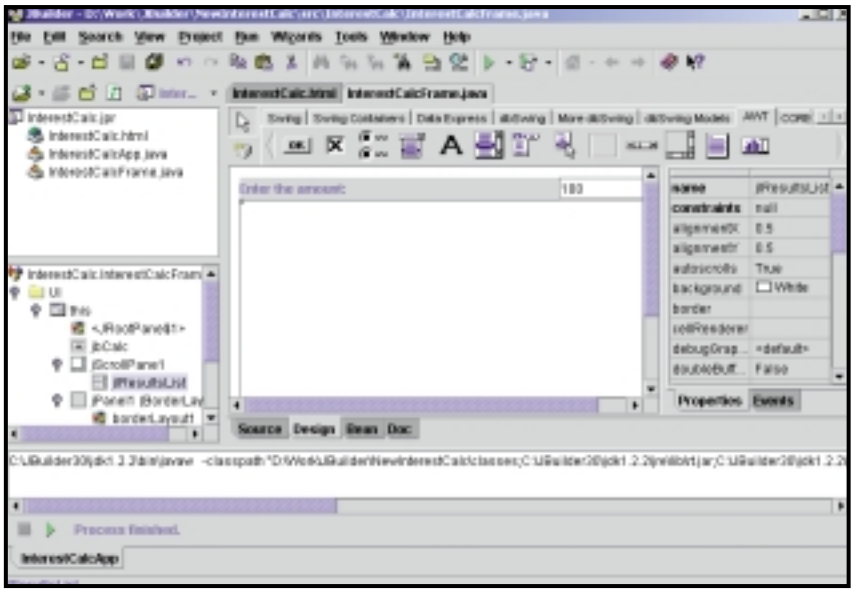
Getting the JBuilder in

Tim Anderson presents an example application to help JBuilder virgins achieve results

You lucky people have recently received two free paths to Java development, one in the form of JBuilder 2.0 distributed on the *PCW* March 2000 cover CD, and the other as JBuilder Foundation, available as a free download on the Borland website (www.borland.com). There are some subtle and not-so-subtle differences. The free JBuilder 2.0 is for personal use only, whereas you can use JBuilder Foundation however you wish. Furthermore, JBuilder Foundation is itself a pure Java application, supported on Windows, Solaris and Linux, whereas JBuilder 2.0 is Windows-only at development time. Unless you are trying out the database features, which are exclusive to full JBuilder versions, Foundation is a better choice because its Swing components are fully up-to-date.

JBuilder comes with a tutorial that creates a pretty but empty application, and unfortunately it does little to demystify either Java or JBuilder. What follows is a more nuts-and-bolts explanation of how to build an example application, using JBuilder Foundation. The application calculates the annual returns on a sum of money invested. For simplicity it is on the basis of a single year-end credit rather than daily interest. Note that everything in Java is case-sensitive, so VB hackers beware!



The finished application in the JBuilder Foundation designer. Note the structure pane at the bottom left, a vital part of the IDE

The starting point is to run JBuilder Foundation, close the Welcome project if it has auto-opened, and choose File, New Project. Choose a sensible working directory and call the project InterestCalc.jpr. Click Next and enter whatever you like into the detail fields, then click Finish.

Empty JBuilder projects do not do much, consisting of just an HTML file. Choose File, New and select Application. Enter InterestCalc for the package and InterestCalcApp for the class. Click Next and call the frame class InterestCalcFrame, and the Title whatever you want, noting that what you put in Title ends up as the default title bar caption for the main window. Select Center Frame on Screen and leave the other options unchecked.

Foundation is a better choice because its Swing components are fully up-to-date

JBuilder generates two Java source files, one for the application and one for the frame. The first task is to design the user interface, so click the Design tab to open the visual design surface. This will be a Swing application, the idea being that the user enters an amount into a text field, and clicks a button to show the returns at different rates of interest. You never know, it might actually be useful.

JBuilder's designer is another culture shock for VB developers. All the components are positioned by a layout manager, so they do not necessarily end up where you place them. This application uses the BorderLayout, which places components either in the centre or along one of the four borders. Select the Swing tab in the component palette, then select a JButton and click on the empty frame to place it. Now edit its properties, setting the name to jbCalc, the constraints property to South, and the text to "Calculate Returns".

In Visual Basic you might now place a list box on the frame. This will not do in JBuilder, because by default a JList does not scroll. To get around this, find jScrollPane on the component palette and place it, with its Constraints set to Center, followed by a JList on top. Call the JList: jlResultsList.

When you work with JBuilder it is sometimes tricky to select the right component on the frame, particularly since the position changes as you add more components. A handy tip is that you can add components to the Structure pane at the bottom left of the IDE, as well as to the design surface itself.

At the top of the frame we need to see a label and a text field. Because these need to appear side by side on the top border, they need their own separate

container. Choose the Swing Containers tab in the component palette and place a jPanel, clicking on the entry for this in the Structure pane (to avoid placing it on the jScrollPane). Set its constraints property to North and its layout property to BorderLayout. Place a jLabel and a jTextField on the jPanel, with the jLabel West and the jTextField East. Call the jTextField jtAmount, set its text property to 100 and its preferredSize to an x co-ordinate of 100, leaving the co-ordinate unchanged. Give the label a text property of, say, 'Enter the amount'. Click the Run icon to make sure you are in good shape so far.

Here is another trap for the VB-unwary. On their own, JList components cannot show any items. To add this vital functionality, you need to add a ListModel, a non-visual class that handles the contents of the list. Click the Source tab of InterestCalcFrame and find the section that begins:

```
public class ↙
InterestCalcFrame extends ↙
JFrame {...
```

(*Key: ↙ code string continues*)

Now you can go back to the design tab, select jlResultsList in the structure pane, drop down its model property and choose ResultsListModel. Alternatively, you could do this in code. If you look at the source, you can see in the jbInit() method the new line:

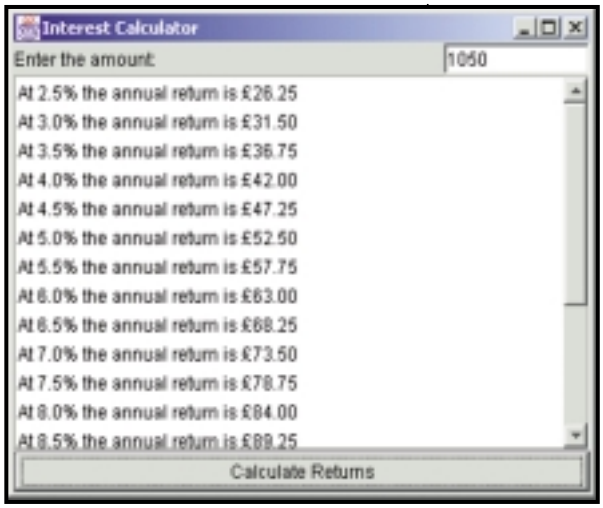
```
jlResultsList.setModel↙
(ResultsListModel);
```

For even greater efficiency you could amend the line that constructs the jlResultsList, since the constructor accepts a ListModel parameter, but you might as well do things the JBuilder way.

The rest of the code for this application is in the actionPerformed event handler for the button. Double-click the button to generate its skeleton. Then add the code as shown in Figure 1 overleaf. You also need to add two import statements to InterestCalcFrame.java:

```
import java.util.*;
import java.text.*;
```

At this point you can run the code and test out the application, noting the nicely formatted currency values achieved with a little extra effort.



At run time the application displays a list of returns

Although this routine is fairly simple, it still has perplexing features if you come from a VB background. Here are a few notable points.

First, although Java supports several basic types like int and float, there is also a set of classes that wrap those types. These wrapper classes have utility methods that are invaluable. In the code, notice that the sumInvested variable is an object of type Float, and the annualReturn is a basic float type. The initial capital makes the difference. You can use the utility methods without actually declaring a new object. For example, the snippet:

```
Float.toString(interestRate)
```

converts a float to a string, even though Float is a class and not an object.

Although this routine is simple, it still has perplexing features if you're from a VB background

The routine begins by initialising a couple of variables and then clearing the list box (or the ListModel) with its removeAllElements method. Next, the value of sumInvested is retrieved using the snippet of code:

```
sumInvested = ↙
Float.valueOf(jtAmount↙
.getText());
```

Unpacked, what this does is retrieve the text of the input field and pass it as a parameter to the utility method Float.valueOf, which converts it to a Float object. For the actual calculation a few lines later, we need to retrieve the value wrapped by the sumInvested Float object, using the floatValue() method.

Currency formatting the Java way is done with another object, this time of type NumberFormat. This has a method, getCurrencyInstance, that takes a Locale parameter. The nice feature here is that by using different Locale objects you get built-in international currency formatting. In this case, the Locale object is constructed by using:

```
thisLocale = new ↙
Locale("en","GB")
```

which gets UK currency. For some reason if you use "UK" you get a hash symbol (the US pound character) instead of a proper pound sign.

Code changes

If you want to create this application as an applet, to display in a browser, you'll need to change the code significantly, since the old awtList component is quite different from the Swing JList. The only other option is to use Sun's plug-in or ActiveX control that lets Navigator and Internet Explorer use JDK 1.2 – but this won't be popular if it's a public website.

It is also worth noting that much of the excitement currently around Java is with non-visual components running on web servers or embedded into databases, such as those from IBM and Oracle. Swing is a superb visual interface class library in some ways, but it does tend to be slower and it is harder to exploit native Windows features, not least COM. On the other hand, if you want to write a utility that runs on Linux as well as Windows, this is a great place to start.

For help with JBuilder, two excellent sources are the Borland newsgroups, at forums.inprise.com, and the Java Tutorial at http://java.sun.com.

Data corruption on NT

Craig Harrison picked on a comment in this column to the effect that NT networks are prone to corrupt shared data files. The problem is that some heavy users of networked applications

such as dBase, Paradox, and more occasionally Access, find that data gets corrupted and has to be repaired. It seems to be a problem mainly caused by opportunistic locking. This is where the server grants exclusive access to a user even though shared access was requested, to speed performance. When another user requests access to the same file, the exclusive access is revised to shared access. In association with write caching, the result from time to time is scrambled data. There is another specific problem with a driver on original versions of Windows 95, so the fix is to update the driver if appropriate, and disable opportunistic locking, which is a matter of changing a registry entry: HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/lanmanserver/parameters/EnableOpLocks and the adjacent entry EnableOpLockForceClose. To disable opportunistic locking, set the first to zero and the second to one. If the entries do not exist, create them. For further details, search the Microsoft Knowledge Base for 'Opportunistic locking'.

Now for an immediate disclaimer. First, I have no idea whether this works on Windows 2000. Second, you should think long and hard before deliberately slowing down the performance of your server. The real solution is to upgrade to a client-server database.

Also, there is some mystery swirling around this issue and there could be many other reasons for corrupt data on a network, like a faulty network card or a badly written client. The problem is real, though, and a contact at DataEase, another elderly file-sharing database, told me this was its single biggest problem with NT. Proceed with care.

Running apps from VB

Robert Neilson asks: 'Is it possible to run another .exe program by clicking a command button?' Visual Basic has a built-in Shell function that does exactly this. The syntax is:

```
Shell(pathname,windowstyle)
```

where windowstyle is an optional argument specifying how the application should initially appear (eg vbHide for a hidden window or vbNormalFocus which is a sensible default).

For richer functionality, it is better to use an API function, ShellExecute. Use the VB API viewer to find this function and paste in the declaration. The advantage of ShellExecute is that it takes a file name, which need not be an executable. That means you can take advantage of the current defaults for things like email, web browsing, or multimedia. For example, if a user has Outlook Express as the default mail client, they will not thank you for opening Outlook to send an email, and in any case Outlook might not be installed.

FIG 1

Code to calculate annual returns

```
void JButton1_actionPerformed(ActionEvent e) {
int countVar;
Float sumInvested;
float annualReturn;
float interestRate;
String stringResult;
String formattedReturn;
NumberFormat formatter;
Locale thisLocale;

interestRate = 2;
thisLocale = new Locale("en","GB");
ResultsListModel.removeAllElements();
sumInvested =  Float.valueOf(jtAmount.getText());
for (countVar = 0; countVar < 20; countVar++) {
interestRate = interestRate + 0.5f;
annualReturn = sumInvested.floatValue() * ↙
(interestRate/100);

formatter = NumberFormat.getCurrencyInstance(thisLocale);
formattedReturn = new ↙
String(formatter.format(annualReturn));

stringResult = new String("At " + ↙
Float.toString(interestRate) +
"% the annual return is " + formattedReturn);
ResultsListModel.addElement(stringResult);
}
```

(*Key:* ↙ *code string continues*)

Permission not granted

If you install Windows 2000, you will notice strong warnings against logging on routinely with administrator permissions. This has always been true of Windows NT, but Microsoft is more up-front about it now, probably because of Internet security risks. The significance for developers is the number of applications which either fail to install or fail to run if you log on as an ordinary domain user. The issue is usually not especially complex. Common examples are an application trying to write to a configuration file in the Windows directory, or an attempt to update a registry key in HKEY_LOCAL_MACHINE, which is read-only by default. Paint Shop Pro 4, for instance, pops up the message 'failed to update the system registry' whenever it is run, although in fairness it continues without visible issue. While it is sometimes acceptable to require administrator permissions for installation, making this a runtime requirement is now exceedingly poor practice.

This could be the death-knell for those old 16bit applications where the source code is not available. Well-behaved apps and setup routines should offer sensible alerts when administrator permissions are needed, and minimise the need by writing to HKEY_CURRENT_USER and installing their files into the application directory. You can detect the problem easily by attempting to write a key or a file, and trapping any exception raised. The permissions problem is by far the most likely cause.

CONTACTS

Tim Anderson welcomes your comments on the Visual Programming column. Contact him via the *PCW* editorial office or email: **visual@pcw.co.uk**