful sources of information regarding JBuilder are:

- Jbuilder newsgroups found at: [forums.inprise.com](http://forums.inprise.com)
- Jbuilders homepage found at: <http://www.inprise.com/jbuilder/>
- Other literature: JBuilder Essentials, *Jensen et al*, 1998 (JBuilder 1)