Building a Java Desktop Database Application

This tutorial shows how to create a desktop Java application through which you can access and update a database. The tutorial takes advantage of support in NetBeans IDE for the following technologies:

- The Java Persistence API (JPA), which helps you use Java code to interact with databases.
- Beans Binding (JSR-295), which provides a way for different JavaBeans components to have property values that are synchronized with each other. For example, you can use beans binding to keep the values of cells in a JTable visual component in synch with the values of fields in an entity class. (In turn, the entity class represents the database table.)
- The Swing Application Framework (JSR-296), which provides some useful building blocks for quickly creating desktop applications.

We will create a database CRUD (create, read, update, delete) application with a custom component used for visualizing the data (car design preview).

This tutorial is largely based on a screencast that was based on a development build of a previous version of the IDE. Some of the user interface has changed since that demo was made, so you might notice some differences between this tutorial and the demo. You can view the demo (about 9 minutes) now or download a zip of the demo.

Expected duration: 45 minutes

Contents

To complete this tutorial, you need the software and resources listed in the following table.

Software or Resource	Version Required
NetBeans IDE	version 6.5
Java Development Kit (JDK)	version 6 or version 5
Java DB database server (Java DB is included with JDK 6)	
Car database SQL script	
CarPreview project	

Setting Up the Database

Before you begin creating a desktop CRUD (create, read, update, delete) application in the IDE, you should already have the IDE connected to the database that your application will use. Having this connection set up in advance will allow you to take advantage of IDE features that automate the binding of the database to your application.

In this tutorial, we provide instructions for using a Java DB database, since there is a convenient interface for starting and stop the Java DB database server from the IDE. However, you can use a different database server without too much difficulty.

First verify that you have Java DB registered in the IDE. Java DB is automatically registered in the IDE in a number of cases, such as when you have the GlassFish application server registered in the IDE or when you are running on JDK 6. If Java DB is not registered in the IDE, register it manually.

To verify whether Java DB is registered in the IDE:

1. Open the Services window.
2. Right-click the Databases > Java DB node and choose Properties.
   If Java DB is registered, the Java DB Installation and Database Location fields will be filled in.
   If Java DB is not registered, fill in the following two fields:
   - **Java DB Installation**. Enter the path to the database server.

- **Database Location**. Enter the folder where you want the databases to be stored.

3. Click OK.

Starting the Server and Creating a Database

Once Java DB is registered with the IDE, you can easily start and stop the database server, as well as create a new database.

To start the database server in the IDE:

- In the Services window, right-click Databases > Java DB and choose Start Server.
  If you do not already have a location set for the database, the Set Database Location dialog box appears. Enter a location for the database server to store the databases. You can create a new folder there if you wish.

Once the server is started, Java DB Database Process tab opens in the Output window and displays a message similar the following:

```
Apache Derby Network Server - 10.2.2.0 - (485682) started and ready
        to accept connections on port 1527 at 2007-09-05 10:26:25.424 GMT
```

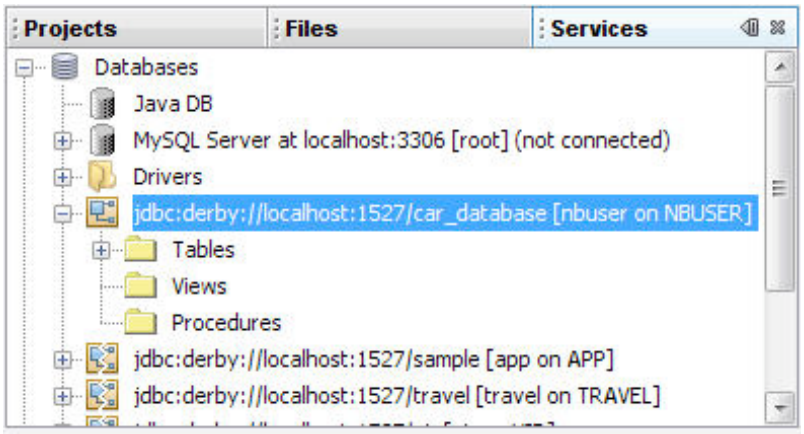
To create the database:

1. In the Services window, right-click Databases > Java DB and choose Create Database.
2. For the Database Name text field, type `car_database`. Also set the User Name and Password to nbuser.
3. Click OK.

Connecting to the Database

So far, we have successfully started the the database server and created a database. However, we still need to connect to the new database before we can start working with it in the IDE. To connect to the `car_database` database:

1. Switch to the Services window (Ctrl+5) and expand the Databases node to see your new database.
2. Right-click the database connection node (`jdbc:derby://localhost:1527/car_database[nbuser on NBUSER]`) and choose Connect.
   The connection node icon should now appear whole (), which signifies that the connection was successful.



3. Expand the connection node, right-click its Tables subnode, and choose Execute Command.
4. Copy the contents of the car.sql file and paste them into the SQL Command 1 tab of the Source Editor.
   This is the SQL script which will populate the database with data about cars.
5. Click the Run SQL button ()in the toolbar of the Source Editor to run the script.

Creating the Application

1. Choose File > New Project.
2. In the first panel of the wizard, expand the Java category and select the Java Desktop Application template. Click Next.
   The Java Desktop Application template provides many basics of a visual application, including basic menu items and commands.
3. In the Name and Location page of the wizard, do the following things:

1. In the Project Name field, type `CarsApp`. The value of this field sets the display name for the project in the Projects window.
2. Select the Set As Main Project checkbox.
3. (Optional) Edit the Project Location field to change the location of your project metadata.
4. (Optional) Select the Use Dedicated Folder for Storing Libraries checkbox and specify the location for the libraries folder. See Sharing Project Libraries for more information on this option.
5. In the Choose Application Shell field, select Database Application.
6. Click Next.

4. In the Master Table page of the wizard, select the database connection for the `CAR` database. The listing for the database should look something like the following: `jdbc:derby://localhost:1527/car_database[nbuser on NBUSER]`
5. Fill in the password (`nbuser`) for the database and select the Remember Password During This Session checkbox. After the connection to the database is established, the Database Table field should display `CAR` and the Columns to Include list should include the names of 10 columns for the CAR database. For now, we will use only five of them in the application.
6. Select the bottom five column names (beginning with `SUN_ROOF` and ending with `MODERNNESS`) and click the < button to move them to the left column. Click Next.
7. In the Detail Options panel, click Finish.
   The wizard then generates the a basic user interface with a table and a database connection. This might take a few seconds as the IDE generates the project and the code.

Running the Generated Application

At this point, you have a basic running application with a graphical user interface (GUI) that has the following features:

- Ability to view and modify values in five columns of the CAR database.
- Basic menu items.
- Persistence of its window state between sessions. When you close the application, the window position and size are remembered. So when you reopen the application, the window opens in the same position as it was when you closed it.
- An About dialog box, which you can easily customize.
- `.properties` files containing the labels in the user interface. Using `.properties` files is a good way to keep the logic of your code separate from the text that appears in the user interface of your application. Such separation is useful for making it easier to localize your program, among other reasons.

To see some of the features that are already built into the application, follow these steps:

1. Right-click the project's node and choose Run.
   After a few seconds, the application stars and a window called Database Application Example appears. This window contains a table and several controls that enable you to edit the CARS database.
2. Select the first record in the table (for the Acura).
3. Select the Price text field and replace the existing value with `46999`. Then press Enter.
   The value should appear updated in the table. (However, that value will not be reflected in the database until you click Save.)
   Similarly, you can update any other values in the table.
4. Click New to create a new record. Then fill in values for each of the fields (Make, Model, Price, Body Style, Color). For example, you can fill in `Trabant`, `Classic`, `1000`, `wagon`, and `blue`. Click Save to save the entry in the database.
5. Click the Database Application Example title bar and drag the application to a different place on your screen.
6. Click the left border of the Database Application Example window and drag to the left to increase the size of the window.
7. In the Database Application Example menu bar, choose File > Exit.
8. In the IDE, right-click the project's node and choose Run Project.
   The Database Application Example window will open in the same size and position it was in when you closed the application.

Reviewing the Generated Parts of the Application

The connection between the master table (a JTable component) and the database is handled with a combination of the

following mechanisms, all of which have been generated by the IDE:

- The `Car.java` entity class, which is used to read and write data to the CAR database table. Entity classes are a special type of class that enable you to interact with databases through Java code. Entity classes use Java annotations to map class fields to database columns.
- The `META-INF/persistence.xml` file, which defines a connection between the database and the entity class. This file is also known as the persistence unit.
- Using *beans binding* to connect the properties of the entity class with the properties of the JTable component. Beans binding is a new technology based on JSR 295 and which will probably be included in a future Java SE release.
- The `entityManager`, `query`, and `list` objects, which are defined in the `CarsView` class and which are listed in the Inspector window.
    - The entity manager object is used to retrieve and commit data within the defined persistence unit scope.
    - The query object defines how the particular data collection is retrieved from the entity manager. (You can change the way that the query object works by selecting the query object in the Inspectory window and changing the `query` property in the property sheet. The `query` property uses JPA query language.
    - The list object is an observable collection that holds the data from the query. An observable collection is a special kind of collection on which you can place a listener to find out when changes to the collection have been made.

Using the Inspector window and the property sheet, you can follow these steps to see how the JTable is bound to data:

1. In the Inspector window, select the mainPanel[JPanel] > masterScrollPane [ScrollPane] > masterTable [JTable] node. Then click the Binding tab in the Properties window.
2. Look at the elements property to confirm that it is bound to a list.
3. Click the ellipsis [...] button to open the Bind masterTable.elements customizer, where you can further customize the binding between the table and the database. For example, you can see that the customizer enables you to specify which table columns are bound.

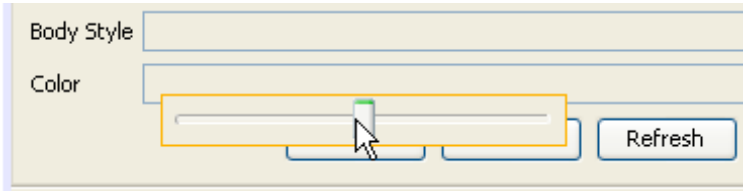
Besides the Binding category in property sheet you can also use the Bind menu in context menu.

Adding Additional Controls

We will now add controls to the form for some of the attributes we initially excluded. Instead of using a table, we will add JSlider components (for the tire size and modernness attributes) and JCheckbox components (for the sun roof and the spoiler).

Follow these steps to add the additional components:

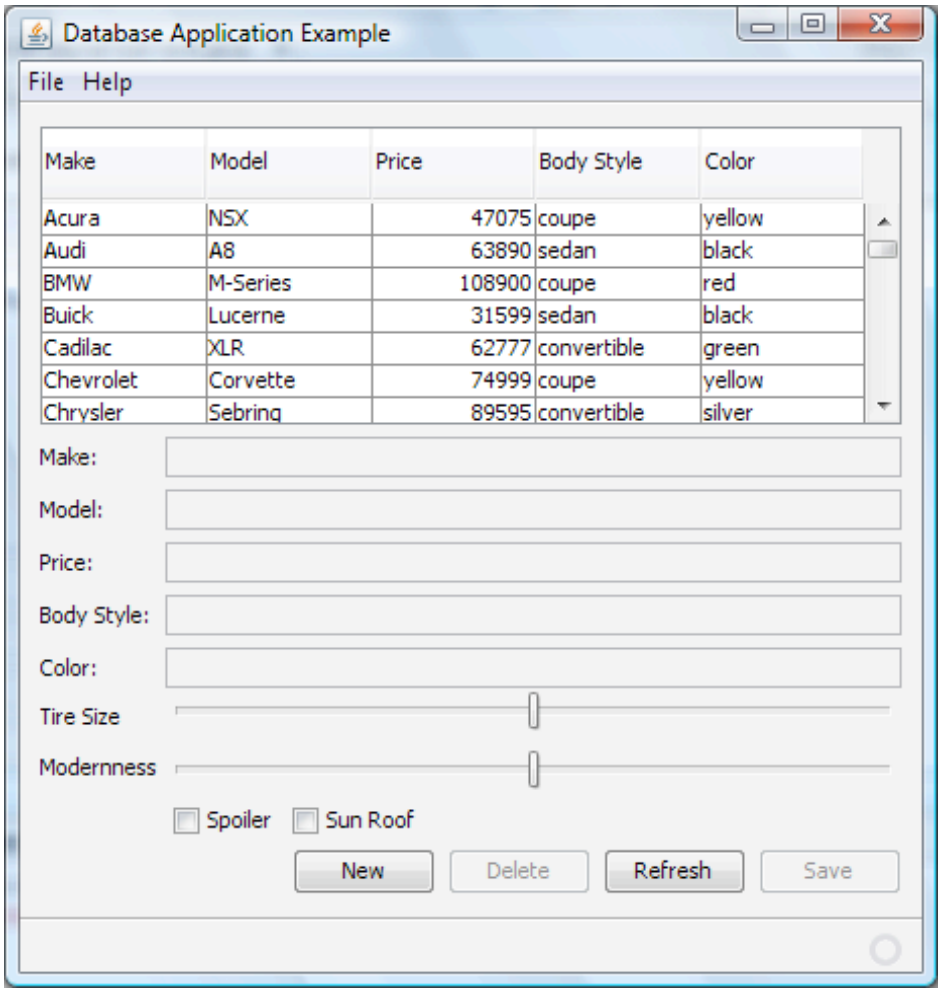
1. Add the first slider by clicking the Slider button in the Palette window and then clicking in the form just above the New button. Before clicking in the form to insert the slider, make sure that no horizontal slotted guiding lines are shown. These lines indicate that the slider will be inserted in the same line as the fields or the buttons. See the figure below to see where you should drop the slider into the form.



   Note: If you drop the component in a place you do not want and thus cause several undesired layout changes, you can use the Undo command to reverse the changes. Choose Edit > Undo or press Ctrl-Z.

2. If necessary, stretch the slider to the left to align it with the left side of the text field components.
3. Stretch the slider to the right to span the whole form width.
4. Add a label to the left of the slider and set its text to `Tire Size`. (Click the label to make it editable.)
5. Add another slider below the first slider, and adjust its width and alignment where necessary.
6. Add another label below the Tire Size label and set its text to `Modernness`.
7. Add two checkboxes below the sliders. Set their text to `Spoiler` and `Sun Roof`. (Make the display text editable by clicking the checkbox once, pausing, and then clicking the checkbox again. You can also right-click the checkbox and choose Edit Text.)

The form should look like the screenshot shown below.

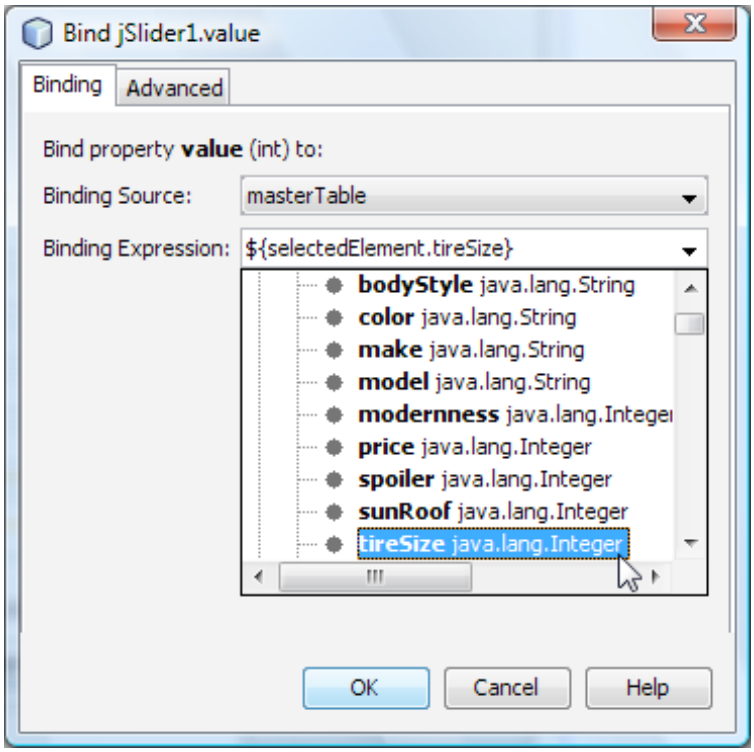


Binding Controls to Values in the Table

We will now use beans binding features to tie the values shown in table cells to the state of the controls we have added. This will allow you to change the values of cells in the table by using the sliders and checkboxes.

To bind the sliders to their corresponding table elements:

1. In the form, right-click the Tire Size slider and choose Bind > value.
2. In the Binding Source drop-down list of the Binding dialog box, select masterTable.
3. In the Binding Expression drop-down list, select the selectedElement > tiresize node.



4. In the form, right-click the Modernness slider and choose Bind > value.
5. In the Binding Source drop-down list of the Binding dialog box, select masterTable.
6. In the Binding Expression drop-down list, select selectedElement > modernness.

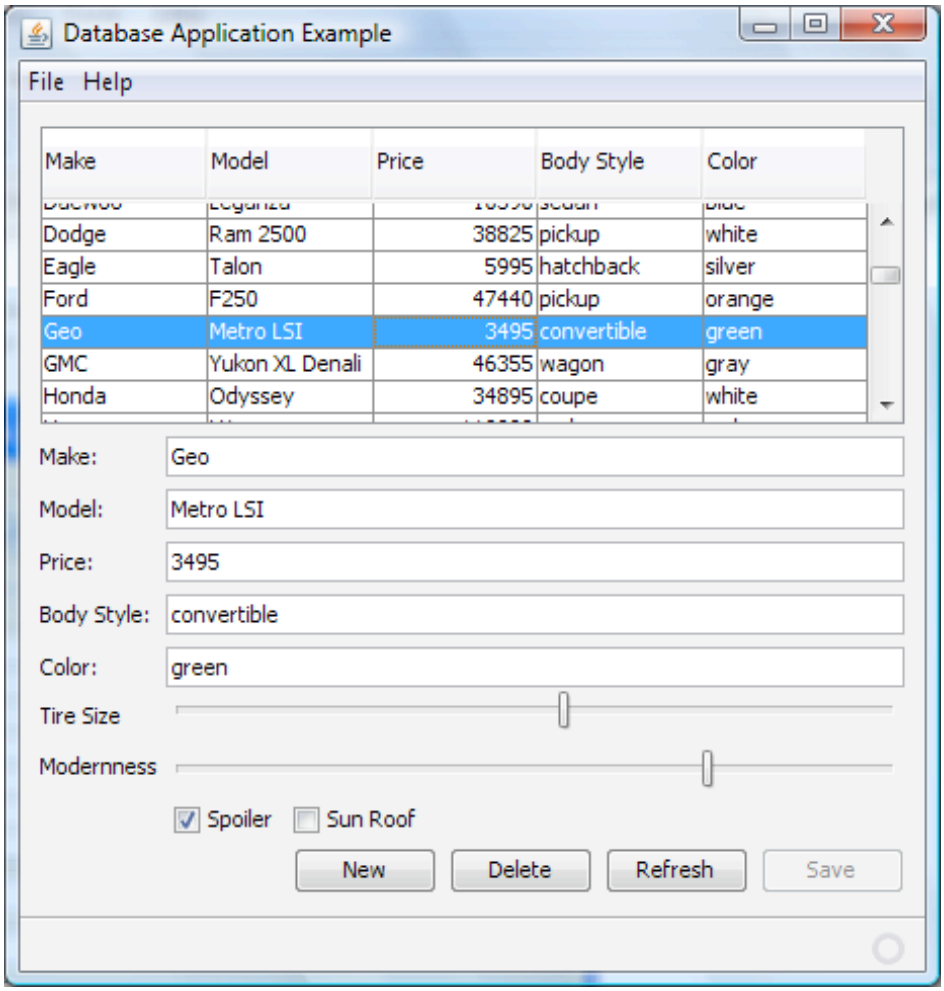
To bind the checkboxes to their corresponding table elements:

1. In the form, right-click the Spoiler checkbox and choose Bind > selected.
2. In the Binding Source drop-down list of the Binding dialog box, select masterTable.
3. In the Binding Expression drop-down list, select selectedElement > spoiler.
4. Click OK to exit the Bind dialog box.
5. In the form, right-click the Sun Roof checkbox and choose Bind > selected.
6. In the Binding Source drop-down list of the Binding dialog box, select masterTable.
7. In the Binding Expression drop-down list, select selectedRow > sunRoof.
8. Click OK.

You should now be able to change database entries using the slider and checkboxes.

To verify that the sliders and checkboxes work:

1. Open the Services window.
2. Make sure the IDE has a connection to the database by right-clicking Databases > jdbc:derby;//localhost:1527/car_database and choosing Connect.
3. Right-click the Databases > jdbc:derby;//localhost:1527/car_database > Tables > node and choose View Data.
4. Look at the SUN_ROOF, SPOILER, TIRE_SIZE, and MODERNNESS values for the first record.
5. Choose Run > Run Main Project to execute the application.
   The running application should look similar to the screenshot shown below.



6. In the running application, select the first record.
7. Move the sliders and change the checkbox selections.
8. Click Save to save the changes into the database.
9. In the Services window, use the View Data command again.
   The values in the database should reflect changes that you have made.

Setting up a Custom Component

For an even more graphic demonstration of beans binding in action, let's add a custom component that will paint a preview of the selected car. We will bind properties of this component to corresponding table elements. Then when you run the application again, the car preview will be modified as you change the selected row and change the values of the various fields.

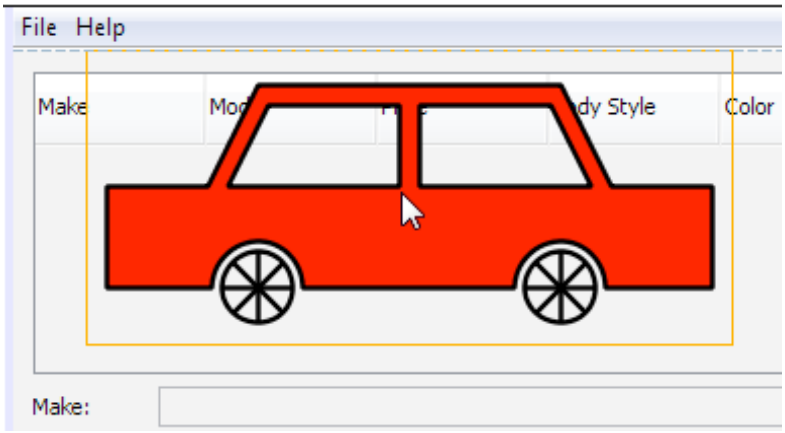
To make the CarPreview component available for the CarsApp project:

1. If you have not already done so, download the CarPreview.zip file.
2. Using a standard zip tool, extract the archives of the zip file.
3. Choose File > Open Project and navigate into the extracted contents of the zip file and select the CarPreview project.
4. Click Open Project.
   The project opens in the IDE.
5. Right-click the CarPreview node and choose Clean and Build.
   This compiles the files in the project so that you can use the CarPreview class as a bean that can be dragged and dropped directly on to the form.

This component was written as a JavaBeans component, so you could add it to the Palette, which would be convenient for adding the component to multiple applications. But for now we will simply drag the component directly into your application directly from the Projects window.

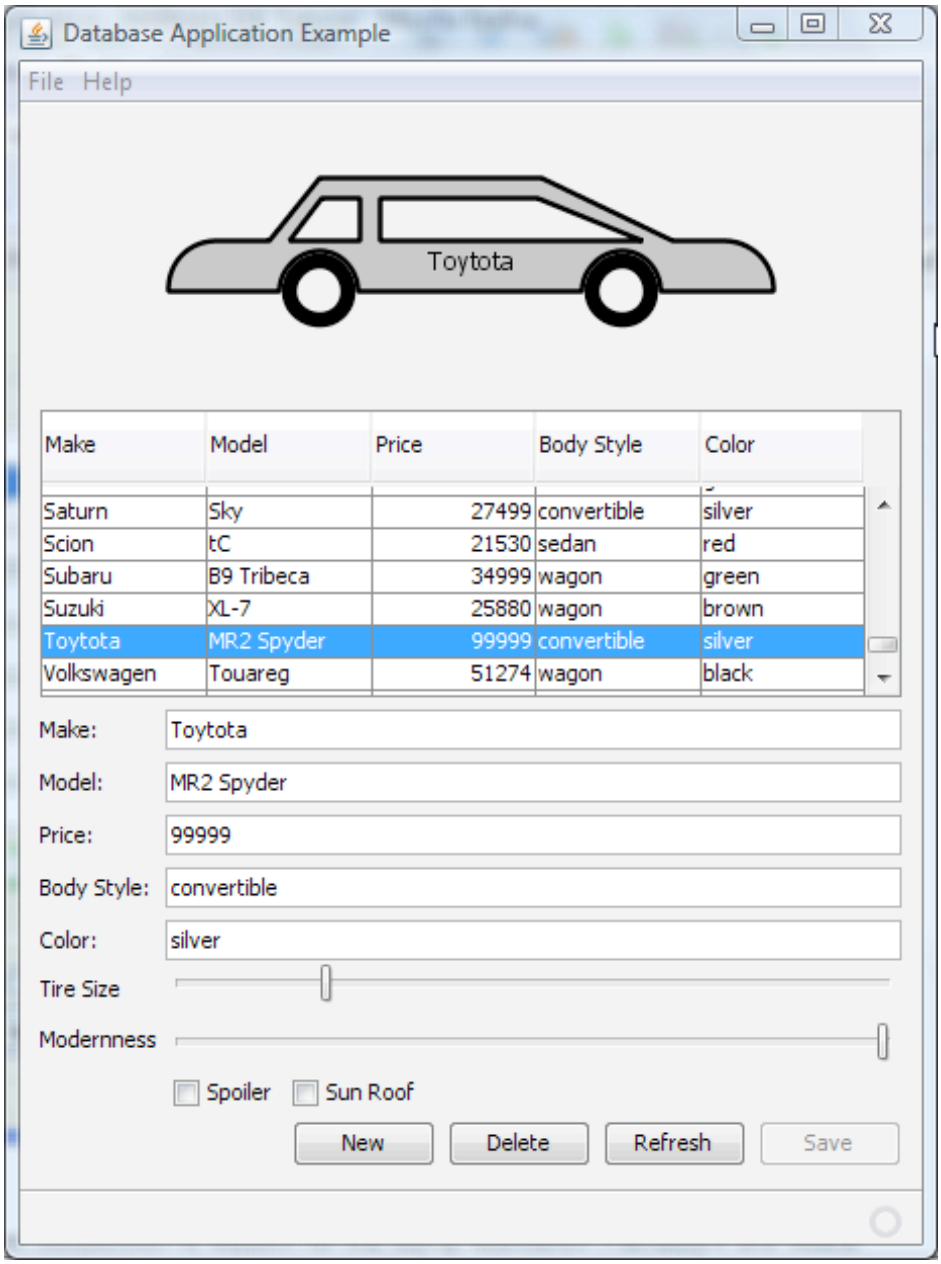
To add the CarPreview component to the application:

1. In the Projects window, expand the Car Preview > Source Packages > carpreview > nodes.
2. Drag the CarPreview.java class to the form. To insert it properly just below the menus, place it over the table left aligned with the other controls and snapped to the bottom of the menu bar, as shown in the image below.



3. Resize the preview component horizontally over the entire form.
4. In the same way that you bound the sliders and checkboxes to elements in the `masterTable` component, bind all the binding properties of the CarPreview component to the corresponding selectedElement attributes of the `masterTable`. Use the Bind popup menu or the Binding tab in the property sheet.
5. Run the CarApp application again.
   In the running application, you can see the CarPreview component change as you select different rows in the table, alter values in the table, move the sliders, and select and deselect the checkboxes.

The image below shows the final running application.



Building and Deploying the Application

Once you are satisfied that your application works properly, you can prepare the application for deployment outside of the IDE. In this section you will build the application's distributable outputs and run the application from outside of the IDE.

Building the Application

The main build command in the IDE is the Clean and Build command. The Clean and Build command deletes previously compiled classes and other build artifacts and then rebuilds the entire project from scratch.

> **Notes:** There is also a Build command, which does not delete old build artifacts, but this command is disabled by default. See the Compile on Save section of the Creating, Importing, and Configuring Java Projects guide for more information.

To build the application:

- Choose Run > Clean and Build Main Project (Shift-F11).

Output from the Ant build script appears in the Output window. If the Output window does not appear, you can open it manually by choosing Window > Output > Output.

When you clean and build your project, the following things occur:

- Output folders that have been generated by previous build actions are deleted ("cleaned"). (In most cases, these are the `build` and `dist` folders.)

- `build` and `dist` folders are added to your project folder (hereafter referred to as the *PROJECT_HOME* folder).
- All of the sources are compiled into `.class` files, which are placed into the *`PROJECT_HOME`*`/build` folder.
- A JAR file containing your project is created inside the *`PROJECT_HOME`*`/dist` folder.
- If you have specified any libraries for the project (in addition to the JDK), a `lib` folder is created in the `dist` folder. The libraries are copied into `dist/lib`.
- The manifest file in the JAR is updated to include entries that designate the main class and any libraries that are on the project's classpath.

Running the Application Outside of the IDE

To run the application outside of the IDE:

1. Start Java DB from outside of the IDE. See http://db.apache.org/derby/docs/dev/getstart/

   Note: You can also start Java DB from inside the IDE, but the server will be stopped when you exit the IDE.

2. Using your system's file explorer or file manager, navigate to the `CarsApp/dist` directory.
3. Double-click the `CarsApp.jar` file.

After a few seconds, the application should start.

Note: If double-clicking the JAR file does not launch the application, see this article for information on setting JAR file associations in your operating system.

You can also launch the application from the command line.

To launch the application from the command line:

1. Start Java DB from outside of the IDE.
2. On your system, open up a command prompt or terminal window.
3. In the command prompt, change directories to the `CarsApp/dist` directory.
4. At the command line, type the following statement:

```
java -jar CarsApp.jar
```

Distributing the Application to Other Users

Now that you have verified that the application works outside of the IDE, you are ready to distribute the application.

To distribute the application:

1. On your system, create a zip file that contains the application JAR file (`CarsApp.jar`) and the accompanying `lib`, which contains the other JAR files that the application needs.
2. Send the file to the people who will use the application. Instruct them to unpack the zip file, making sure that the `CarsApp.jar` file and the `lib` folder are in the same folder.
3. Instruct the users to follow the steps in the Running the Application Outside of the IDE section above.

Next Steps

This tutorial has provided an introduction to support for the Swing Application Framework and Beans Binding in the IDE.

For information on creating an application that includes one-to-many and many-to-one relationships and separate data entry dialog boxes, see

Creating a Custom Desktop Database Application.

For information on using Hibernate for a Swing application's persistence layer, see Using Hibernate in a Java Swing Application.

For a more information on designing GUI applications, see Designing a Swing GUI.

For more information on using beans binding in the IDE, see Binding Beans and Data in a Desktop Application.

Send Us Your Feedback

See Also

- Basic IDE and Java Programming Learning Trail.
- Beans Binding project page on java.net
- Swing Application Framework project page on java.net
- GUI Editor FAQ
- Patrick Keegan's web log on Swing development in NetBeans IDE
- Java EE Tutorial: Introduction to the Java Persistence API