Object-Oriented Design and Programming in LabVIEW™ Exercises

Course Software Version 2010
November 2010 Edition
Part Number 325617A-01

Worldwide Technical Support and Product Information
`ni.com`

National Instruments Corporate Headquarters
11500 North Mopac Expressway   Austin, Texas 78759-3504   USA   Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 5050 9800, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 358 (0) 9 725 72511, France 01 57 66 24 24, Germany 49 89 7413130, India 91 80 41190000, Israel 972 3 6393737, Italy 39 02 41309277, Japan 0120-527196, Korea 82 02 3451 3400, Lebanon 961 (0) 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 (0) 348 433 466, New Zealand 0800 553 322, Norway 47 (0) 66 90 76 60, Poland 48 22 328 90 10, Portugal 351 210 311 210, Russia 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222, Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

To comment on National Instruments documentation, refer to the National Instruments Web site at `ni.com/info` and enter the Info Code `feedback`.

Contents

4

Object-Oriented Tools and Design Patterns

Exercise 4-1 Channeling Pattern

Goal

Use the channeling pattern to guarantee the execution of pre-processing and post-processing steps in your application algorithm.

Scenario

Create the top-level methods for `Application.lvclass` and `Sound Demo.lvclass`. The main VI for the project will exist in `Sound Demo.lvclass`. The overall application algorithm will be defined by `Application.lvclass`.

Design

The top-level public method of Application should be named `Run.vi`. It should guarantee execution of the following steps, in order:

1. **Setup**—This method does nothing for the abstract Application class. Child classes should override this method with any work needed to set up the main body of the execution. This includes requesting resources that will be needed for the entire duration of the execution and reading any configuration files. Child classes are not required to override Setup because there are plenty of applications that can jump straight into their main execution logic.

2. **Execute**—This method does nothing for the abstract Application class. This VI MUST be overridden by child classes. Children should place their main execution logic in this method. This method should only be called if Setup does not return any errors.

3. **Shutdown**—This method does nothing for the abstract Application class. Child classes should override this method with any work needed to quit the application. This includes releasing any resources acquired during Setup and writing any configuration files if options changed during the run of the application. This method will only be called if no error was returned from Setup and it will be called even if Execute returns an error.

4. **Handle Error**—This method is called downstream of the other methods so it can detect any error that originated in Setup, Execute, or Shutdown. It does not see any errors that were passed into Run. In other words, it only detects errors generated by this application. Child classes may override this VI to do error logging, throw error dialogs, or other appropriate actions when their application has an unclean exit. Handle Error does not have any way to restart the application. It is for error management, only.

The top-level method of `Sound Demo.lvclass`, `Main.vi`, should call into `Application.lvclass:Run.vi`, using a Sound Demo class constant to determine which implementation of Setup, Execute, and Shutdown should be called. This is the main VI of the project and is the starting point for demonstrating sound players.

Implementation

Implement Application.lvclass:Run.vi

1. Open `Sound Player Demo.lvproj`, if it is not already open.

2. Create the Run method to define the execution algorithm for Application and its children.

   ❑ Right-click **Application.lvclass** and select **New»VI from Static Dispatch Template**.

   ❑ Save the VI as `Run.vi` in `<Exercises>\...\Sound Player Demo Project\Application`.

3. Build the block diagram shown in Figure 4-1, using methods from `Application.lvclass`.

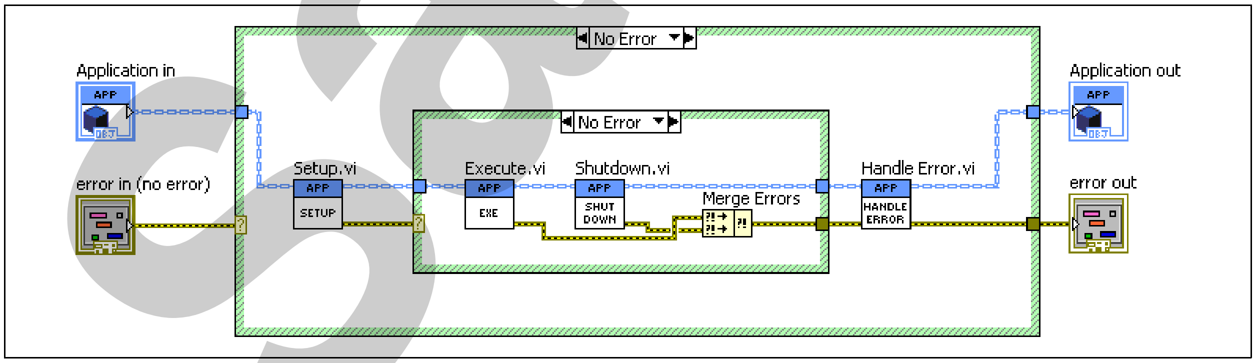

Figure 4-1. Application.lvclass:Run.vi Block Diagram

Notes Do not pass the error wire into Setup, Execute, or Shutdown because you have already guaranteed with the Error/No Error Case Structures that these method will never execute when an upstream error has occurred. By placing these structures in Run.vi, we have removed the need for each dispatch version of Setup, Execute and Shutdown to replicate the implementation of Error/No Error Case structures.

The error wire from `Execute.vi` is wired to the top terminal of the Merge Errors function to give it priority over errors generated by `Shutdown.vi`.

The error wire is *not* wired from `Execute.vi` to `Shutdown.vi`. This is intentional, because you want to be sure that any resources allocated in `Setup.vi` are closed in `Shutdown.vi`, regardless of whether or not `Execute.vi` generated an error.

4. Resize the front panel to remove unnecessary empty space, as shown in Figure 4-2.

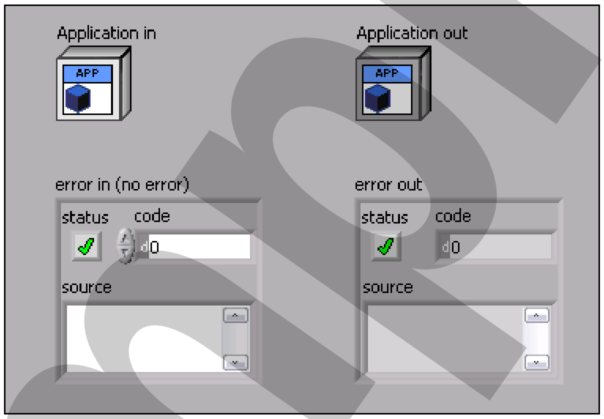

Figure 4-2. Applciation.lvclass:Run.vi Front Panel

5. Modify the icon and connector pane as shown in Figure 4-3.

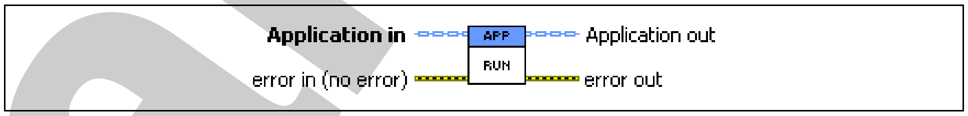

Figure 4-3. Application.lvclass:Run.vi Icon and Connector Pane

6. Document the purpose of this method.

   ❑ Select **File»VI Properties**.

   ❑ In the VI Properties dialog box, select **Documentation** from the Category list.

   ❑ In the **VI Description** text box, describe the purpose of this method. For example: `This VI contains the main architecture for all Applications. It uses the channeling pattern to ensure that each implementation will follow the same basic steps: Setup, Execute, Shutdown, Handle Error.`

❑ Click **OK** to close the VI Properties dialog box.

❑ Save and close `Run.vi`.

Implement Sound Demo.lvclass:Main.vi

1. Create the main VI for the Sound Player Demo project. This VI uses the algorithm defined by `Application.lvclass:Run.vi`.

   ❑ Right-click **Sound Demo.lvclass** and select **New»VI**.

   ❑ Save the VI as `Main.vi` in `<Exercises>\...\Sound Player Demo Project\Sound Demo`.

2. Create the block diagram of `Main.vi` as shown in Figure 4-4.

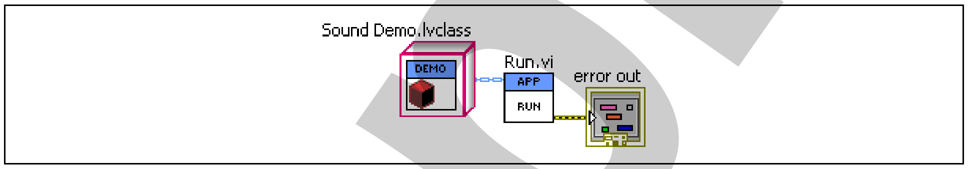

Figure 4-4. Sound Demo.lvclass:Main.vi Block Diagram

Note The `Sound Demo.lvclass` constant is passed into `Application.lvclass:Run.vi` to determine which implementations of Setup, Execute, and Shutdown to dynamically dispatch to.

3. Modify the front panel of `Main.vi` as shown in Figure 4-5.

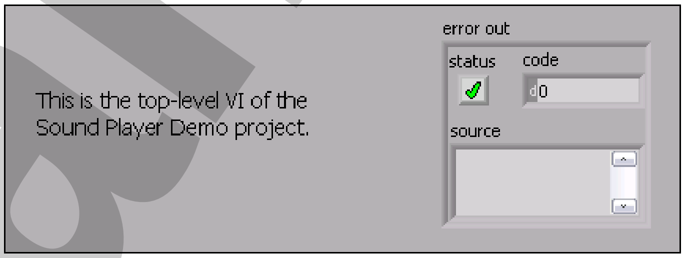

Figure 4-5. Sound Demo.lvclass:Main.vi Front Panel

4. Modify the icon and connector pane of `Main.vi` as shown in Figure 4-6.

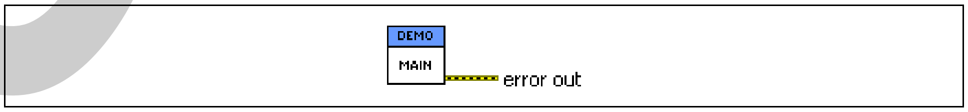

Figure 4-6. Sound Demo.lvclass:Main.vi Icon and Connector Pane

5. Document the purpose of this method.

   ❑ Select **File»VI Properties**.

   ❑ In the VI Properties dialog box, select **Documentation** from the Category list.

   ❑ In the **VI Description** text box, describe the purpose of this method. For example: `This is the top-level VI that starts the Sound Player Demo application.`

   ❑ Click **OK** to close the VI Properties dialog box.

6. Save and close the VI.

Test

At this point, you will test `Sound Demo.lvclass:Main.vi`. The application currently has most of the functionality. The only feature that remains is the ability to play sounds on either the Sound Visualizer or the Sound Card.

Currently, `Sound Demo.lvclass:Setup.vi` always plays sounds on the Sound Visualizer. You implement the ability to select either player in Exercise 4-2.

For now, verify that the application plays sounds on the Sound Visualizer.

1. In Windows Explorer, navigate to `<Exercises>\...\Sound Player Demo Project\Sound Demo Support Files`.

2. Select `sample.wav`, `Sound XML File Creator.vi`, and `sounds.xml`. Drag these three files into the **Sound Demo Support Files** virtual folder in `Sound Player Demo.lvproj`.

3. Open `Sound XML File Creator.vi`. Explore the block diagram to understand the functionality that has been implemented.

4. Run `Sound XML File Creator.vi`. This VI creates an XML file containing Sound data that will be read by `Sound Demo.lvclass:Execute.vi` and used to create the waveforms that are played.

5. After running `Sound XML File Creator.vi`, open and run `Sound Demo.lvclass:Main.vi`. After configuring the visualizer, you should see a series of waveforms play on the Visualizer.

6. Click **OK** in the Sound Demo dialog box.
7. Save and Close all VIs.
8. In the project window, select **File»Save All** to save the project and the class modifications that have been made.

End of Exercise 4-1

Exercise 4-2 Factory Pattern

Goal

Implement the factory pattern to place the appropriate child class data onto a parent class wire to execute the appropriate dynamic dispatch method.

Scenario

Modify `Sound Demo.lvclass:Setup.vi` so that it dynamically dispatches to the appropriate implementation of `Initialize With Dialog.vi` based on the type of Sound Player selected by the user.

The factory pattern is easily extended by adding additional child classes to the parent type.

Design

`Sound Demo.lvclass:Setup.vi` currently uses a `Sound Visualizer.lvclass` constant to always play sounds on the Sound Visualizer. We want to modify this code so that the user can specify either the Sound Card or the Sound Visualizer to play the sounds.

This implementation will occur in two stages:

1. Replace the `Sound Visualizer.lvclass` constant with an enumerated control wired to a case structure that will pass either a `Sound Card.lvclass` constant or a `Sound Visualizer.lvclass` constant, depending on the enumerated value selected. This will allow the demo to play sounds using either the Sound Card or the Sound Visualizer.
2. Replace the `Sound Visualizer.lvclass` and `Sound Card.lvclass` constants with code to dynamically load the appropriate child of `Sound Player.lvclass` based on the enumerated value selected. When this code is built into an executable, this will allow the Sound Player Demo to execute without having to load both classes and all of their methods into memory every time the code executes.

Implementation

Implement the Factory Pattern

1. Open `Sound Player Demo.lvproj` if it is not already open.
2. Open `Sound Demo.lvclass:Setup.vi`.

3. Modify the front panel of `Setup.vi` as shown in Figure 4-7.

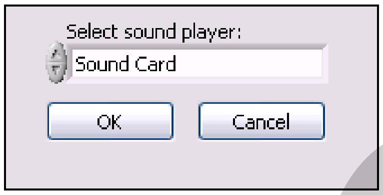

Figure 4-7. Sound Demo.lvclass:Setup.vi Front Panel

- ❑ Select sound player—This is a ring control with two values: Sound Card and Sound Visualizer.
- ❑ Configure this control as a **Strict Type Definition** and save it as `Select Sound Player.ctl` in `<Exercises>\...\Sound Player Demo Project\Sound Demo`.
- ❑ Add `Select Sound Player.ctl` to `Sound Demo.lvclass`.

4. Modify the Case Structure following the While Loop as shown in Figure 4-8 so that the structure will pass a different child of Sound Player.lvclass based on the sound player selected by the user.

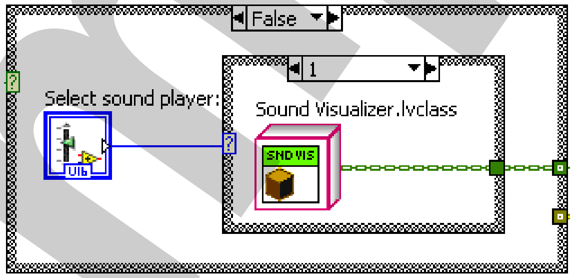

Figure 4-8. Sound Demo.lvclass:Setup.vi Block Diagram—Select Sound Player Case Structure

- ❑ Modify the `0, Default` case of the inner Case structure so that it passes `Sound Card.lvclass` in the same way that the `1` case shown in Figure 4-8 passes `Sound Visualizer.lvclass`.

5. Save and close `Setup.vi`.

6. Open and run `Sound Demo.lvclass:Main.vi` to verify that you are able to select either the visualizer or the system sound card to play the sounds. Verify that both players work.

Note If sound drivers are not installed on your system, attempting to use the Sound card results in an error.

Implement Dynamic Loading of Classes in the Factory Pattern

1. Create a new method named `Load Sound Player.vi` for `Sound Demo.lvclass`.

   ❑ Right-click **Sound Demo.lvclass** and select **New»VI**.

   ❑ Save the VI as `Load Sound Player.vi` in `<Exercises>\...\Sound Player Demo Project\Sound Demo`.

2. Modify the block diagram of as shown in Figure 4-9 so that it dynamically loads the appropriate class from disk based on which type of player the user selects.

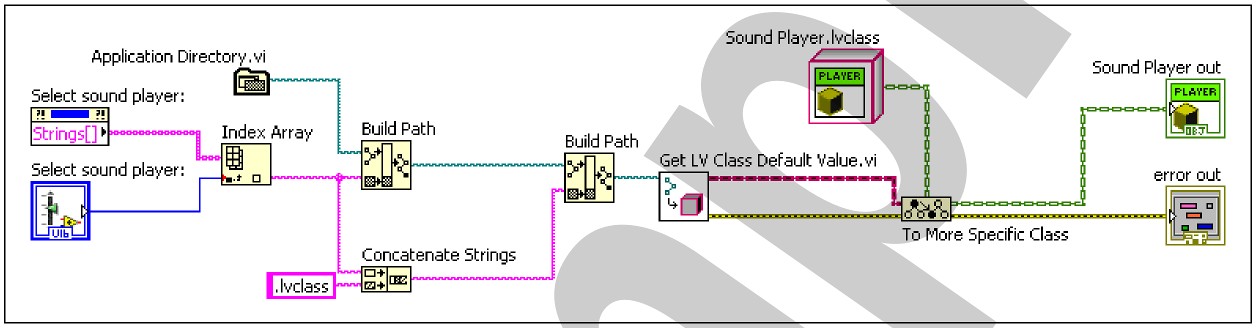

Figure 4-9. Sound Demo.lvclass:Load Sound Player.vi Block Diagram

Note This VI builds the path to the class files using the string values of the Select sound player: ring control. This removes the need for case structures and enables you to make a single change to the ring control to expand functionality to a new Sound Player.

3. Modify the front panel of `Load Sound Player.vi` as shown in Figure 4-10.

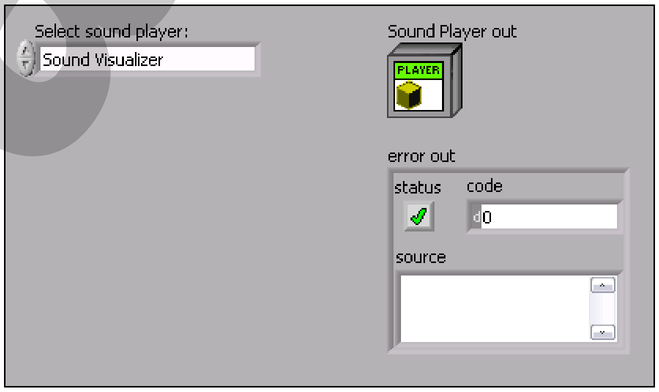

Figure 4-10. Sound Demo.lvclass:Load Sound Player.vi Front Panel

4. Modify the icon and connector pane of Load Sound Player.vi as shown in Figure 4-11.

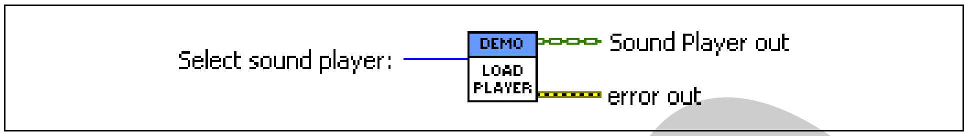

Figure 4-11. Sound Demo.lvclass:Load Sound Player.vi Icon and Connector Pane

5. Document the purpose of this method.

   ❑ Select **File»VI Properties**.

   ❑ In the VI Properties dialog, select **Documentation** from the Category list.

   ❑ In the **VI Description** text box, describe the purpose of this method. For example: `Dynamically load either Sound Visualizer or Sound Card.lvclass based on user input.`

   ❑ Click **OK** in the VI Properties dialog box.

6. Save and close `Load Sound Player.vi`.

7. Open `Sound Demo.lvclass:Setup.vi` and modify the Case structure that loads the Sound Player as shown in Figure 4-12.

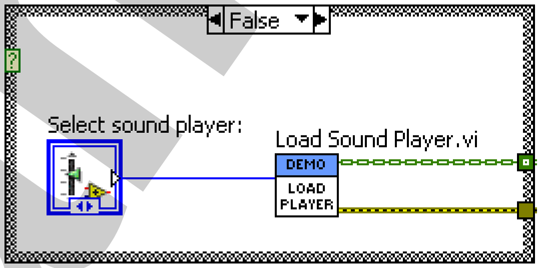

Figure 4-12. Sound Demo.lvclass:Setup.vi Block Diagram—Dynamically Loading Classes

8. Verify that classes are now dynamically loaded into memory.

   ❑ Remove `Sound Visualizer.lvclass` and `Sound Card.lvclass` from the project. This is done so that we can build an executable that will only load the child of Sound Player into memory that is actually used.

9. In the project window, select **File»Save All** to save the project and all of the class modifications that have been made.

   - ❑ Expand **Dependencies** in the project window. Notice that `Sound Card.lvclass` and `Sound Visualizer.lvclass` are listed as Items in Memory.
   - ❑ Close and reopen `Sound Player Demo.lvproj`.
   - ❑ Expand **Dependencies** in the project window. Notice that `Sound Card.lvclass` and `Sound Visualizer.lvclass` are no longer listed.
   - ❑ Open and run `Sound Demo.lvclass:Main.vi` and select **Sound Card** as the sound player.
   - ❑ When the VI finishes, expand **Dependencies** and notice that only `Sound Card.lvclass` is listed under **Items in Memory**.

Note `Sound Card.lvclass` was loaded in memory when it was called.

   - ❑ Run `Sound Demo.lvclass:Main.vi` again, and select **Sound Visualizer** as the sound player.
   - ❑ When the VI finishes, expand **Dependencies** and notice that `Sound Visualizer.lvclass` is now also listed under **Items in Memory**.

Note You can dynamically load VIs, but you cannot dynamically unload them.

End of Exercise 4-2

Notes