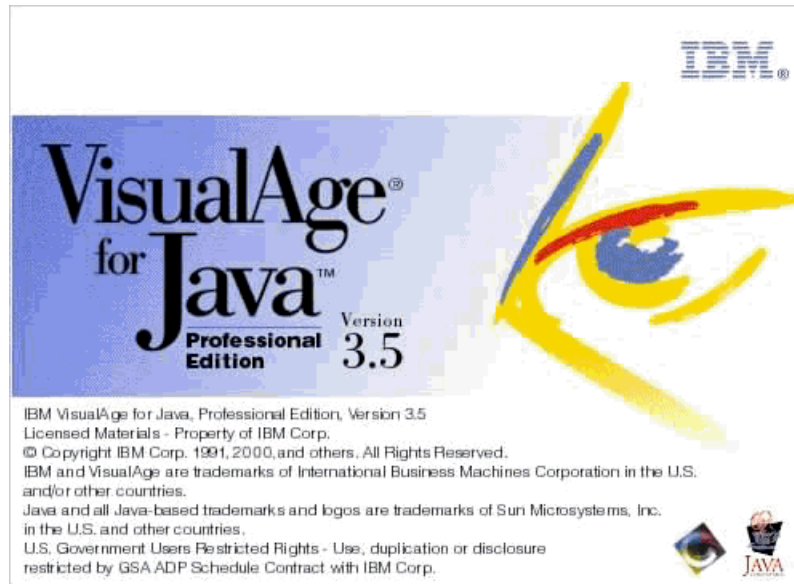

Using VisualAge for Java with the AS/400

Lab 27LF



By:

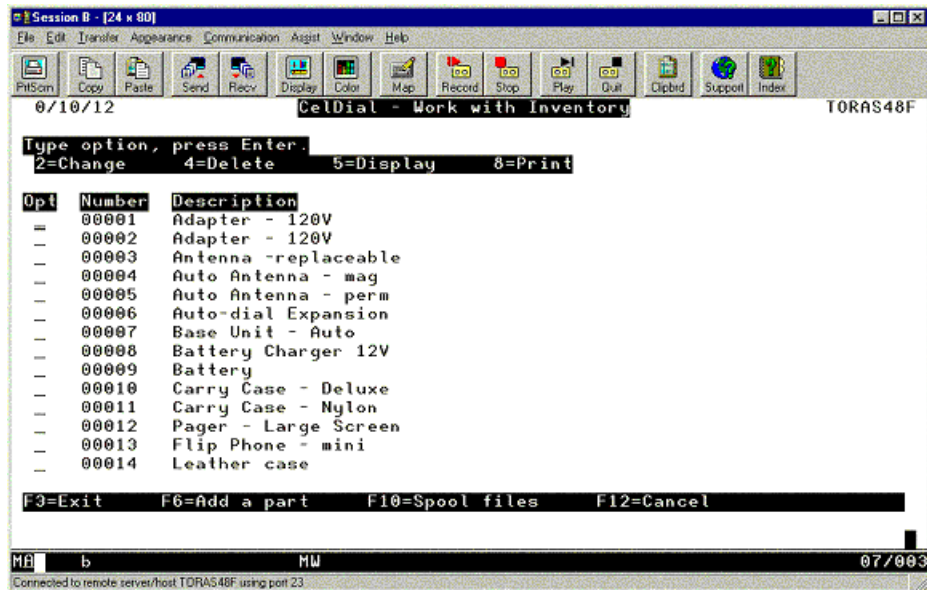
George Voutsinas

Goal:

- Create a Java application equivalent to the 5250 CellDial application.
- This application is found in your personal library, and can be invoked by typing "CALL WRKINV" on your 5250 emulation.

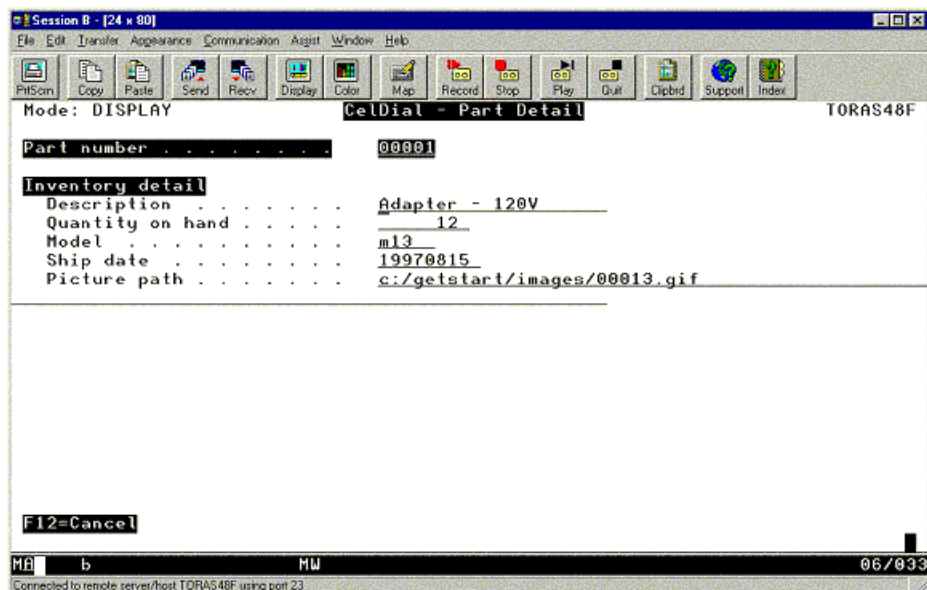
First Screen - "List View" (a subfile)

- The first screen of the 5250 application lists all the types of items in our inventory.

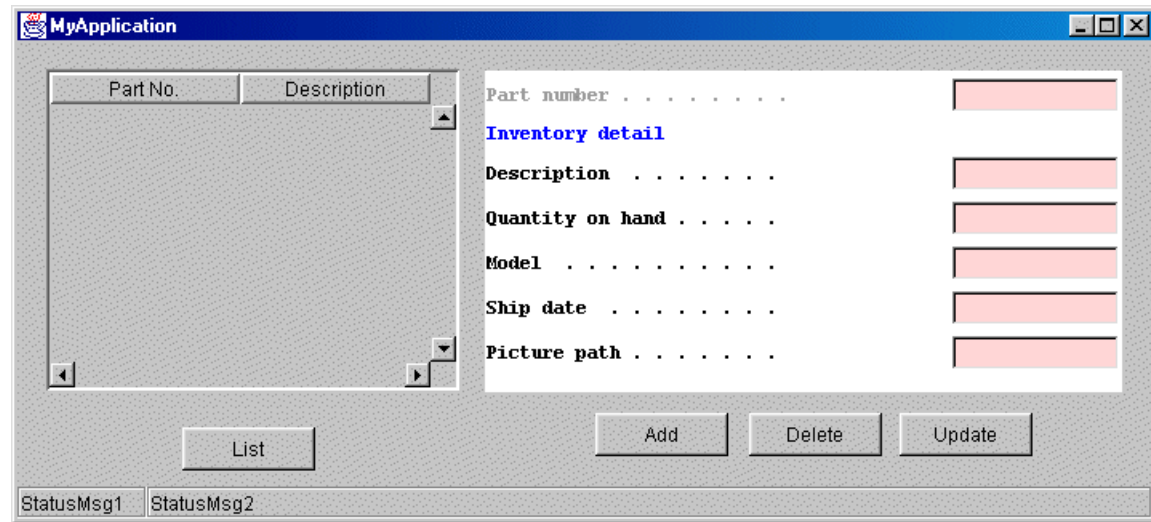


Second Screen - "Details View"

- The second screen of this application displays the details of the selected item from the "List View".



- The new Java application will be a merged version of these 2 screens, and will eventually look something like this.



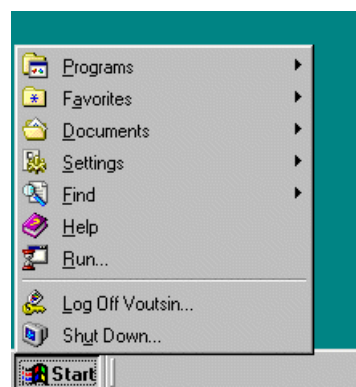
Step 1: Start VisualAge for Java

If you do not see the "Workbench", then we need to launch VisualAge Java.

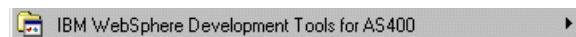
- Click on the "Start" button.



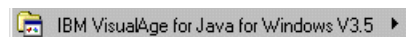
- Now select "Programs"



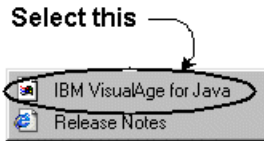
- Now select the AS/400 tools"



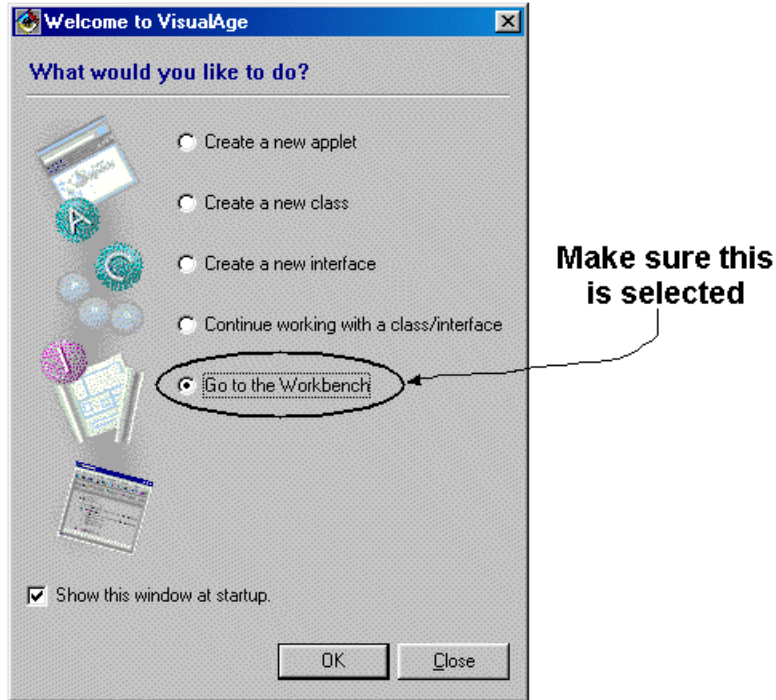
- Now select VisualAge for Java



- Now select the program



- Now select the "Workbench"



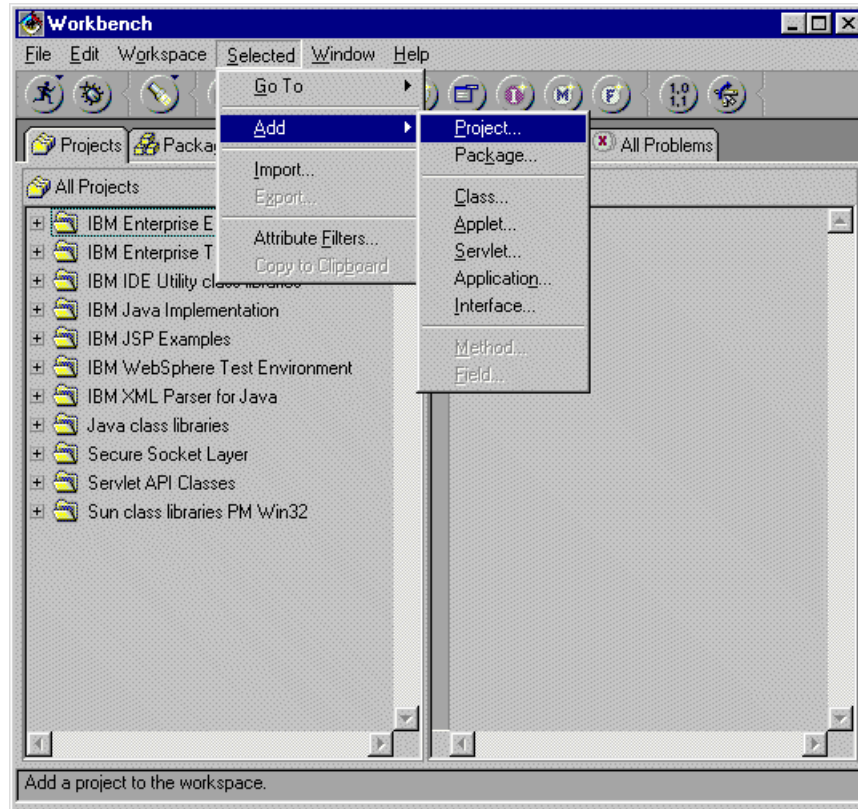
Step 2: Create a "Project"



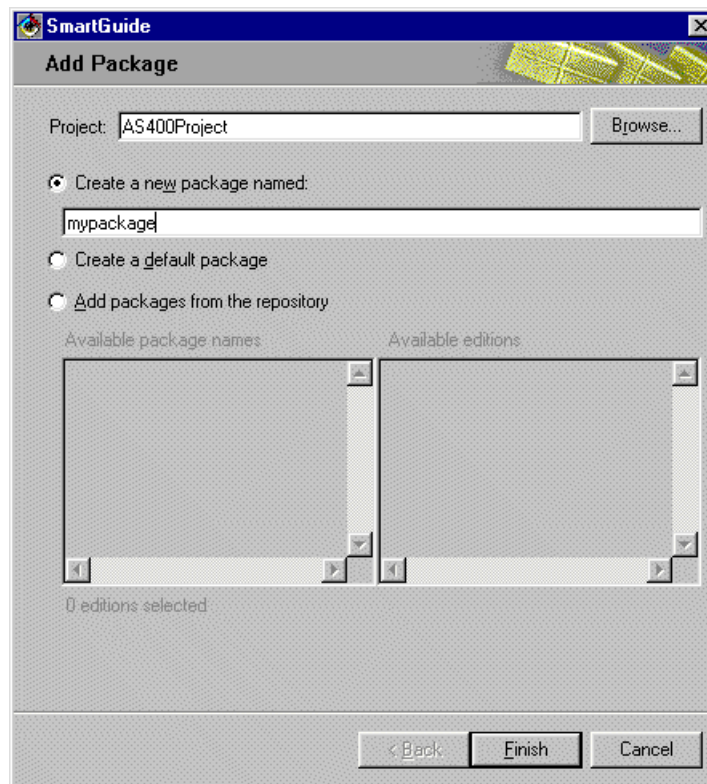
Project?

- Projects only exist inside "VisualAge for Java"
- They are not part of the Java language
- They are used to help organize Java code

- Select the "Add Project" Smartguide from the "Selected" menu item:



- Type in "AS400Project" and click on the "Finish" button.



- The project "AS400Project" should now be visible at the top of the list in the workbench.

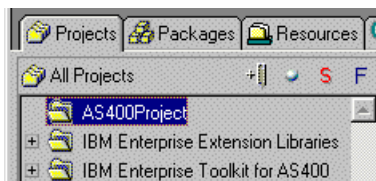
Step 3: Create a "Package"



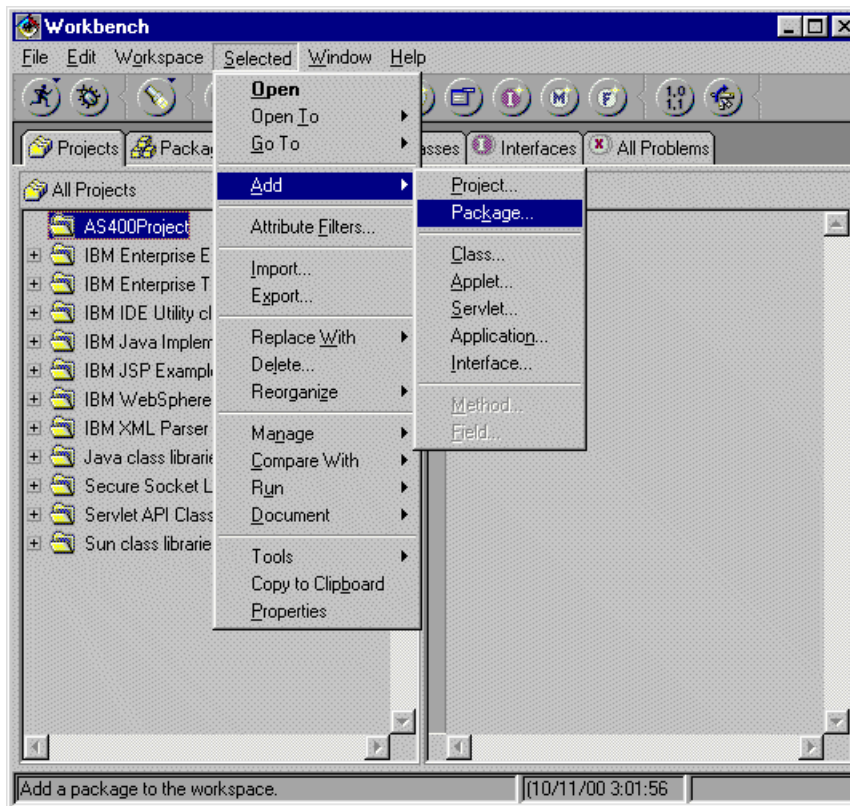
Package?

- Packages are part of the Java language
 - They are used to help organize Java code into collections
 - By convention, they are lower case names that represent the type of Java code contained within
 - They can contain "." (periods), but cannot start or end with a "."
 - Example: com.ibm.as400 (this is the official IBM package for AS/400 specific Java classes)

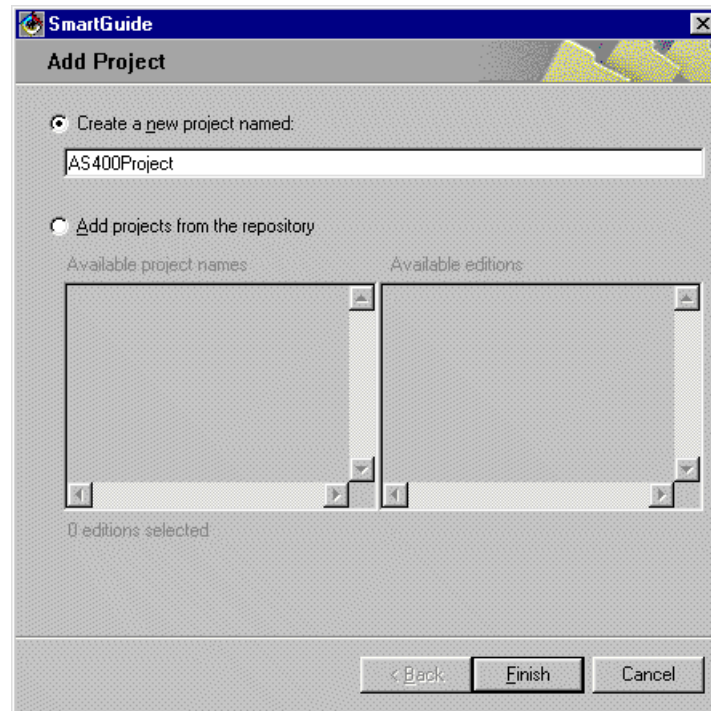
- Select the new project you just created (make sure its highlighted)



- Select the "Add Package" Smartguide from the "Selected" menu item



- Type in "mypackage" and click on the "Finish" button
- The package "mypackage" should now be visible inside the project



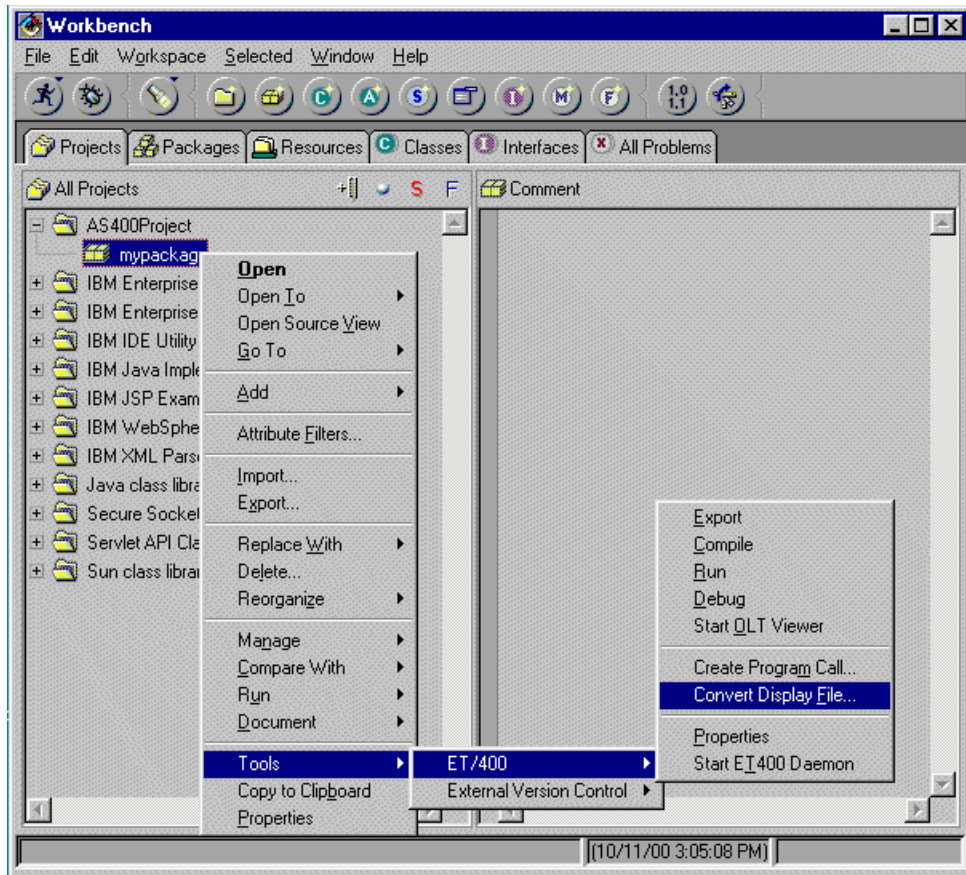
Step 4: Lets convert the "Details" view into Java

- Although this step can be done manually, lets instead take advantage of the AS/400 "**Convert Display File**" Smartguide.
- This Smartguide will convert a 5250 screen (DDS), into a Java Bean.

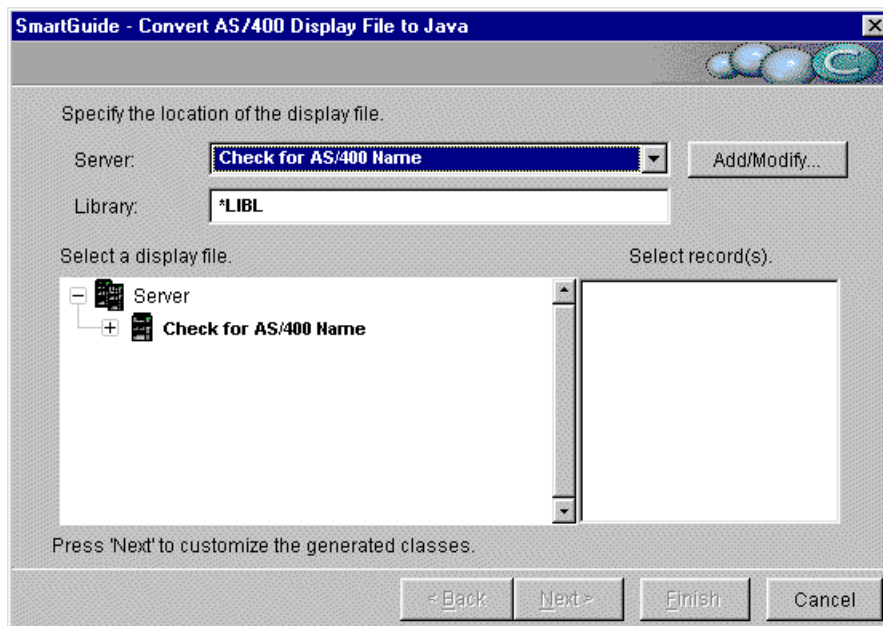


Java bean?

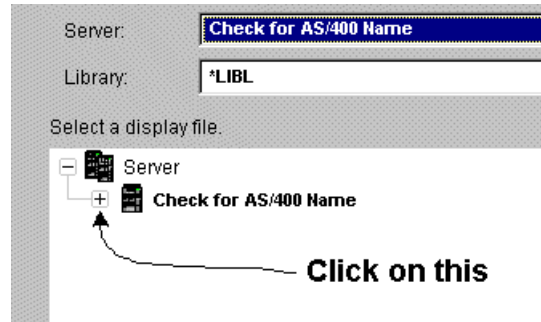
- A Java Bean is simply a Java class that has been coded in such a way, that a graphical tool (a GUI builder) can manipulate it.
 - This allows the user to draw the application instead of coding it.
 - Java Beans are part of the Java standard, and all Java tools can manipulate classes based on this standard.
- Select the package "**mypackage**" that you just created (make sure its highlighted).
 - Using the "RIGHT" mouse button, click on the package "**mypackage**".
 - Move the mouse pointer (see the picture below) down to "**Tools**", then to "**ET/400**" and click on "**Convert Display File...**" (BE PATIENT, this might take a minute or two)



- When the Smartguide appears, select the AS/400 assigned to you from the dropdown.

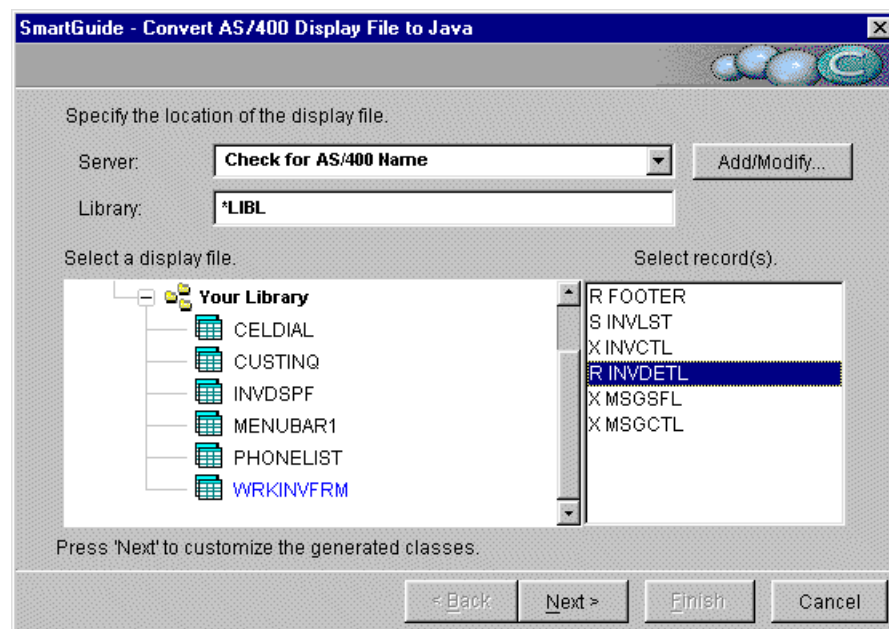


- Now that you have selected your AS/400, we need to get your library list, so click on the "+" symbol found next to your AS/400 (see below)



- By clicking on the "+", the Smartguide will attempt to connect to the AS/400 and retrieve the information required.
- In order to do this, a signon dialog will pop-up. (possibly 2)
- Type in the "userid" and "password" supplied to each terminal.

- Once you signon, you will now see a list of libraries appear below the AS/400. (this is your "Library List - *LIBL")
- Now click on the "+" symbol found next to your library (same name as your userid)
- By clicking on the "+", a list of DDS files will be displayed inside your library (see below)



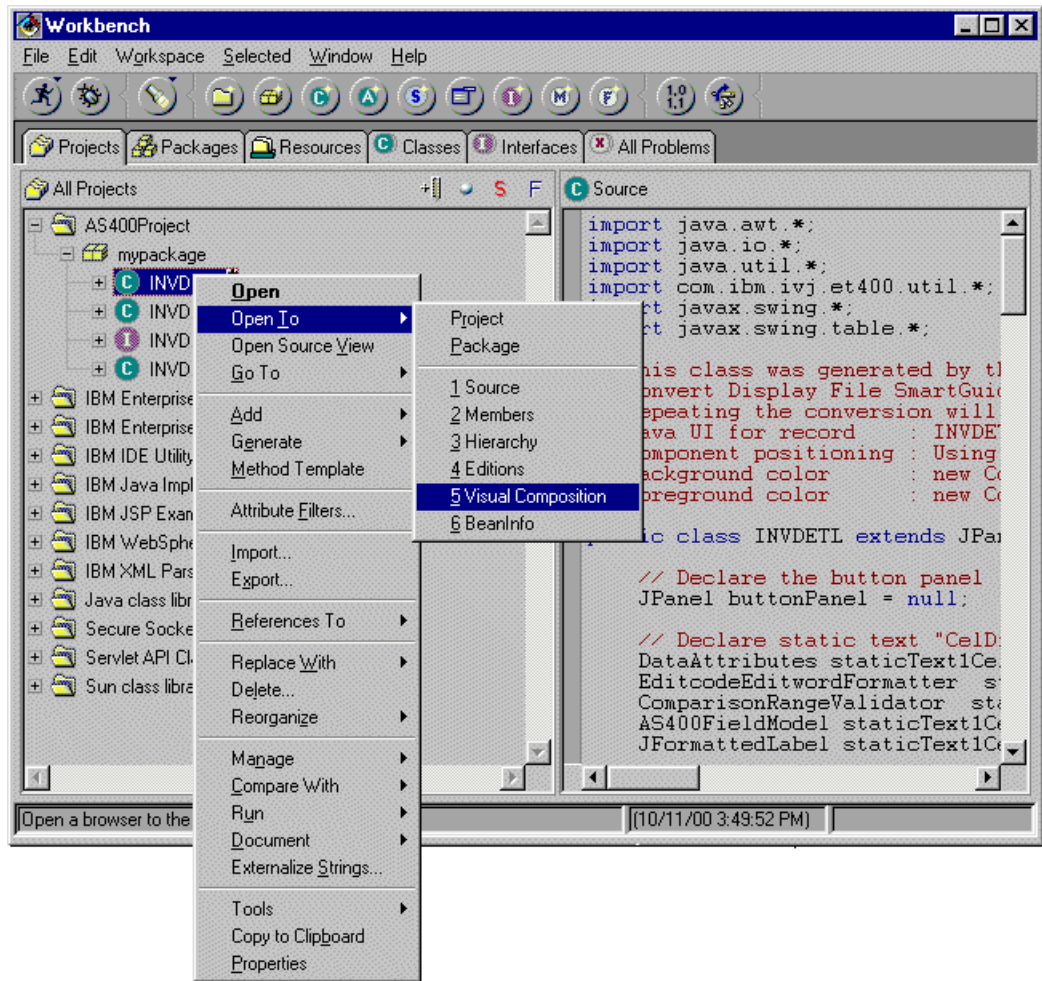
- Now select the DDS file "WRKINVFRM"
- A list of records will appear for the selected DDS
- Now select the record "INVDETL"

↑ see this image

- Once you've made the selection, click on the "Next" button 2 times.
- This will bring you to the last page of the Smartguide.
- Now click "Finish"

NOTE: When you click "Finish", the record will be converted and imported into the Workbench. This may take a few minutes.

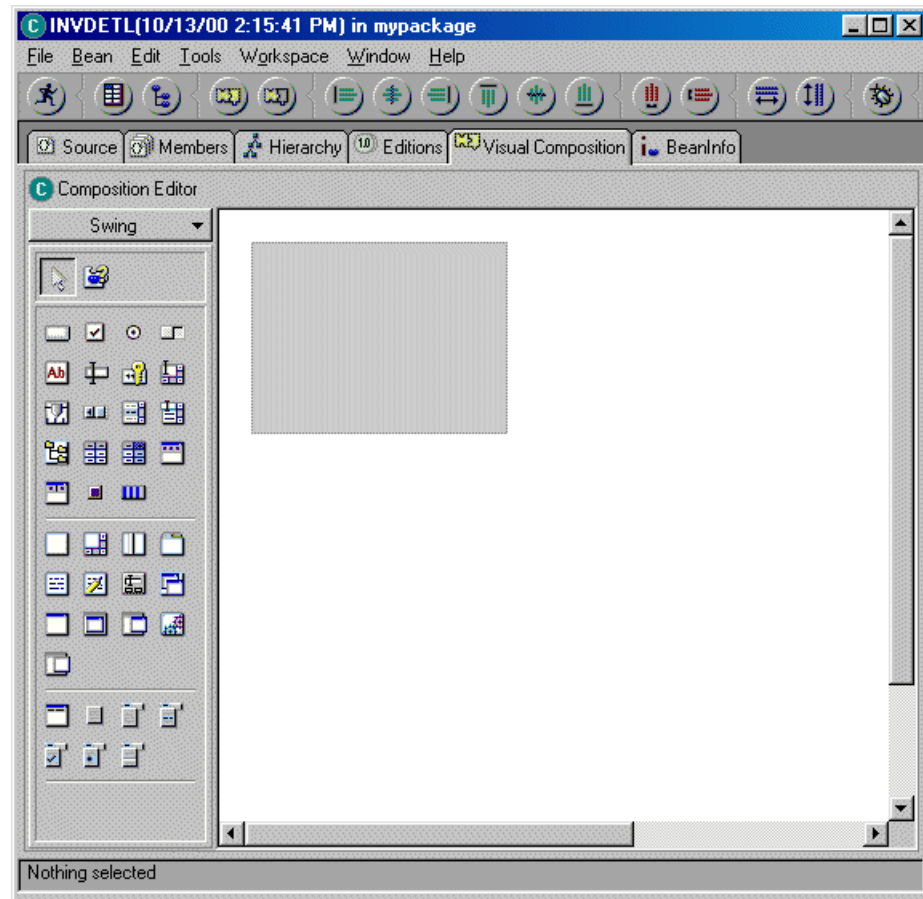
- Four new files can now be seen inside your package "mypackage"



- But before we can create our application, we need to modify the generated Beans.
- Select the file "INVD" (Note: the original text says "INVD" but the image shows "INVD" files)
- Using the "RIGHT" mouse button, click on the file "INVD".

↑ see this image

- Move the mouse pointer down to "Open To", and click on "5 Visual Composition"
- After you click on "5 Visual Composition", a new window will appear.
- This is the "Visual Composition Editor" (or VCE) which allows you to graphically create your application

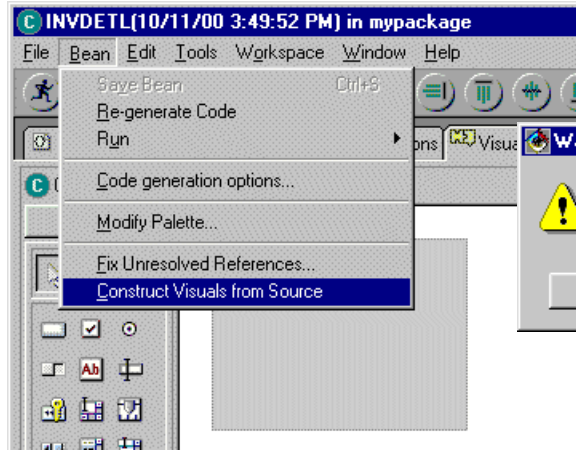


- You'll notice that the VCE only contains a small grey box.

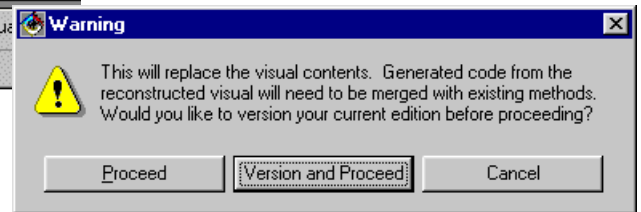


see this image

- This is because the builder has not been told to display our Bean with more detail.
 - In order to display more detail, the VCE needs to construct all the visual pieces based on the source the "**Convert Display File**" Smartguide generated.
- Lets tell the VCE to give us more detail
 - Select the menu item "Bean", and click on (see below) the option "**Construct Visual from Source**"



- As soon as you click on this option, a new dialog will pop-up

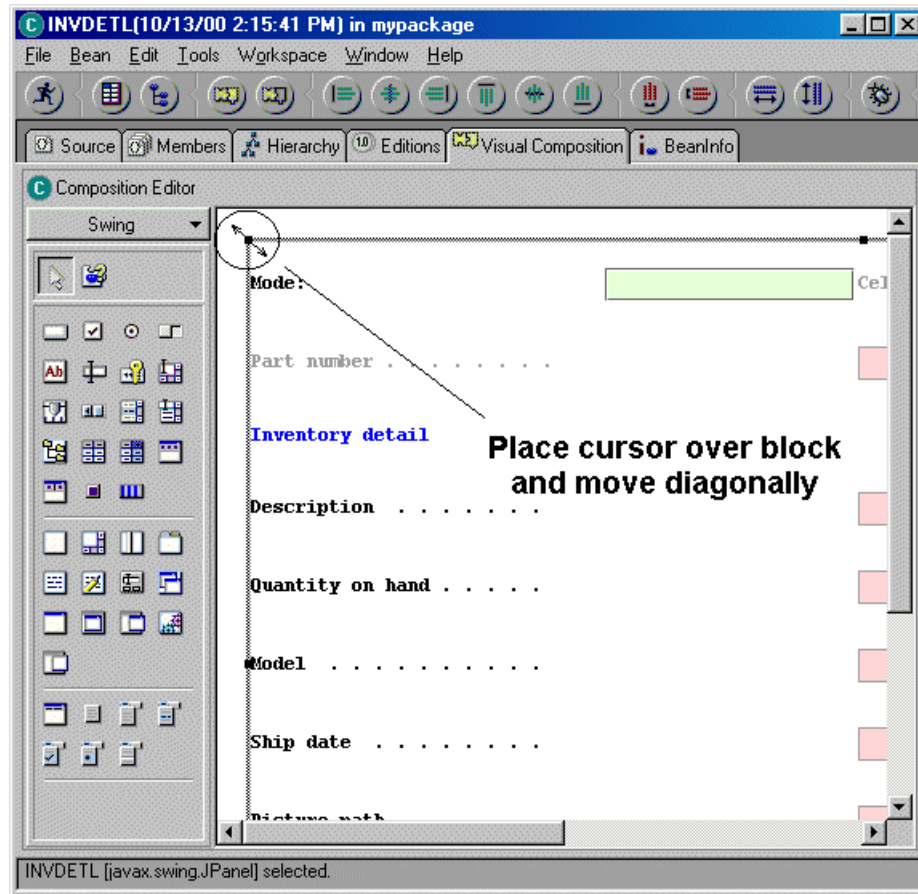


- This dialog is asking if you would like to version (basically save a copy) the current source of "INVDETL"
- For now click "Proceed"

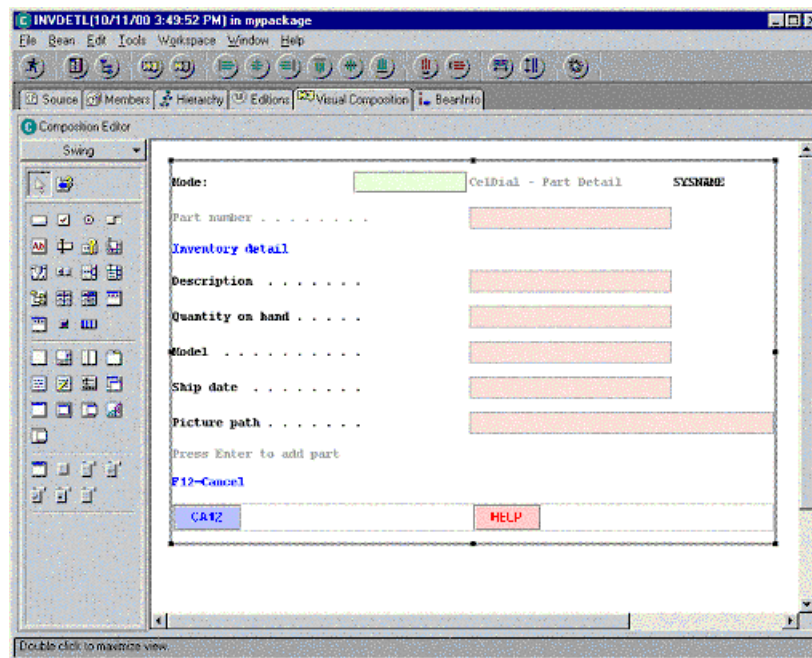


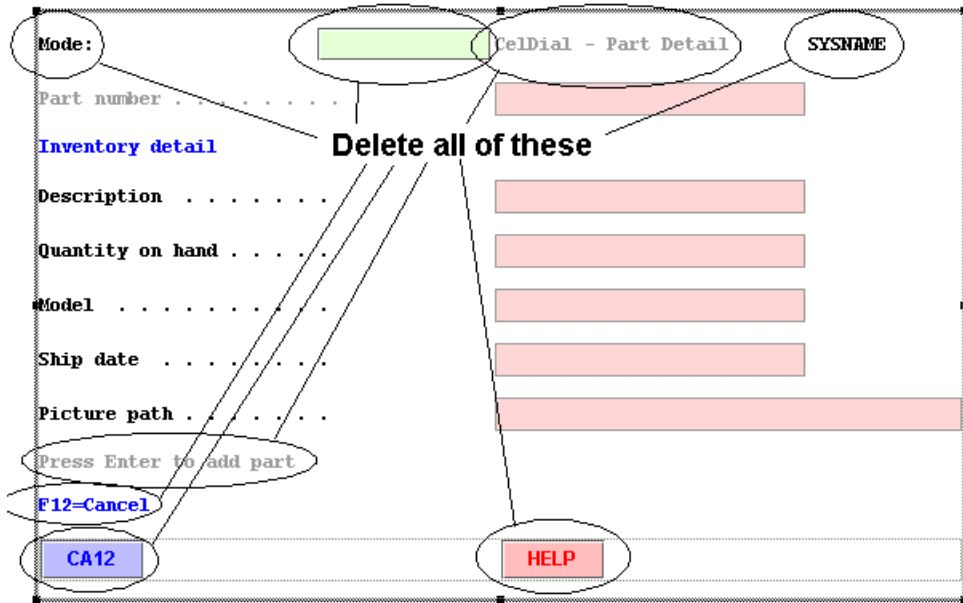
Versioning?

- VisualAge for Java has the ability to "version" code.
- This allows us to keep a backup copy of the code we are about to modify, without having to rename it.
- VisualAge for Java allows you to compare different versions of the same file. This can be handy if something stops working, but worked well in an older version.
- In a large team environment, you can give versioning authority to individuals or keep it for the Administrator (ie QSECOFR)..
- Feel free to ask for more info...
 - Once the construction is complete, the contents will be too large for our screen.
 - Re-size the generated image by placing your mouse pointer over the top left block, and moving down and to the right. (see below)



- Once you re-size the image it should look something like this (see below)
- Now that we have converted and shrunk the screen, we should also clean things up.

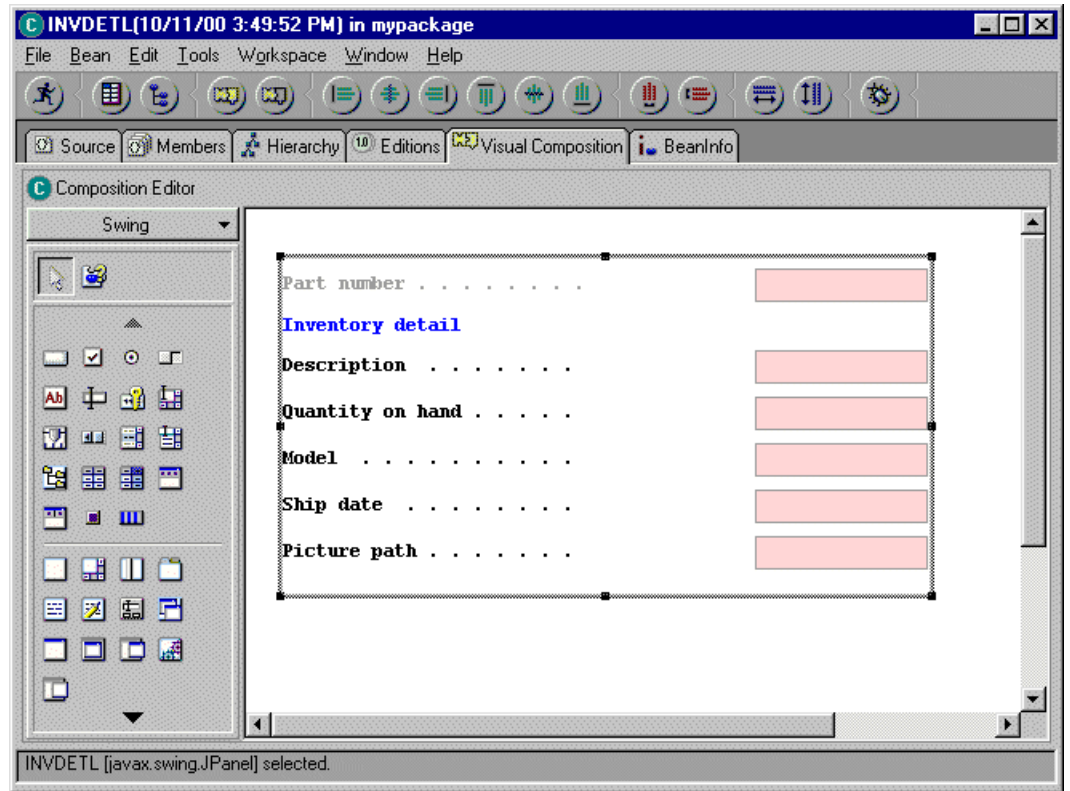




Since we will be providing our own buttons and titles, lets delete everything indicated by the following image.

↑ see this image

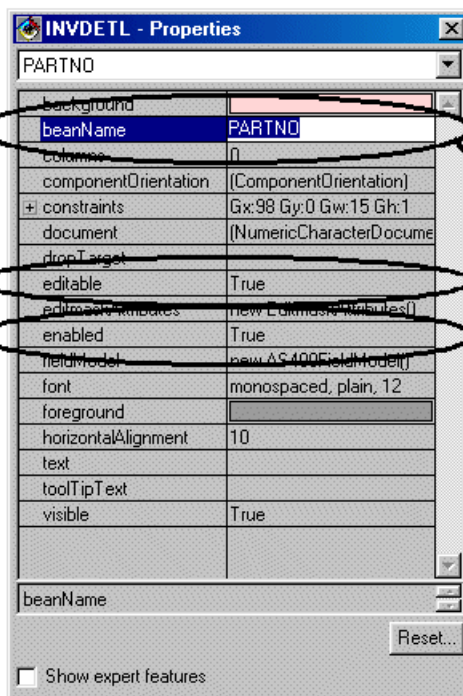
- To delete, click on each object, 1 at a time, and hit the "Delete" key.



- Once you have deleted all the extra objects, you should have a screen that looks like this

↑ see this image

- To make things a little easier for us later, we will also rename all the fields so that they represent the DDS "field" names, set the "enabled" and "editable" attribute to "true" (see table below)



Change these 3 settings for each "pink" field


- To modify the fields, **double-click** on one, and modify the "Property" window that pops up.

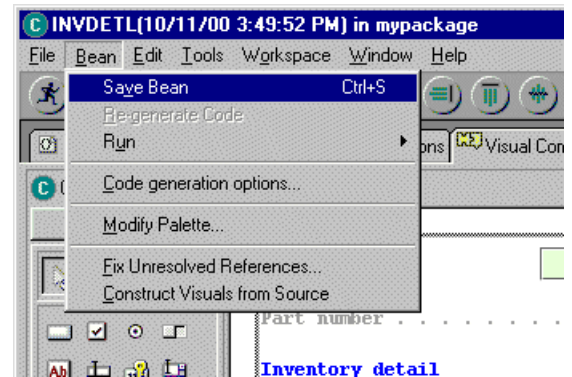
- The field who's property you are modifying will look like this



Field	Change "beanName" too	Change "enabled" & "editable" too
Part number	PARTNO	true
Description	PARTD	true
Quantity on hand	INVENTORY	true
Model	MODEL	true
Ship date	PARTSHIPD	true
Picture path	PARTPIC	true

- Once you have updated all the fields, close the properties window by clicking on the (top right corner)
- You have now finished modifying the INVDETL Bean.

- Lets make sure we save all our work, so select the file menu "**Bean**", and click on "**Save Bean**"
- Now close the VCE by clicking on  (upper right corner)



Step 5: Lets create our new application


- To help us create our new application, we will take advantage of the "**Create Application**" Smartguide

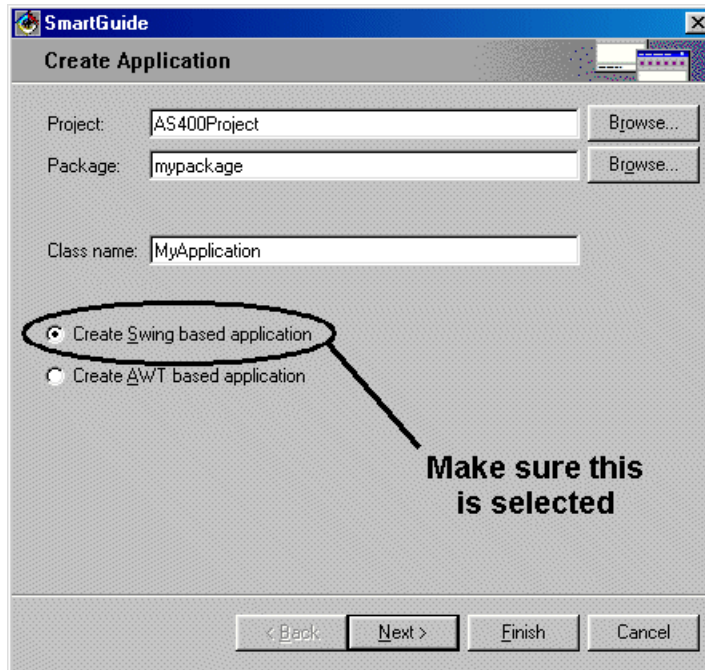


Create Application Smartguide?

- This Smartguide allows the user to create a small application with complex Java features pre-coded for you.
- Features like:
 - A splash screen - the image that pops up before the main window appears
 - Help menu.
 - A menu bar - where you would find items like "**File**", "**Edit**" etc
 - A tool bar - tool bars are a graphical representation of common tasks like "**Cut**", "**Copy**", "**Paste**"

(example: )

- Before we start this Smartguide, make sure your package "**mypackage**" is selected (ie. make sure its highlighted)
- To start this Smartguide, click on this button found on the Workbench: 



- You'll notice, that since you had selected your package "mypackage" that information is pre-filled. (Note: if it's not pre-filled, press cancel. Try again, making sure to select your package "mypackage")
- Now we'll fill in the "Class name".



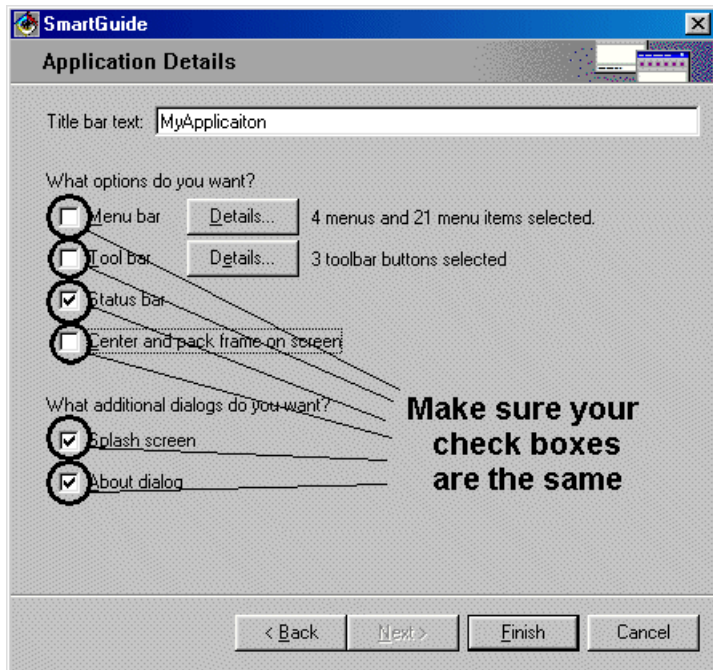
Class Name?

- In Java almost everything is a class, this includes applications
- An application is a class with a main entry point
- By Java convention, all classes start with a capital letter
- Type in the name "MyApplication"
- Make sure the "Swing" radio button is selected
- Click on the "Next" button to move to the last page of the Smartguide.



Swing or AWT?

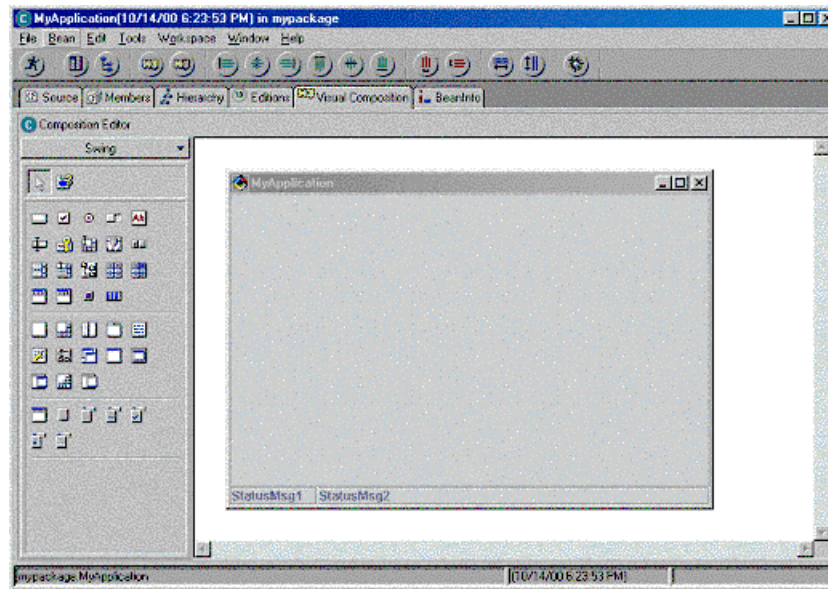
- Swing & AWT (Abstract Windowing Toolkit) are the graphical components of Java.
 - AWT was created first, and is the only one you can use with Applets (some newer web browsers support Swing, but not well)
 - Swing is newer, and much more robust. Support was added that catered to business applications, like tables (better know to us as subfiles).
- You should now be on the 2nd page of this Smartguide.





To keep things simple, uncheck the 3 options "Menu bar", "Tool bar", and "Center and pack frame on screen"

← see this image

- Now click on the "Finish" button
- **NOTE:** This is going to take a minute. The Smartguide is generating all the code, and will open the VCE (Visual Composition Editor)



- You should now see the VCE with the beginnings of the application we need to create.
- Before we go any further, lets run the application that was generated.
- Click on the "Run" icon . This will save everything and run the application
- Notice how the splash screen (the cow) appears before the application window appeared.
- Lets end the application, click on  (upper right corner) .

Step 6: Lets add the "List View" support to our new application

- The "List View" (see our Goals on page 2), is basically a subfile which displays a specific set of fields from a database.
- The subfile in Java terms is simply a "table" which is populated from a database.
- The idea of using a table to display information is not specific to the AS/400, but is used by most Java applications via JDBC.




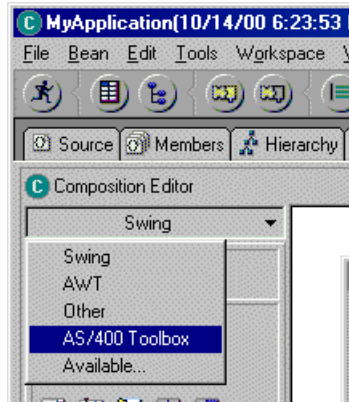
JDBC?

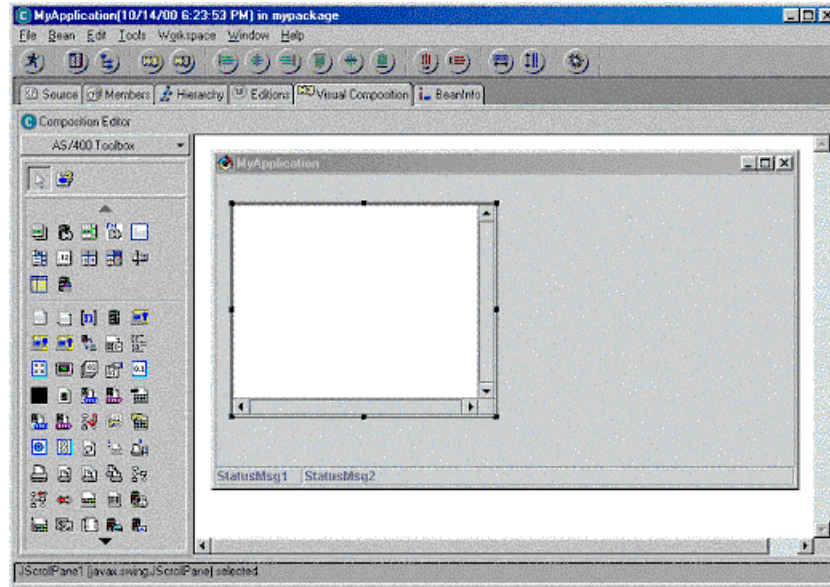
- JDBC (Java Database Connectivity) is a standard set of API's which are based on SQL for manipulating/accessing a database.
- By coding in JDBC, you can easily migrate your code to other platforms, since most databases comply with the SQL standard.
- Although the AS/400 database is SQL compliant, most RPG/COBOL/etc developers take advantage of record-level access instead.
 - Since most of us (AS/400 folks) manipulate databases with record-level access, we'll create our Java application using the same approach. (via the "AS/400 Toolbox for Java")
 - We'll create this "table" with the help of the **JFormattedTable** & **JFormattedTableColumn** beans.

- These beans are 100% Java, but have been created specifically for record-level access applications.
- Select the "Beans Palette" list, and click on "AS/400 Toolbox".

see this image →


- **NOTE:** This step may take a minute or two, since it is collecting all the icons for each bean.
- Select the **JFormattedTable** bean by clicking on its icon in the palette .

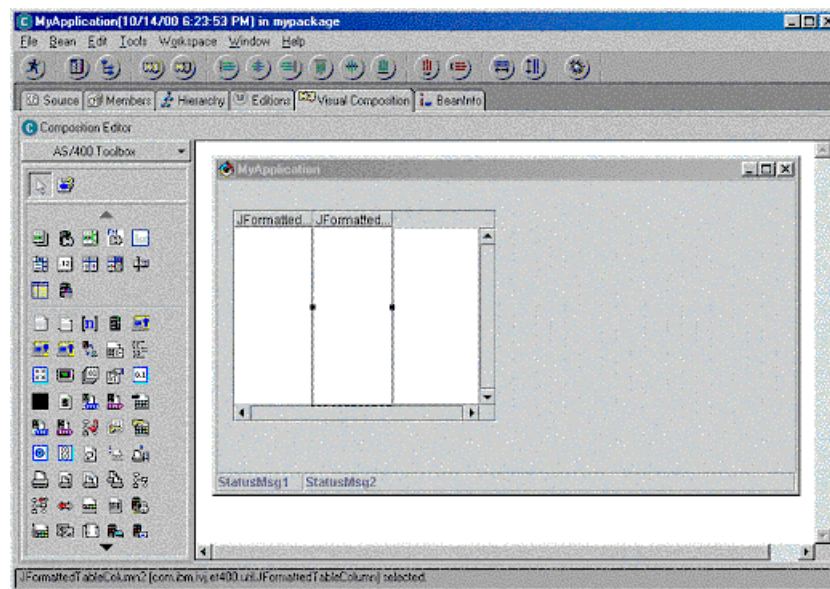




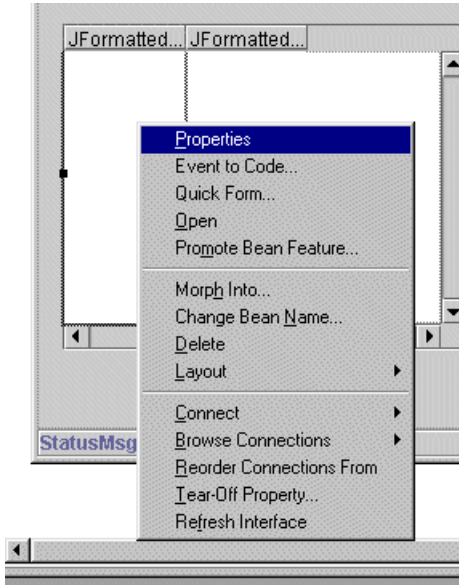
- Move your cursor, over our application. You'll notice that the cursor turns into a "+", indicating that you can drop this bean in that area.
- Re-size the "table" you just dropped by placing the cursor over a block and dragging the cursor. (same thing we did in step 4)
- Your VCE should look something like this:
- **NOTE:** ask for help if things look wrong

• Now lets add 2 columns to our subfile (ie table).

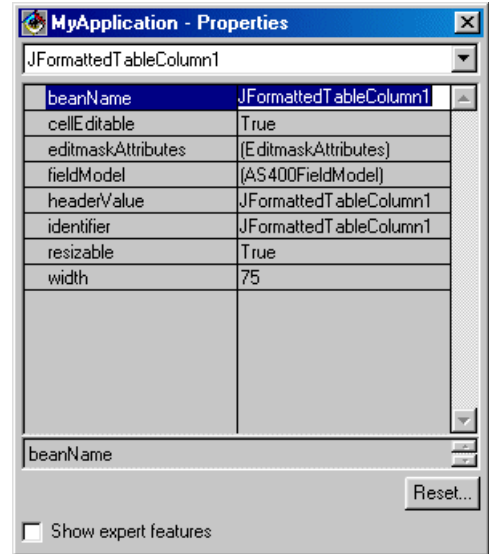
- Select the **JFormattedTableColumn** bean by clicking on its icon in the palette. 
- This time move your cursor onto the table, and once again, you'll notice your cursor change to a "+" symbol.
- Repeat this procedure one more time, so that you see 2 columns in the table




- We now need to define what fields in the database, each column represents.




- Right click on the first column, and select the "Properties" option
- This will open the properties dialog.
see this image →



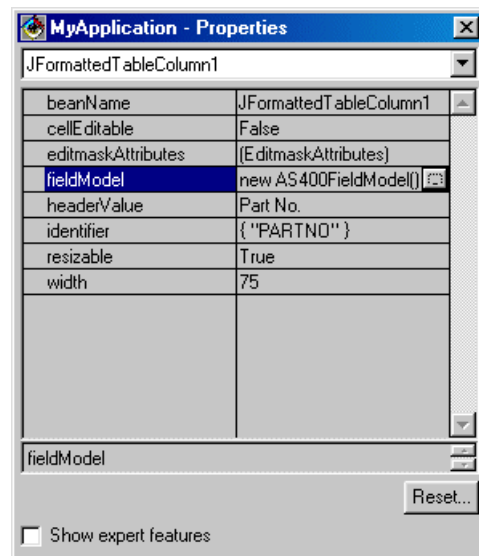
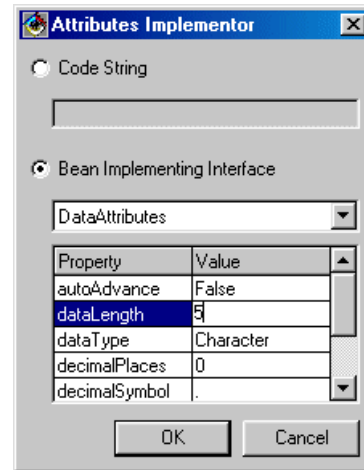
Properties?

- All beans have properties. They are predefined information used by the bean when it executes.
- Each property has a "set" & "get" method (in RPG terms, subroutine) which allows tools, like the VCE, to manipulate them.
 - Lets change the properties for the first column
 - Click on the field next to **identifier** and enter "**PARTNO**" (complete with quotation marks)
 - **NOTE:** We just defined what database field this column represents
 - Click on the field next to **headerValue** and enter "**Part No.**" (don't need quotes)
 - **NOTE:** We just gave the column a nice title. (you can use your imagination if you like)
 - Click on the field next to **cellEditable** and change it to **false**.
 - **NOTE:** Like our application in our Goals section, we'll only allow changes to data in the "**Details View**"
 - Click on the field next to fieldModel, you'll notice a button appear , click on it.

- In the new window that opened click on the field next to dataAttribute, again you'll notice a button appear as before , click on it.
- Click on the field next to dataLength, and enter the number 5.

see this image →

- **NOTE:** We just defined the size of the data to be displayed. Since PARTNO is a numeric field, we limit its size to 5 (similar to the subfile definition)
- Click on the OK button to close this dialog
- Click on the OK button again to close this dialog as well.
- **NOTE:** you should be back at the property window an it should look like this (see below)



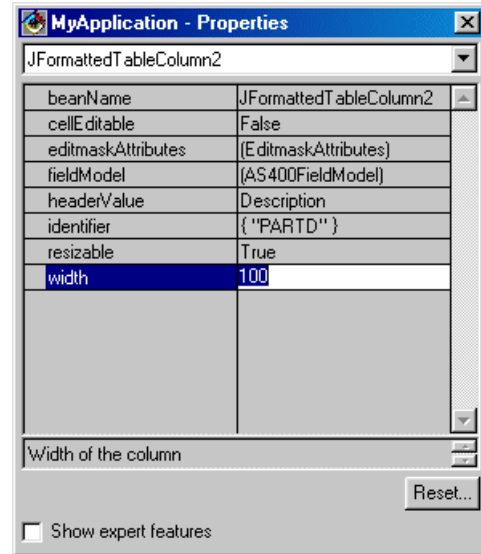
NOTE: You can leave the property window open, since all Java beans have properties. By Left-clicking on a bean, you'll see the properties for that object.

- Lets now change the properties for the second column.
- Left-click on the second column to select it. You'll notice that the properties window now displays the properties for this bean. (If you closed the property window, Right-click on the column, and select "Properties" to reopen it)

- Click on the field next to **identifier** and enter "**PARTD**" (complete with quotation marks)
- **NOTE:** We just defined what database field this column represents
- Click on the field next to **headerValue** and enter "**Description**" (don't need quotes)
- **NOTE:** We just gave the column a nice title.
- Click on the field next to **cellEditable** and change it to **false**.
- Click on the field next to **width** and enter **100**

see this image →

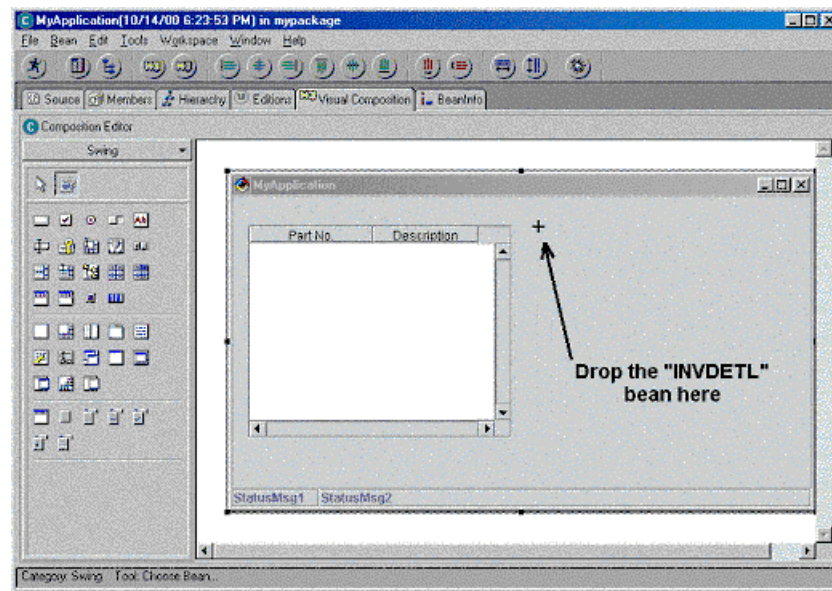
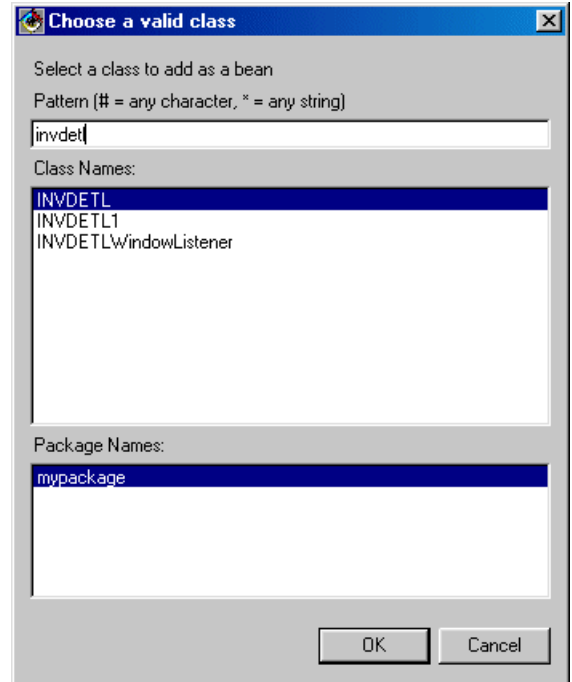
- Now close the properties window
- We have now finished adding the "**List View**" portion of our application



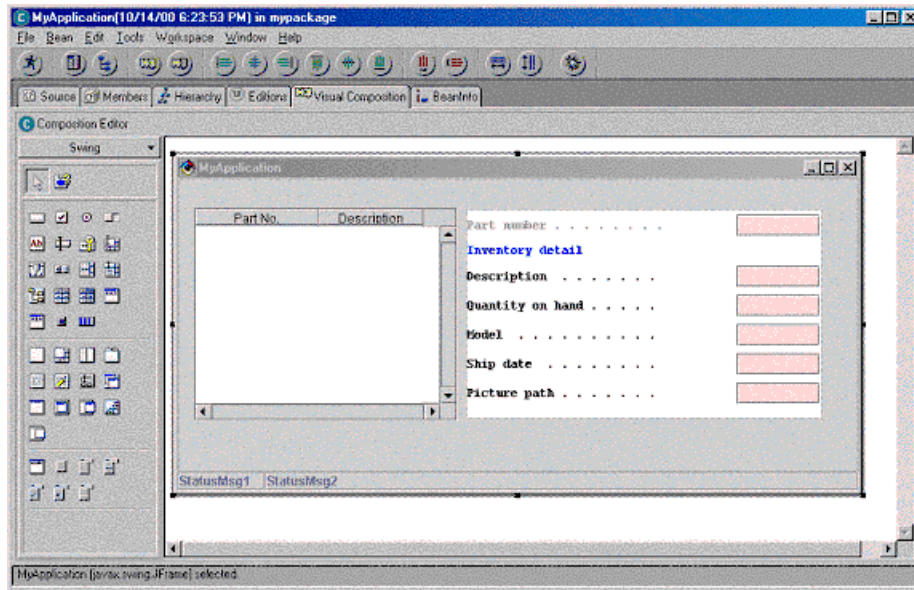
Step 7: Lets add the "**Details View**" support to our new application

- The "**Details View**" (see our Goals on page 2), is basically a set of fields that display specific information
- At this point you can create each field manually, or you can generate this view based on the existing DDS "**Details View**" as we did in **Step 4**
- Since we already converted the "**Details View**", lets use the bean we created.
- In the VCE, select the "**Choose bean...**" Icon from the palette. 🖱️
- Click on the "**Browse...**" Button

- In the window that opened, enter "invdetl"
- You'll notice that it automatically filters and finds all Java classes starting with these letters
- Also notice that it has highlighted the best match, which is the "INVDETL" bean, found in our package "mypackage"
- This was the bean we created in **Step 4** with the "Convert Display File..." Smartguide
- While this class is highlighted, click on the "OK" button
- You should now be back to the "Choose bean..." Prompt, click the "OK" button.
- You are now ready to place this bean onto our application.
- Move your cursor over the open area next to our subfile in our application, you'll notice the cursor turns into a "+", indicating that this is a valid location for this bean. (see below)



- Re-size the "INVDETL" bean and anything else you wish until your screen looks something like this



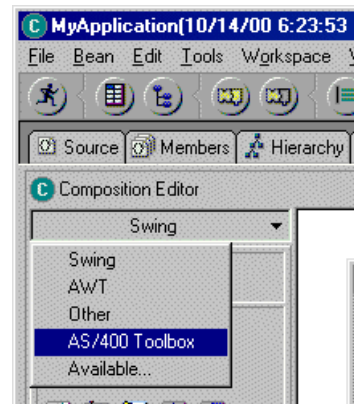
- The details view is now ready, now we need to hook the 2 views together and to the database on the AS/400.

Step 8: Lets add the record-level access beans

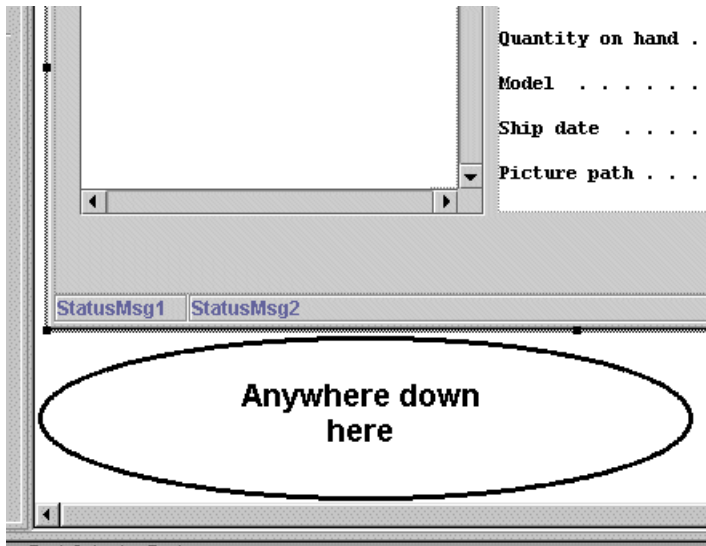
- A number of record-level access beans have been created for ease of use.
- These beans are 100% Java, but have been designed to take advantage of the AS/400 record-level access.
- Select the "Beans Palette" list, and click on "AS/400 Toolbox", if it has not already been selected.

see this image →



- Select the **ListManager** bean from the palette.

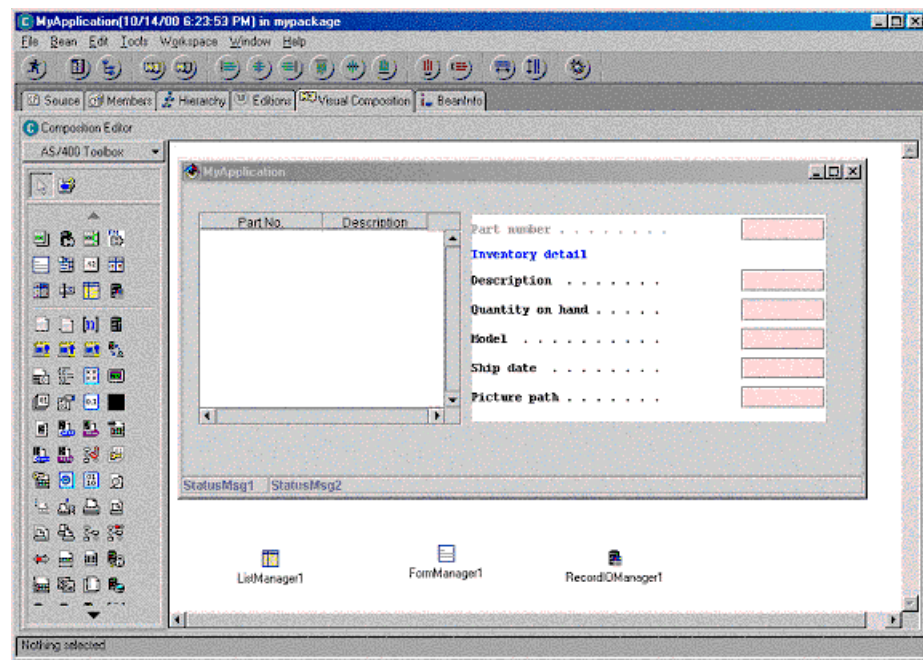


Move your cursor, below our application onto the open white space, and drop the bean.



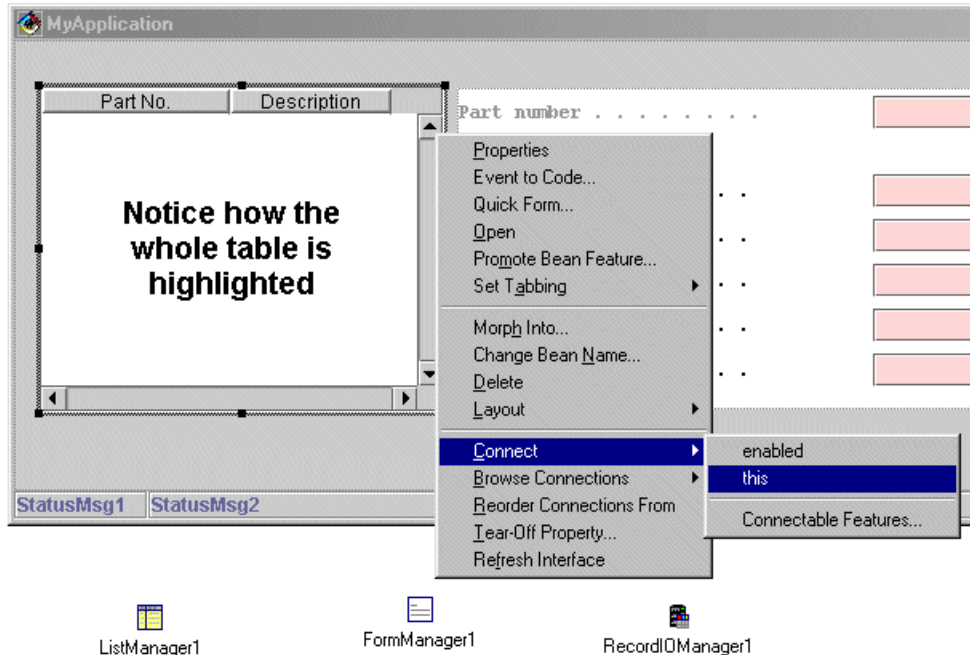
← see this image

- Repeat the above steps and add a FormManagerbean  and a RecordIOManagerbean  to the same open white space
- Your VCE should now look like this (see below)



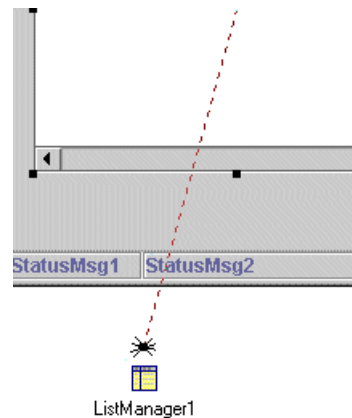
Step 9: Lets connect our subfile with its controller

- Select the table (ie subfile), by right-clicking on one of the scroll bars. (the whole table should be highlighted)
- Right-click on the highlighted table (make sure you are pointing at one of the scroll bars), move down to "**Connect**" and click on "**this**". (see below)



- You'll notice that your cursor has turned into a spider like object, with a trailing connection to the object you are coming from.

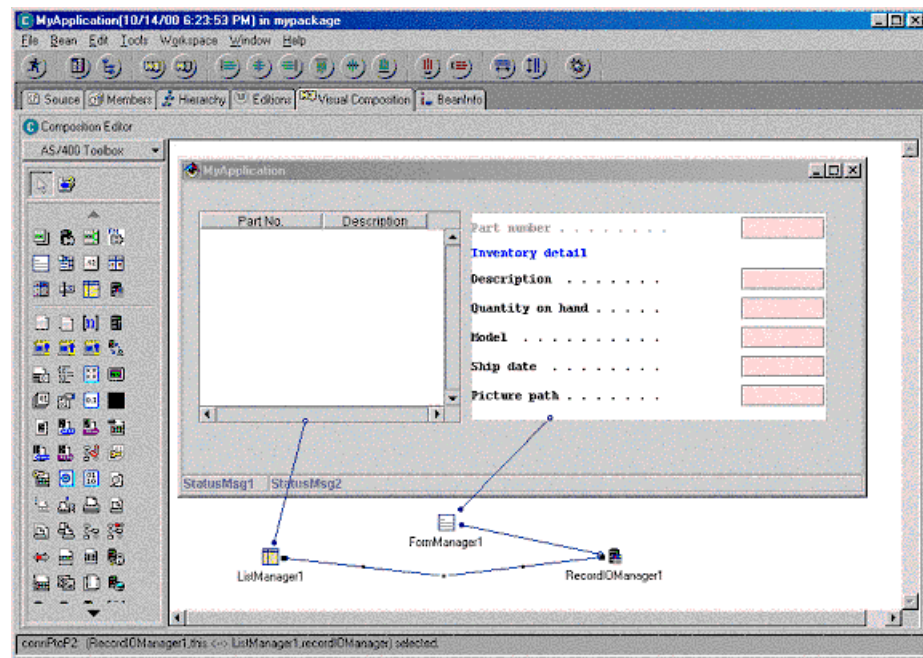
see this image →



- With your "spider", left-click on the "ListManager1" icon.
- In the pop-up menu that appears, select **displayContainer**.
- Congratulations, you've just made a property-to-property connection.
- **NOTE:** We just told our subfile controller, where to display the subfile information.
- Now select the **ListManager1** bean and right-click. In the pop-up menu that appears, move to "Connect" and click on **recordIOManager**.
- With your "spider", left-click on the **RecordIOManager1** bean. In the pop-up menu that appears, select "this".
- **NOTE:** We just told the **ListManager1** bean (which controls subfiles) where to get its data from. The **RecordIOManager1** bean is designed to open an AS/400 database, and do things to it.

Step 10: Lets connect our details view with its controller

- Select the **INVDETL** (our details view) bean and right-click. In the pop-up menu move to **"Connect"** and click on **"this"**
- With your **"spider"**, left-click on the **FormManager1** bean. In the pop-up menu that appears, select **"displayContainer"**.
- **NOTE:** We just told our **"Details View"** controller, where to display the information.
- Now select the **FormManager1** bean and right-click. In the pop-up that appears, move to **"Connect"** and click on **recordIOManager**.
- With your "spider", left-click on the **RecordIOManager1** bean. In the pop-up menu that appears, select **"this"**
- **NOTE:** We just told the **"Details View"** controller, where to get its data from.
- Your VCE should now look like this:

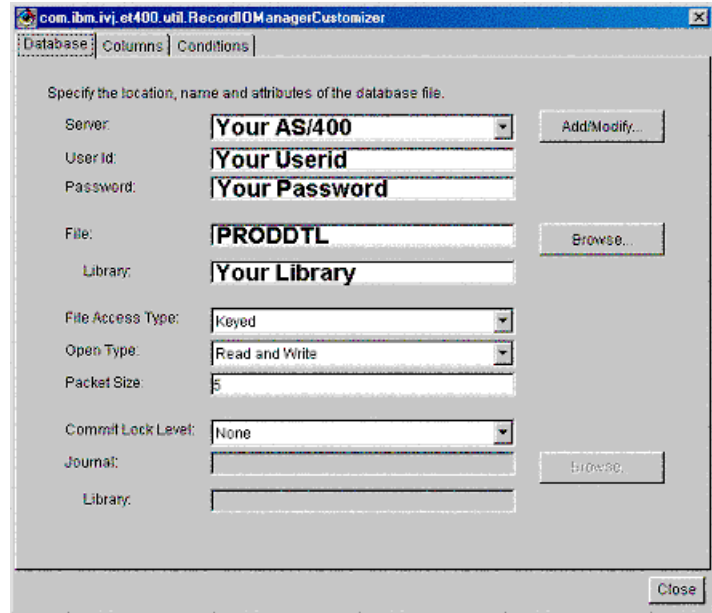


Step 11: Lets define where the database is

- Since both our "subfile" and our "details view" both depend on the same bean for information, we need to provide that bean with the location of the data.
- Using the left mouse button, double click on the **RecordIOManager1** bean.
- This is another technique for opening the properties dialog.
- Now click on the **"Custom Properties..."** Button located bottom right of the property dialog.

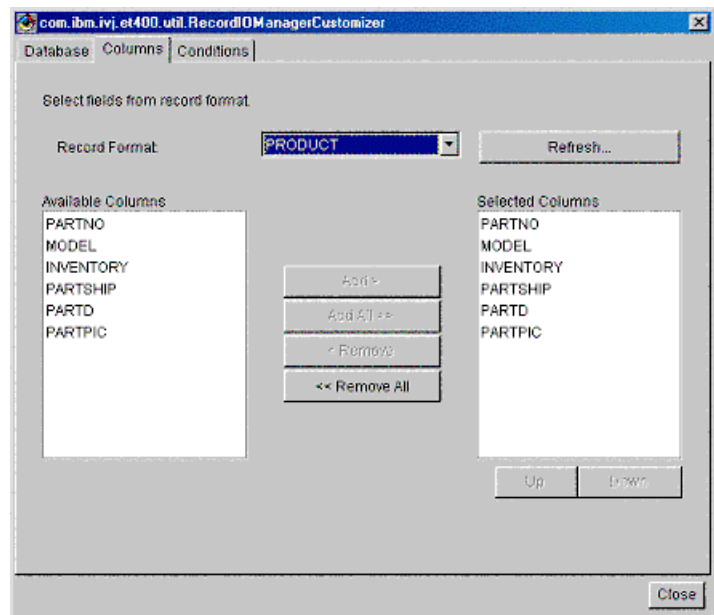
- Select the AS/400 assigned to you for the "Server" field
- Type in your **userid** and **password** respectively
- The name of the database is **PRODDTL** and is found in your user library.
- Type **PRODDTL** in the **File** field
- Type your library name in the **Library** field (the name should be the same as your userid).

see this image →



- Now select the "Columns" tab.
- The record format of the **PRODDTL** database and all of its available columns will be shown. By default, all columns are selected.
- Click the "Close" button
- Now close the property window.


see this image →

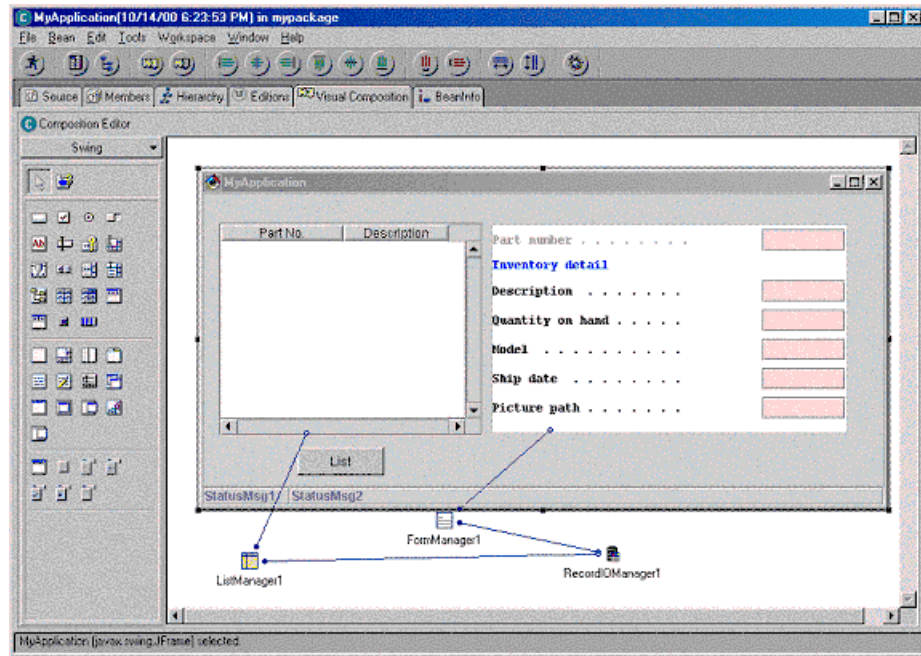


- **NOTE:** We just told the database controller,
- which database to manipulate, where it is
- Which record format to use
- And which fields to bring down from the AS/400.

Step 12: Lets add a button


- In order for a Java application to actually do something, it must be given some signal by a user.


- The signal mechanism most commonly used is a "button"
- Select the "Beans Palette" list, and click on "Swing", if it has not already been selected.
- Click on the JButton bean 
- Move your cursor below the table in our application, and drop the button. (see below)



- Double click on the button to open the property window.
- Click on the field next to **text** and enter "List" (don't need quotes)
- Close the property window.
- **NOTE:** We just gave the button a name.
- Now select the "List" bean and right-click. In the pop-up that appears, move to "Connect" and click on **actionPerformed**.
- With your "spider", left-click on the **ListManager1** bean. In the pop-up menu that appears, select "Connectable Features"
- The "End connection to" window should now be shown.
- Ensure that the "Method" radio button is selected and select the **readAllRecords()** method.
- Click "Ok"
- **NOTE:** We just told the subfile controller, to list all records in the database when the button is pressed. In the real world, you would most likely only get the first 10 (or so) records. You could then get the next set based on scrolling up or down. (this is more advanced)

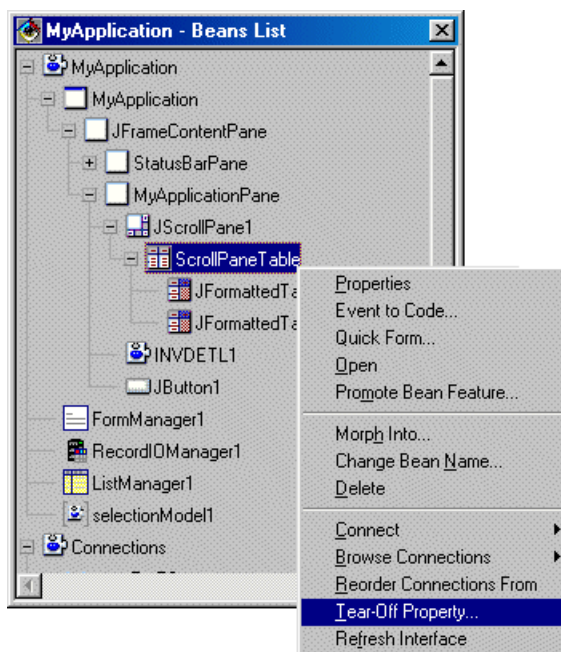
Step 13: Lets run our application

- On the VCE toolbar, click on the "Run" icon  .
- When the window opens press the "List" button and signon.

- Remember use the userid and password supplied with your terminal.
- You'll notice that all the records have been retrieved, and only the columns we added to the table are displayed.
- End the application by clicking on 

Step 14: Lets update the "Details View" (a little more difficult)

- Whenever you click on a row in the table (ie. subfile), we would like to see the details of our selection in the "Details View"
- We now need access to a part that we cannot click on because 2 columns are sitting on top of it
- In the VCE, select "Tools" from the menu bar, and click on "Beans List".



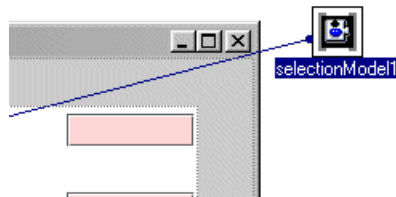
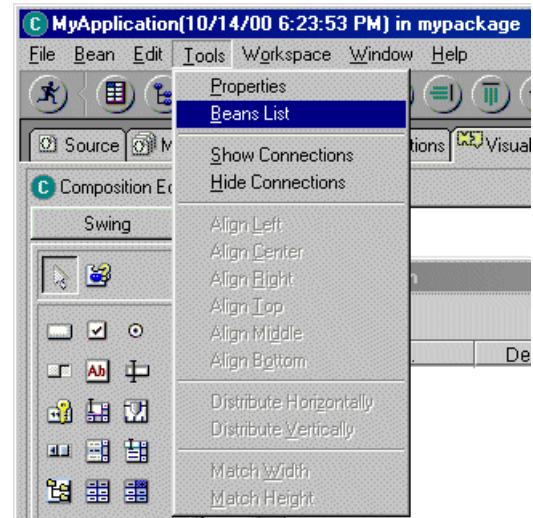
← see this image

- The "Beans List" is an easy way of selecting beans you have in the current VCE session.
- It also gives you an idea of the amount of Java code behind this application
- We now need access to a method (ie. Subroutine) inside this component. The method we are seeking is the one that will be invoked, each time a row is selected in the table
- In this case "selectionModel"

- Expand "MyApplication"
- Expand "JFrameContentPane"
- Expand "MyApplicationPane"
- Expand "JScrollPane1"
- Select "ScrollPaneTable" and right-click
- In the pop-up menu, click on "Tear-Off Property..."



- In the "Tear-Off Property Dialog", scroll down and click on "selectionModel(ListSelectionModel)"
- **BE CAREFUL**, there are 2 methods with the same name which take different parameters.
- Click on the "OK" button

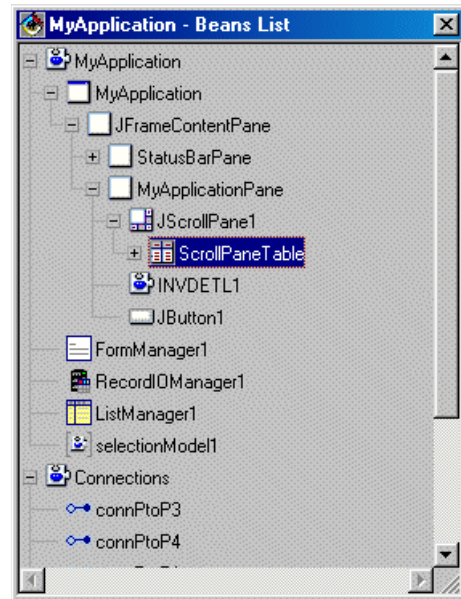


- You'll now notice that a new bean has been added to your VCE

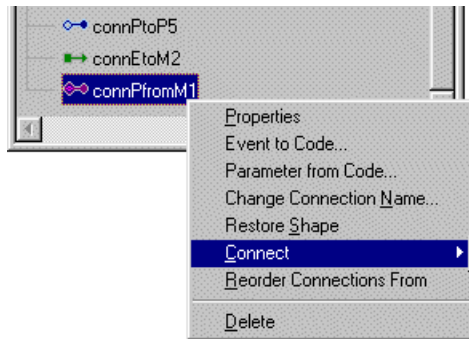
- Now that we have access to the correct method, lets wire things up.
- Select the "selectionModel1" method and right-click. In the popup that appears, move to "Connect" and click on **valueChanged**.
- With your "spider", left-click on the **FormManager1** bean. In the pop-up menu that appears, select "Connectable Features"
- The "End connection to" window should now be shown.
- Ensure that the "Method" radio button is selected and select the **readRecord(int)** method.
- Click "Ok"
- You'll notice that the connection you have made is not a solid line. This indicates that a parameter has not been set. (the "int" in **readRecord(int)**).
- Move your cursor over one of the ends of the connection you just made. You'll notice the cursor turns into a spider, now right-click. In the pop-up menu that appears, move to "Connect" and click on "recordNumber".
- With you "spider", left-click on the **ListManager1** bean. In the pop-up menu that appears, select "Connectable Features"
- The "End connection to" window should now be shown.
- Ensure that the "Method" radio button is selected and select the **getRecordNumber(int)** method.
- Click "Ok"
- You'll notice that this connection also is not a solid line. This indicates that a parameter has not been set. (the "int" in **getRecordNumber(int)**).

- For this next step, we need the "Beans List" window. If its closed, open it again by selecting "Tools" from the menu bar, and clicking on "Beans List".
- We now need access to the object containing the 2 columns
- Expand "MyApplication"
- Expand "JFrameContentPane"
- Expand "MyApplicationPane"
- Expand "JScrollPane1"
- "ScrollPaneTable" should now be visible

see this image →



- Now select the last connection you just made.

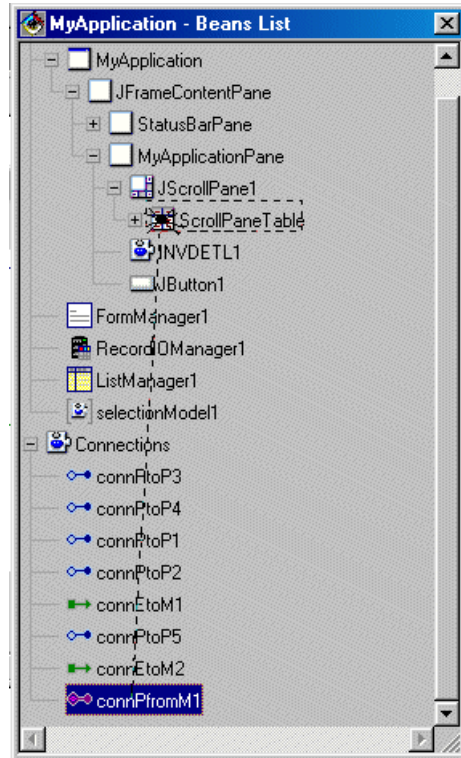


It should be the only non-solid line.

- You'll notice the connection is also selected in the "Beans List" window. Right-click on the connection name in the "Beans List" window. In the pop-up menu that appears, move to "Connect" and click on "row".



← see this image

- With your "spider" move up and left-click on the ScrollPaneTable in the "Beans List" window (see below)




- In the pop-up that appears, select "selectedRow"
- Now close the "Beans List" window
- **NOTE:** We just told the "Details View" which row to display based on the selected row in our table (ie subfile).

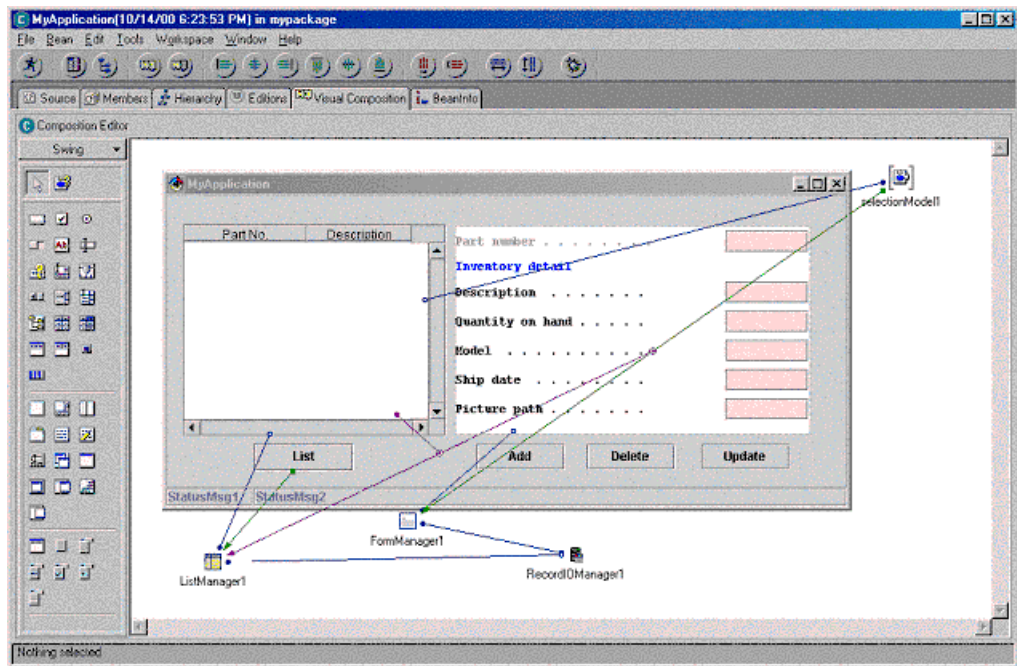
Step 15: Lets run our application

- On the VCE toolbar, click on the "Run" icon .
- When the window opens press the "List" button and signon.
- Remember use the userid and password supplied with your terminal.
- Once the records are displayed, click on a row.
- **CONGRATULATIONS:** You've completed the application
- **WAIT:** what about stuff like adding, deleting, and updating a record?
- Continue to the optional section if you have time, we'll add this support, and see it in action.
- End the application by clicking on .

Optional Step 16: Lets add the buttons

- Select the "Beans Palette" list, and click on "Swing", if it has not already been selected.
 - Click on the JButton bean 
 - Move your cursor below the "Details View" in our application, and drop the button.
 - Repeat this 2 more times, so that 3 buttons are below the "Details View"
 - Double click on one of the buttons. This will open the property window.
 - Click on the field next to text and enter "Add"

- Repeat this 2 more times, changing the remaining buttons to "Delete" and "Update"
- Close the property window
- You should now have a VCE that looks like this: (see below)



Optional Step 17: Lets hook up the "Add" button

- Select the "Add" button and right-click. In the popup that appears, move to "Connect" and click on **actionPerformed**.
- With your "spider", left-click on the **FormManager1** bean. In the pop-up menu that appears, select "Connectable Features"
- The "End connection to" window should now be shown.
- Ensure that the "Method" radio button is selected and select the **addRecord()** method.
- Click "Ok"
- **NOTE:** We just told the "Details View" controller (**FormManager1** bean), that each time the "Add" button is pressed, collect the data in the "Details View" and add it to the database. Since the **FormManager1** bean is connected to the **RecordIOManager1** bean, it will add the record to the database defined in that bean.
- When the application is running, it would be nice to see any changes you make to the database, as soon as you make them.
- Recall that, when the "List" button was pressed, we told the subfile controller to read all records. Since that point in time, the subfile has not been refreshed.
- Lets add some more connections
- Select the connection between the "Add" button and the **FormManager1** bean.
- You'll notice that little blocks appear at both ends of the connection.
- When the "Add" button is pressed, the database is updated. At this point in time, we would also like to refresh the table (ie. Subfile).

- Move your cursor over one of the ends of the connection you just selected. You'll notice the cursor turns into a spider, now right-click. In the pop-up menu that appears, move to **"Connect"** and click on **"normalResult"**.
- With your "spider", left-click on the **ListManager1** bean. In the pop-up menu that appears, select **"Connectable Features"**
- The **"End connection to"** window should now be shown.
- Ensure that the **"Method"** radio button is selected and select the **readAllRecords()** method.
- Click **"Ok"**
- **NOTE:** This method will clear the table (ie. Subfile) and loads it with a fresh set of data.
- But now we would also like to clear the **"Details View"** once the data is saved
- Select the original connection between the **"Add"** button and the **FormManager1** bean which called the **addRecord()** method.
- Move your cursor over one of the ends of the connection you just selected. You'll notice the cursor turns into a spider, now right-click. In the pop-up menu that appears, move to **"Connect"** and click on **"normalResult"**.
- With your "spider", left-click on the **FormManager1** bean. In the pop-up menu that appears, select **"Connectable Features"**
- The **"End connection to"** window should now be shown.
- Ensure that the **"Method"** radio button is selected and select the **clearAllData()** method.
- Click **"Ok"**
- **NOTE:** After the data is saved, the **"Details View"** will now be cleared.

Optional Step 18: Lets hook up the **"Delete"** button

- Select the **"Delete"** button and right-click. In the popup that appears, move to **"Connect"** and click on **actionPerformed**.
- With your "spider", left-click on the **FormManager1** bean. In the pop-up menu that appears, select **"Connectable Features"**
- The **"End connection to"** window should now be shown.
- Ensure that the **"Method"** radio button is selected and select the **deleteRecord()** method.
- Click **"Ok"**
- **NOTE:** We just told the **"Details View"** controller (**FormManager1** bean), that each time the **"Delete"** button is pressed, delete the record displayed in the **"Details View"**.
- Similar to the **"Add"** functionality, we would like to refresh the table (ie. subfile)
- Select the connection between the **"Delete"** button and the **FormManager1** bean.
- You'll notice that little blocks appear at both ends of the connection.
- When the **"Delete"** button is pressed, the database is updated. At this point in time, we would also like to refresh the table (ie. Subfile).
- Move your cursor over one of the ends of the connection you just selected. You'll notice the cursor turns into a spider, now right-click. In the pop-up menu that appears, move to **"Connect"** and click on **"normalResult"**.
- With your "spider", left-click on the **ListManager1** bean. In the pop-up menu that appears, select **"Connectable Features"**

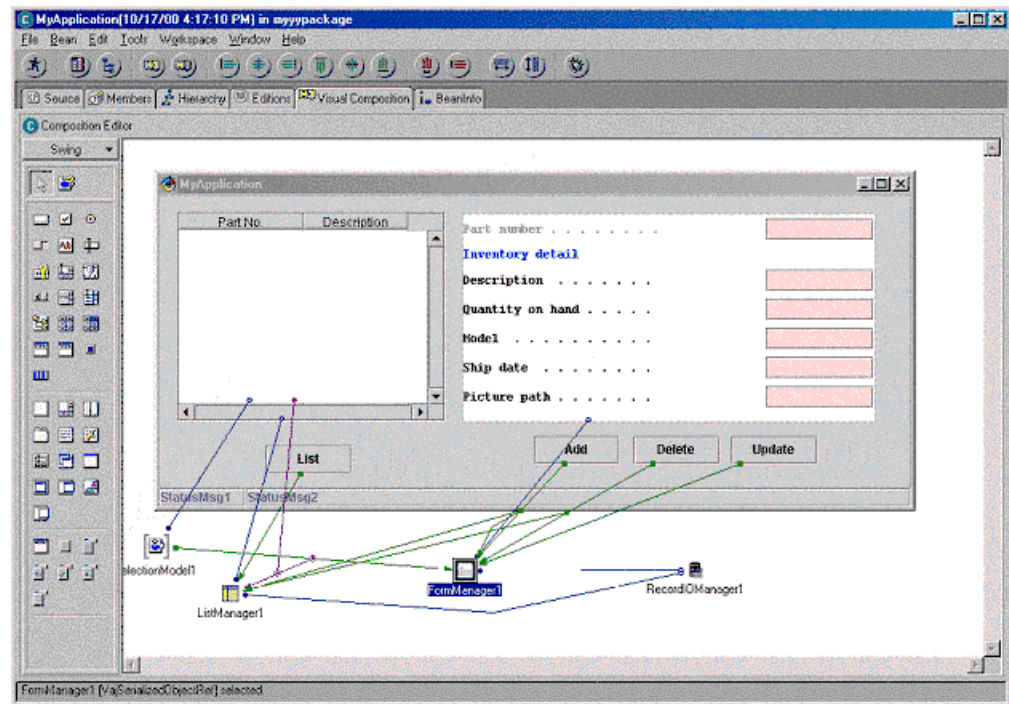
- The "End connection to" window should now be shown.
- Ensure that the "Method" radio button is selected and select the **readAllRecords()** method.
- Click "Ok"
- **NOTE:** This method will clear the table (ie. Subfile) and loads it with a fresh set of data.

Optional Step 19: Lets hook up the "Update" button


- Now select the "Update" button and right-click. In the popup that appears, move to "Connect" and click on **actionPerformed**.
- With your "spider", left-click on the **FormManager1** bean. In the pop-up menu that appears, select "Connectable Features"
- The "End connection to" window should now be shown.
- Ensure that the "Method" radio button is selected and select the **updateRecord()** method.
- Click "Ok"
- **NOTE:** We just told the "Details View" controller (**FormManager1** bean), that each time the "Update" button is pressed, update the record displayed in the "Details View" with the new values.
- The updateRecord method is special. You don't need to clear the "Details View", and the table (ie. Subfile) will be updated automatically.

Optional Step 20: Lets run it

- Your VCE should now look something like this:



- On the VCE toolbar, click on the "Run" icon  .

- When the window opens press the "**List**" button and signon.
 - Remember use the userid and password supplied with your terminal.
 - Once the records are displayed, click on a row.
 - Try adding a new item
 - Notice how the number field is limited to 5 (Step 6)
 - Try deleting
 - Try update
-
- End the application by clicking on 
-
- Close the VCE and take a look at the amount of code generated for you inside your package "**mypackage**".
 - Not too bad...
CONGRATULATIONS: You've completed the application