CHAPTER 19

Menus and Toolbars

Man is a tool-using animal….
Without tools he is nothing, with tools he is all.

—Thomas Carlyle (1751–1881), *Sartor Resartus*

Every user of client/server applications is familiar with the traditional design elements of menus and toolbars. Both provide easy-to-use and well-understood user interfaces for many application functions. Menus enable users to execute the standard tasks in an application. Toolbars enable users to execute the most commonly used tasks in the menu. Providing these two elements to your users gives them the tools to perform their work most efficiently. Menus and toolbars in Java applications and applets work exactly like their counterparts in the client/server world.

This chapter explains some general considerations for designing menus and toolbars for Java applications and applets (which this book refers to as "Java client applications"). The best way to describe how to create menus and toolbars in JDeveloper is to step through examples. The hands-on practices at the end of the chapter supply such examples.

NOTE
While the design considerations in this chapter apply to any style of development, the techniques describe the use of menu and toolbar components from the Swing and AWT libraries. Designing menus and toolbars for a JSP light-client application has similar design concerns, but the controls that you use are different in appearance and in development. This chapter emphasizes Swing components, which most developers prefer because of their superior functionality.

Design Considerations

An integral part of any user-interface design is determining how the user will perform the actions required to complete a task. For example, when designing an online transaction processing application, you have to decide how the user will add, search, modify, delete, and save data. The first decision you must make is whether you will use menus and toolbars at all. If you do not use them, you have to decide how to supply the functionality that they normally provide. The deployment method that you select for an application will, to a large extent, help you with this decision.

In a client/server application, menus and toolbars are natural features that users easily understand and expect to see. Since applets on the Web emulate the controls that users are accustomed to, menus and toolbars have a fairly standard appearance. If the application is deployed through servlets, JavaServer Pages (JSP) applications, or other HTML interfaces, menus and toolbars may appear, but can take on many different formats.

The role of menus in an HTML environment is played by textual links, graphical links, or navigation bars. The role of toolbars is played by buttons or icons on the page. Pulldown menus sometimes appear in HTML applications through the use of JavaScript (because HTML does not support the standard menu look-and-feel). The functions that menus and toolbars deliver are integrated into design elements of the website, and that design is often unique to each site.

Once you have decided that the application requires menus and toolbars, you need to design the layout and organization and determine which functions you want to provide. There are some general factors to consider and guidelines to follow when creating this design.

TIP
Refer to Sun Microsystems' website for more design information about menus and toolbars. You can connect to http://java.sun.com/products/jlf/ed1/dg/higm.htm or search for "design menus and toolbars" in the java.sun.com website.

What Do You Put on a Menu?

When structuring a menu system, it is important to emulate standard menu items that users are accustomed to in most Windows applications. This will lessen the learning curve that the user interface may require and speed up user acceptance of your application. Menu design should take into consideration the organization of elements that your users expect. For example, if your menu structure contains File, Edit, View, Window, and Help menus, users will quickly understand where to look for a particular function because these menus are commonly used in Windows applications. While most Windows client/server applications interact with files, your applications will more commonly interact with a database, so you may have to stretch the meaning of the item names in some cases. A standard menu structure follows:

File Menu This menu usually contains Open, New, Save, Close, and Exit items among others. You can provide those same items in data-aware Java applications, even though the concept of file interaction does not usually apply. An Open item could select a table or application to browse; a Close item could clear current data from the form; and Save could commit the changed data.

Edit Menu Menu items may add or remove records or otherwise manipulate data. For example, you can include an item to copy an existing record or to fill the current record with default values. Another common item to include on the Edit menu is one for Options that allows users to modify personal preferences such as colors, fonts, and backgrounds. Options may also include how default values are filled in and what is shown when the user opens the application—for example, a find window or an automatic query of all records.

View Menu Items on this menu may navigate to a particular section or record. The View menu can also contain items for Find, Sort, and Filter to modify how a set of records is displayed. You can also allow users to display or hide a toolbar using a check mark menu item.

Application-Specific Menus These menus, which should be placed to the left of the Window menu, will provide functionality that is specific to the application. It is best to keep the number of other menus to a minimum so that the user is not overwhelmed with choices. As a rule of thumb, a maximum of ten pulldowns (main menu headings), each with a maximum of 15 selections, will give users up to 150 choices. This should be plenty for a standard application. If the items are logically arranged, users will be able to find a function easily without having to browse the menus frequently.

Selections in other menus might include functions such as navigating to other applications, stepping the user through a difficult task, or using wizards to enter data.

Window Menu The items on this menu can be used to arrange the various open windows and allow navigation between them.

Help Menu Items perform the expected calls to the help system or Help About dialog.

Other Menu Features

Menus offer other design features. You can take advantage of all of these in JDeveloper, but some may require extra coding.

Nested Menus

Menus can contain nested submenus, as shown in an example of the JDeveloper IDE menu in Figure 19-1. In other words, when you select a menu item from a pulldown, another side menu opens for the actual selection. In this example, when you select Refactor from the Tools menu, a submenu containing three choices opens.

Design Suggestion Although you can create menus to virtually any level of depth, a good rule of thumb is to limit yourself to three levels, as in Figure 19-1. Using more than three levels requires some extra dexterity from your user and can lead to frustration if the user cannot easily select a menu item.

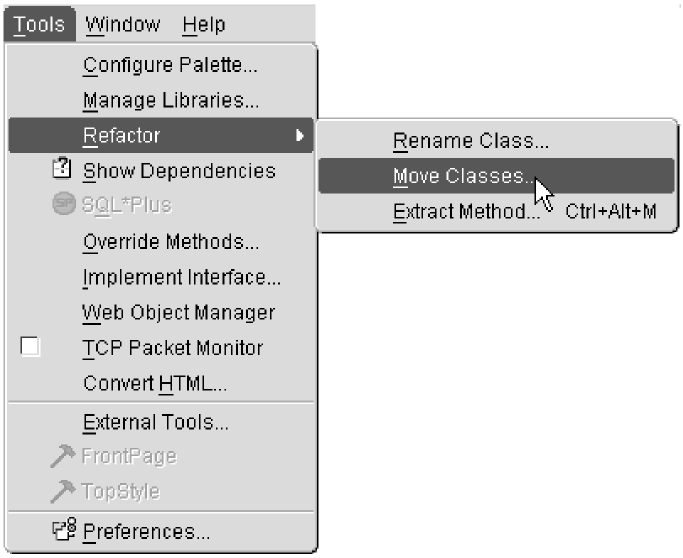
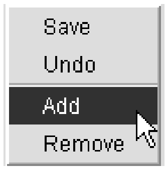


FIGURE 19-1. *Multi-level menu*

Popup Menus

You can define *popup menus* that appear when the user clicks the right mouse button. Popup menus are not attached to the top of the window, but normally appear at the mouse cursor location when the secondary mouse button (right button for right-handed mouse users) is clicked on an object. They are also called *context menus* or *right-click menus* because they appear based on the context or location of the mouse cursor. Popup menus are usually customized for the object that was clicked and allow quick access to functions that apply only to that object. For example, right clicking a text item could display a menu for Cut, Copy, and Paste to manipulate the text in the item. Right clicking a panel in a data form might display a menu that allows users to add or remove records and save or undo changes. Users find right-click (more properly called "alternate-button click" to accommodate left mouse users) menus a fast way to get to a particular menu item. They require less mouse movement than using a pulldown menu to access the required functions. An example of a popup menu is shown to the right.



Design Suggestion It takes some training to make users aware of and accustomed to the functionality on popup menus. However, if your application requires repetitive actions to access or input data, popup menus may help. Examine the possible ways that your application will be used when determining what items to include on popup menus, and customize them for the most commonly used menu items that a user would require when working with a particular object. Popup menu items are usually items that also appear on the main application menu.

Design the popup menu as simply as possible. For example, nested submenus within a popup menu make the popup menu difficult to use. Popup menus often contain fewer items than a normal pulldown menu, and you may want to set a limit of six or eight items (plus separators).

NOTE
As mentioned in the Introduction, this book calls context or popup menus "right-click menus."

Check Mark and Radio Group Items

Menu items can appear as normal items or with check marks or radio buttons. A *check mark menu item* represents a single state of a toggle, such as displaying or hiding a toolbar. For example, you might have a View menu that contains a check mark menu item (Toolbar) that displays the component toolbar. If the menu item is checked, the toolbar is displayed. If users select the item when it is checked, the check mark vanishes and the toolbar is hidden.

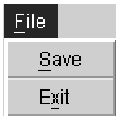
Radio group menu items offer more than one option and display the selected option with a round bullet icon. If you create a group of items, only one of the group items will be "checked" at once. The menu item functionality will take care of the visual display, but you need to write code to handle what happens when the user makes a selection.

For example, you might have a main menu item called Sort that is a submenu. When a user selects the item, a nested menu appears with a radio group of items for the columns that will be sorted (such as Name, Hire Date, Salary, and so on). When the user selects one of this group, the radio button next to the item will appear filled in, and the code you write would re-sort the display based upon that column.

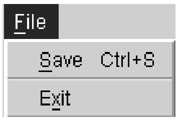
Design Suggestion Other than the View menu suggestions just mentioned, check and radio group menu items are not frequently used. It is more common to provide the functionality that they offer using an options dialog.

Mnemonics and Accelerators

Mnemonics are quick access keys that help users select a menu or menu item without using the mouse. Pulldown menus are typically activated using the ALT key combined with the first letter in the menu text. For example, to activate the File pulldown, the user would press ALT-F. In this example, "F" would be underlined to indicate to the user that this was the key to press with the ALT key. The user accesses a menu item within the pulldown using the underscored letter. For example, the Exit item in the File menu would be activated when the user pressed the X key. The letter used as the mnemonic must appear in the menu item's text property. An example of the underlined letters that indicate mnemonics is shown to the left.



An *accelerator key* is a shortcut to the functionality that is offered in the menu. The user can press this key combination to run the same code that the menu item activates. This saves the user from having to interact with the menu. Mnemonics and accelerators are similar, but accelerators do not display the menu selection and are usually a CTRL key combination instead of the mnemonic's ALT key combination. The accelerator key combination appears next to the menu item, as shown here for the Save function.



NOTE
The Swing class JMenuBar uses a property called "accelerator" to supply the functionality of an accelerator. If you use the AWT class MenuBar, the property name is "shortcut."

Design Suggestion When you make the decision about which mnemonics and accelerators to include, a good rule of thumb is that you should provide mnemonics for every menu selection so that a user without a mouse can successfully navigate within your application. It is important to respect the standards supported by existing applications. For example, you should provide the commonly known accelerators such as CTRL-S for Save. Examine common Windows programs to get a feeling for the common accelerators. In addition, you should avoid reassigning commonly used accelerators such as CTRL-X, CTRL-C, and CTRL-V that usually represent the functions Cut, Copy, and Paste, respectively.

It is important to think about accelerators and mnemonics because they allow users an alternative to using the mouse to access application functions. This is particularly critical for users who, because of a visual or other disability, are unable to use a mouse.

Disabling and Enabling Items

Many modern applications disable menu options that are not applicable to a certain mode. For example, if you have not made any changes to data on a form, the Save option is not required and therefore should be disabled. When a change is made to the data, the item would be enabled.

Design Suggestion Well-designed applications enable and disable menu items at the appropriate time, and this feature is the ultimate in user friendliness. However, disabling options requires a bit of code, and you have to weigh the benefits of this friendliness against the extra time and effort involved in coding and debugging. Depending upon your target audience's tolerance and the frequency with which a function may be accessed, issuing a dialog to indicate that a function is not available in a certain mode will serve the purpose and not cause undue user frustration.

Menu Item Icons

You can associate a .gif file with a menu or menu item. You can display the menu text with the graphic.

Design Suggestion Although this technique allows you to associate a visual clue with the text, it is unclear whether menu icons assist the user in more quickly identifying a menu item. The extra space that the graphic requires is probably not worth the benefit in most situations.

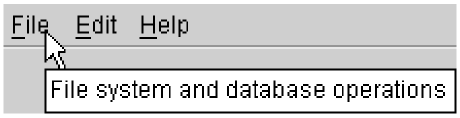
Separators

A *separator* is a thin horizontal line that is not user-selectable and contains no functionality.

Design Suggestion Arrange the menu so that the items are grouped by similar functionality, and add a separator between groups. For example, an Edit menu may contain, among others, functions for Cut, Copy, and Paste. Those functions are logically similar and may be grouped using separators before the first and after the last item. The menu objects for this example would be Separator, Cut, Copy, Paste, Separator. Another use for separators is to visually group a set of radio group menu items.

Tooltips

Menu items can have tooltip text associated with them. This text pops up when the user rests the mouse over the item, as the following shows:



Design Suggestion Menu tooltips are useful only if menus or menu items are hard to understand. Since you should strive for intuitive wording in your applications, you should not need this feature for most menu items.

NOTE
You will not see accelerators, mnemonics, or icons in the Menu Editor. If you define them in the properties or code, they will appear only at runtime.

What Do You Put on the Toolbar?

Think of the toolbar as a subset of the menu. All functions on the toolbar should also be available in the menu. Therefore, it is best to design the menu first and repeat frequently used menu selections in the toolbar. Examples of commonly used items for a transaction processing system would be Save and Undo. Users need to interact with data frequently, and saving and undoing changes are often-performed tasks. You may have to validate your designation of commonly used functions with a user trial of your application.

Toolbars are usually iconic in nature. In other words, the user selects a toolbar button by identifying an iconic picture located in a panel. This saves space and clutter on the screen over the alternative of identifying the buttons with text labels. Toolbar buttons are also a standard size as opposed to text buttons that are usually sized to their labels. Although there is a bit of a learning curve for users to understand what the pictures represent, it is faster for users to find a picture in the toolbar than a word once that learning curve is overcome.

Other Toolbar Features

Many features of menus are also available for toolbars. However, the design considerations are different for toolbars and menus, as follows:

- **Icons** While you may not choose to use icons on menu items, you would usually use them on toolbar buttons.
- **Mnemonics and accelerators** If you follow the guideline that toolbar items are derived from menu items, the menu items will fulfill the need that mnemonics and accelerators serve. Therefore, you do not need mnemonics and accelerators for toolbars.
- **Tooltips** While you do not use tooltips for menu items, you normally supply tooltips for an iconic toolbar button so that the user can get a hint about the function that the button performs.
- **Enabling and disabling items** You might want to go through the same process as you do with menus to determine how you will handle enabling and disabling items. Since toolbar buttons duplicate functionality on the menu, you can extend the code you write to disable and enable a menu item to include the toolbar button with the same functionality.

You also need to consider multiple toolbars and toolbar arrangements when designing toolbars.

Multiple Toolbars

After you decide what to put in the toolbar, you need to determine how many toolbars to supply. Sometimes, there are more items to put in a toolbar than there is horizontal space in the window. Also, there may be logical, task-oriented groupings of buttons that you might want to represent.

The solution for these issues is to supply more than one toolbar. Each user may work with your application in a different way, and you can account for this by allowing the user to select which toolbar is displayed using an options dialog.

Design Suggestion Use multiple toolbars if you have many functions that need to be represented or if you think some users might want to have control over the groups of buttons that are displayed. If the latter is the case, you have to write code to enable users to specify which toolbars are displayed.

Toolbar Arrangements

Although toolbars normally appear on the top of the window just under the menu, you can also build toolbars in a vertical orientation to leave more vertical space for user-interface objects.

You can also define the toolbar as part of a separate window or as a panel that the user can "undock" from the main window so that it floats outside the main window. This gives the user the option to shrink the main window but still have all elements visible without scrolling. The toolbar buttons will be easily available in this arrangement.

Design Suggestion Horizontal toolbars are a standard and expected interface object. Vertical toolbars are a personal preference that you might want to let the user determine. Floating toolbars are also something that you can supply to the user. Some styles of applications such as drawing programs undock the toolbars to provide more drawing space, but database applications usually appear with the toolbar initially docked to the main window.

Summary of User Access Methods

When you design your menu and toolbar, you have to consider the functions that the users need to access. You also need to plan the method that will provide each of these in the application. The following are the main methods available to application designers:

- Toolbar button
- Menu item (main menu)
- Accelerator key
- Popup menu item

For example, the Cut function is usually available in the Edit menu as well as in the popup menu for items that support data editing. In addition, the design could provide for a toolbar button that offers users the choice of clicking the button for the Cut function. As in most applications, the accelerator CTRL-X would be mapped to the same function.

There is an additional method that you can offer to particular users—the command line. While this is not the normal method for applets, there may be some functions that you would allow the user to perform from a command-line prompt (such as opening or converting a file).

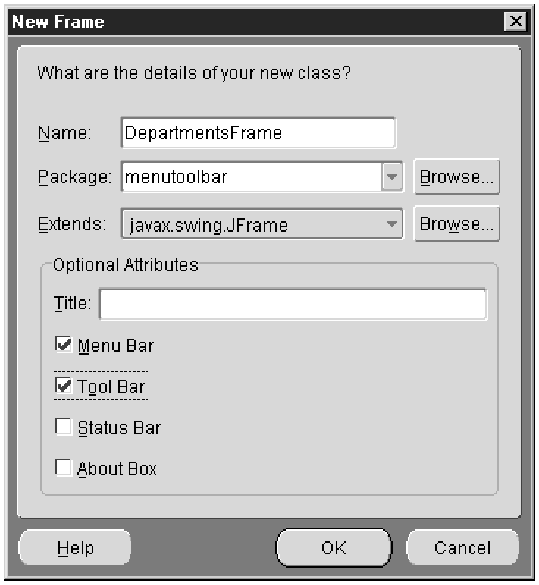
Menus and Toolbars in JDeveloper

JDeveloper supports the creation of Java client applications that include all of the menu and toolbar functionality described earlier. Therefore, you have wide scope when designing the features that you want to include. Some features are easier to implement because JDeveloper writes the code for you. For example, to attach an icon to a button, you just fill in the *icon* property of the button object with the .gif file name, and the `setIcon()` code will be created. Setting the mnemonic and accelerator for a menu item can be accomplished using properties, as the hands-on practices at the end of this chapter demonstrate. Runtime behavior such as disabling and enabling menu items and buttons requires writing custom code. For example, to enable the Save menu item, you would add the following code to the application:

```
saveMenuItem.setEnabled(false);
```

You can automatically generate default menus and toolbars using the wizards. For example, the New Frame dialog that is called from the New Application dialog contains checkboxes for a menu and toolbar, as Figure 19-2 shows. When you check those checkboxes, the New Frame dialog creates a default menu that contains a File menu with an Exit item. The toolbar contains buttons for file open, file close, and help. Both structures contain no code (other than a generic call to exit the application) and are only outlines of what you would use for a real application. Therefore, you need to add code and buttons and modify the contents after the dialog is finished. Other wizards, such as the JClient Form Wizard, also automatically generate a full menu and toolbar. (The JClient Form Wizard is available in the New gallery's Client Tier\Swing/JClient for BC4J category.) In this case, the code for the buttons and menu items is also generated and completely functional.

There are other JDeveloper features for menu objects and toolbar objects that are worth exploring. The hands-on practices at the end of this chapter give you some of the steps to complete when creating menus and toolbars.


New Frame

What are the details of your new class?

Name: DepartmentsFrame

Package: menutoolbar

Extends: javax.swing.JFrame

Optional Attributes

Title:

[✓] Menu Bar

[✓] Tool Bar

[ ] Status Bar

[ ] About Box

FIGURE 19-2. *Specifying a menu and toolbar in the New Frame dialog*

Menu Objects

The object for a menu system that appears on the top of a window is called a *menu bar* (implemented using a Swing component such as JMenuBar). This is the root object that you drag into an application from the JDeveloper Component Palette. A menu bar is a container that holds two kinds of objects: menus and menu items (implemented by components such as JMenu and JMenuItem, respectively). Menus correspond to the headings that you click for a pulldown or submenu (nested menu or menu within a menu). Menu items are the items in the pulldown menu. In the case of multi-level menu systems, submenus look similar to menu items, but include a right arrow to indicate that they display other menu items. For example, in Figure 19-1, the Tools menu contains a selection for Refactor, which is actually a menu because it contains selections such as Rename Class and Move Classes. Therefore, you can say that menus may contain other menus and menu items.

The Menu Editor

The easiest way to lay out a menu is with the Menu Editor feature of the UI Editor. You access this window using the following steps:

1. In the Navigator, select UI Editor from the right-click menu on the frame Java file.

   Additional Information: This displays the UI Editor with GUI components (if any) in the Java file.

2. If a menu exists, expand the Menu node in the Structure window, and click the menu node to switch the display to the Menu Editor as shown in Figure 19-3.

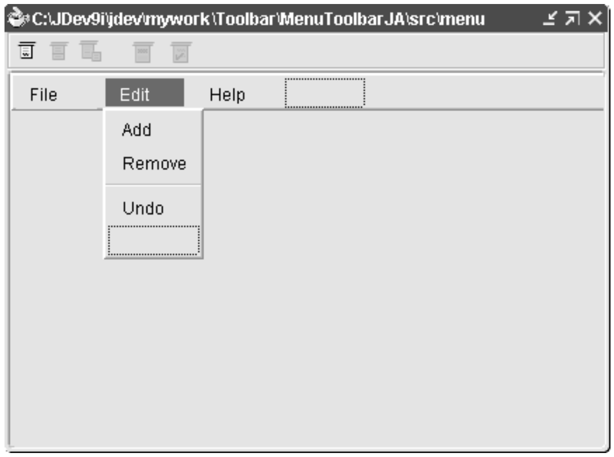


FIGURE 19-3. *Menu Editor*

3. If you want to add a menu, click the Menu node in the Structure window. Select the JMenuBar component from the Swing Containers page of the Component Palette, and drop this component onto the Menu node in the Structure window.

   Additional Information: You can switch back to the UI Editor by clicking any component in the UI node of the Structure window.

NOTE
The JDeveloper Menu Editor contains toolbar buttons such as Insert MenuItem, Insert Separator, and Insert Submenu that you can use to manipulate menus and menu items. You may find that the toolbar buttons speed up your work in the Menu Editor.

The right-click menu for Menu Editor objects contains functions that you use to create the menu structure. After creating the menu structure, you write code that will be activated when a menu item is selected. The right-click menu options allow you to create or delete an item or separator and create a submenu (nested menu). The right-click menu also has selections to convert a normal menu item to a checkable menu item and to disable or enable a menu item. The latter just toggles the *enabled* property from "True" to "False." You can also toggle this property at runtime using the `setEnabled()` method.

You fill in the menu item label in the editor and press ENTER to register the label. This will move the cursor to a blank item where you can type the next text label. The code and properties areas will update as you make changes to the design area. To edit the text for an existing item, double click the item in the Menu Editor.

TIP
You can drag and drop menu items, separators, and submenus within the Menu Editor to rearrange their order. Dropping on top of an item will arrange the dragged item above that item. Selecting a right-click menu option to add an item or separator will add the item or separator above the selected item.

Toolbar Objects

Toolbars are made up of a set of buttons inside a container called a *toolbar.* The toolbar has special properties that allow users to detach the container from the main window. You can use the *floatable* property of the toolbar to specify whether users can detach the toolbar. Since the toolbar is a container, you can manipulate the group of buttons by manipulating the toolbar. For example, if you want to hide the toolbar buttons, you write code to hide the toolbar. All buttons inside the toolbar will also be hidden.

You lay out the toolbar using the UI Editor. The sequence consists of dropping a toolbar container object in the application and adding buttons inside the toolbar.

Buttons require event code to execute the desired action. You can attach icons to buttons so that they have a picture on top. You can also fill in the *text* property with a text string that will appear on top of the button. (AWT objects use the *label* property for the button text.) Normally, toolbar buttons use icons as decoration, but not text labels. Therefore, you need to remove the value of the *text* property (or the `setText()` call in the source code) so that there will be no text label.

A hands-on practice in this chapter steps through the process of creating a toolbar with buttons.

TIP
*As mentioned, you can reorder objects on the Menu Editor by dragging and dropping. You can reorder buttons in a toolbar in the same way, using drag and drop. You can also reorder buttons and menu items by cutting them from the Structure window (*CTRL-X*) and pasting them (*CTRL-V*) on top of another object node in the Structure window. The pasted object will be placed on the top or on the bottom of the list of objects beneath the target object.*

NOTE
The order of objects in the Structure window represents the order in which the objects will be rendered in the application. This ordering scheme is referred to in Java as the "z-order" and is described in a sidebar in Chapter 20 called "A Word About Z-Order."

Using a Navigation Bar

A quick way to create a toolbar that has record and database manipulation buttons is to use a navigation bar. The JClient controls Component Palette page contains a JUNavigationBar class that you attach to a client data model. This navigation bar provides Next, Previous, First, Last, Add, Delete, Save, Undo, Find, and Query functions for the data model. You do not have to write any customized code to make the navigation bar buttons work. A hands-on practice in this chapter contains examples of how to create a navigation bar.

Hands-on Practice: Prepare a Sample Application

The practices that follow require a sample data form Java application as a starting point. You can download this starting application from the authors' websites and skip this practice. Alternatively, you can use the following steps to create this application.

You will use the JClient Form Wizard to create a database interface form for the Departments table.

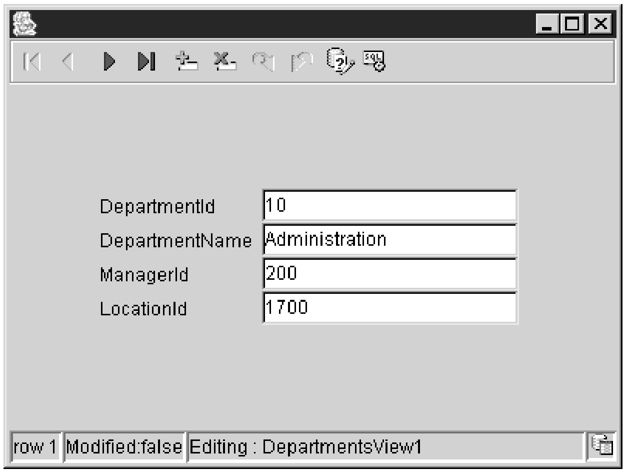
1. In the Navigator, find an existing workspace that contains an HR BC4J project (such as the one created in Chapter 1). Alternatively, you can use the Project Containing New

Business Components item in the New gallery to create a default BC4J project for the DEPARTMENTS table. After you have a BC4J project, select New Empty Project from the right-click menu on that workspace node. The New Project dialog will start.

2. Specify the name of the directory as "MenuToolbarJA." Name the project "MenuToolbarJA." Click OK to create the project.
3. Click Save All.
4. Select New from the right-click menu on the new project node. Double click Form from the Client Tier\Swing/JClient for BC4J category. The JClient Form Wizard will start. Click Next if the Welcome page appears to display the Form Types page.
5. Click Next to accept the defaults on this page (single table with a form), and click Next to accept the defaults on the Form Layout page (a single column with labels to the left). The Data Model page will appear.
6. Click the New button to define a data model. The BC4J Client Data Model Definition Wizard will start. Click Next if the Welcome page appears to display the Definition page.
7. Select the HR project and application module from the pulldowns if they are not already selected and click Next to display the Definition Name page.
8. Click Next to display the Finish page. Click Finish to create the data model and return to the JClient Form Wizard.
9. Click Next to display the Panel View page. Select DepartmentsView1 and click Next to display the Attribute Selection page.
10. The default selections on this page are usable, so click Next to display the File Names page. Enter the following names:
    Package name as "menutoolbar"
    Form name as "DepartmentsForm"
    Master panel name as "DepartmentsPanel"
    Generate a menu bar unchecked
11. Click Next and Finish to exit the wizard and create the files.
12. Open DepartmentsForm in the Code Editor and search for the following line of code:

```
this.setSize(new Dimension(800, 600));
```

13. Change "800, 600" to "400, 300" so that the window will appear smaller when it is run.
14. Click Save All. Click Run to run the DepartmentsForm file. The application will look something like the following:



DepartmentId	10
DepartmentName	Administration
ManagerId	200
LocationId	1700

row 1 | Modified:false | Editing : DepartmentsView1

15. Close the window.

What Just Happened? You used the wizard to create a data form Java application that accesses the Departments table. The wizard also helped you create a client data model that defined the application model for the BC4J project in the same workspace. This data form is fully functional for all data manipulation. However, the form has no menu.

To run this form, you run DepartmentsForm, which is a subclass of JFrame, a Java window. DepartmentsForm calls DepartmentsPanel, which contains the UI elements such as labels and text fields. The next practice will add a menu to the window class (DepartmentsForm.java).

Hands-on Practice: Build a Menu

This practice builds a menu system for the sample application created in the previous practice. Adding a menu to an existing application consists of the following main phases:

I. **Lay out the menu elements**

II. **Set the menu element properties**

III. **Write the menu item code**

As mentioned earlier, there is a preparation step where you design the structure that your menu will use. The menu you will build in this practice uses the structure shown in Figure 19-4.

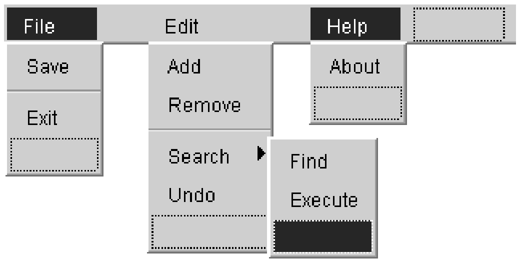


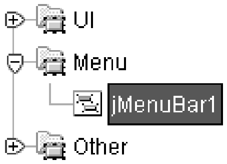
FIGURE 19-4. *Sample menu structure*

NOTE
This practice is written as a demonstration of the menu editing facilities in JDeveloper. It assumes that you are creating a menu system from scratch. Since the items on this menu are the same as those created by the JClient Form Wizard, you could also just rearrange the generated items.

I. Lay Out the Menu Elements

The first phase defines the structure of the menu and adds menu elements to the JFrame (window) class, DepartmentsForm.java. At this point, you do not need to worry about all of the properties or names of the elements. Use the following steps to lay out a menu bar:

1. Open the UI Editor for DepartmentsForm.java if it is not already open. Click the Menu node in the Structure window. The UI Editor will change to a Menu Editor blank window.
2. In the Swing Containers Component Palette page, click JMenuBar, and click the Menu node in the Structure window to add the menu to the application. The Structure window will display the new menu bar as shown here:

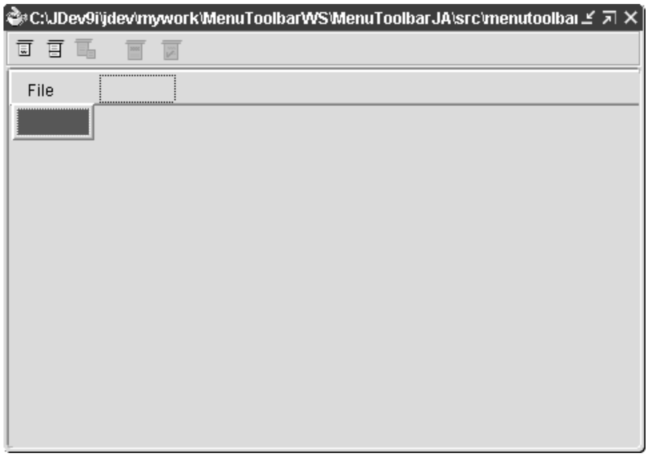


3. Change the *name* property to "mainMenuBar" in the Property Inspector.

Additional Information: At this point, you will see nothing in the UI Editor other than a dotted rectangle that you will use to enter a menu item.

NOTE
You can change between the UI Editor and Menu Editor by clicking the appropriate node in the Structure window (UI and Menu, respectively).

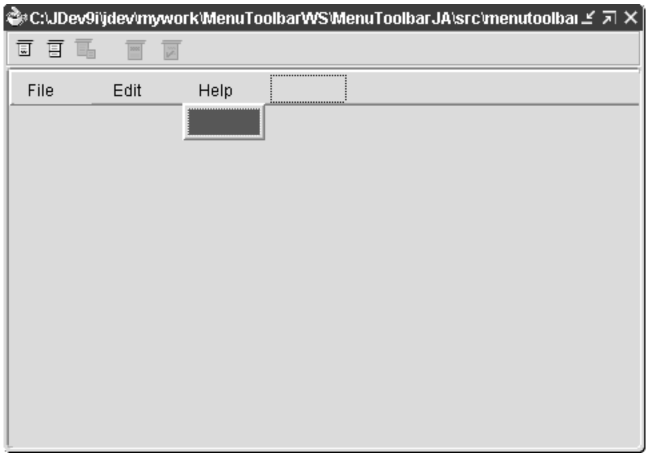
4. To add the first menu item, click the dotted rectangle in the upper-left corner of the Menu Editor and enter "File." Press ENTER after typing the text. The text is written into the *text* property of the menu element. The Menu Editor window will look like this:


C:\JDev9i\jdev\mywork\MenuToolbarWS\MenuToolbarJA\src\menutoolbar
File

Additional Information: When you add a menu, the Menu Editor opens a new menu to the right and a new item below. Each time you add a menu item, the Menu Editor opens a blank item below it. New menu elements are shown with a dotted or selected box. Clicking a new element allows you to type text.

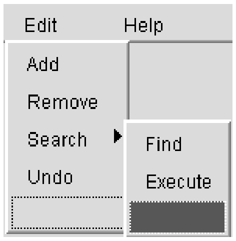
CAUTION
*If new (dotted rectangle) items do not appear when you enter the "File" label, try the operation using the Windows look-and-feel (**Tools | Preferences**, Environment page, set the "Look and Feel" field to "Windows," click OK, and reload JDeveloper).*

5. Click the box to the right of the File menu, and type "Edit" for the text. Press ENTER. This adds another menu (top-level) element. Add another menu element for "Help." Your menu will look like this:


C:\JDev9i\jdev\mywork\MenuToolbarWS\MenuToolbarJA\src\menutoolbar
File
Edit
Help

6. Click the File menu and click the box under it. Type the label as "Save" and press ENTER. Click the box under Save and type "Exit" and press ENTER.
7. Select "Exit" and click Insert Separator in the Menu Editor toolbar. This adds a separator above the Exit item.
8. Add items under the Edit and Help menus to match the structure shown in Figure 19-4.

   Additional Information: For the submenu on Search, select "Search" and click Insert Submenu in the Menu Editor toolbar. This converts Search to a submenu and opens an edit box to the right. Enter "Find" for the menu item and "Execute" for another menu item in this submenu. The resulting Edit menu is shown to the right.



9. Click Save All.

NOTE
You can also select menu editing actions such as Insert Submenu from the right-click menu.

What Just Happened? You just added a menu bar component to the application and used the Menu Editor to define the menus and menu items that will be displayed. Since the menu bar attaches to the window title bar, you do not have to worry about the layout in the frame.

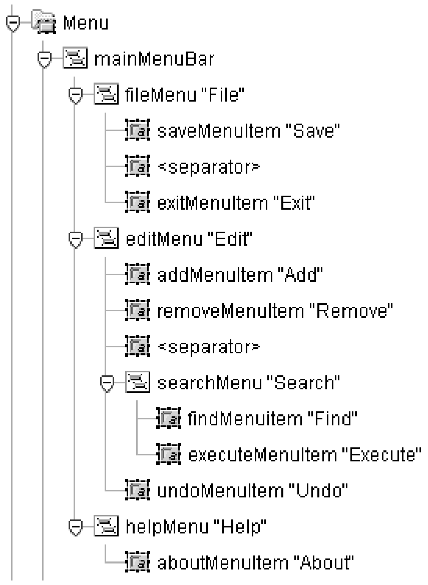
NOTE

If your design included check menu items, you would use the Make Checkable toolbar button to convert the item to a check menu item.

II. Set the Menu Element Properties

Now that the menu structure is laid out, you need to modify the properties. The primary property that you need to change is the *name* property. Other properties to modify are *mnemonic* and *accelerator*. Changing properties requires visiting the Property Inspector for each item. Use the following steps to accomplish this:

1. Click the JMenu1 (top) icon under mainMenuBar in the Structure window. Change the *name* property to "fileMenu" in the Property Inspector and press ENTER.
2. Change the name of the first menu item under fileMenu to "saveMenuItem."
3. Repeat steps 1 and 2 for all menus and items using the same pattern. You do not need to rename the separators. The Structure window should appear as follows:



```
Menu
  mainMenuBar
    fileMenu "File"
      saveMenuItem "Save"
      <separator>
      exitMenuItem "Exit"
    editMenu "Edit"
      addMenuItem "Add"
      removeMenuItem "Remove"
      <separator>
      searchMenu "Search"
        findMenuitem "Find"
        executeMenuItem "Execute"
      undoMenuItem "Undo"
    helpMenu "Help"
      aboutMenuItem "About"
```

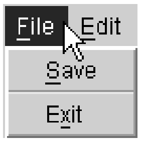
4. Add mnemonics for each menu item by assigning the *mnemonic* property using the following table:

Menu or Menu Item	***mnemonic* Property**
File	F
Save	S
Exit	X

Menu or Menu Item	***mnemonic* Property**
Edit	E
Add	A
Remove	R
Search	S
Find	F
Execute	X
Undo	U
Help	H
About	A

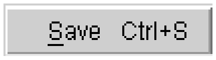
Additional Information: A mnemonic is a letter or character that users can press to activate the menu item. The rule is that you need to select a mnemonic letter that is part of the string in the *text* property. Letters may be repeated as above if the items that they represent are in different menus.

For example, you can set the File menu to pull down when the user presses ALT-F by assigning the *mnemonic* property as "F." This assignment will place an underscore on the matching letter during runtime as the following shows:



4. Click the Save menu item. Assign the *accelerator* property in the Property Inspector by selecting "KeyStroke" from the pulldown. This opens a dialog where you select the modifier and key for the accelerator. In this case, select "CTRL_MASK" (the CTRL key) as the modifier and "VK_S" (the "s" key) as the key. Click OK to close the dialog. Click Save All.

   Additional Information: Recall that accelerators are shortcut keys that allow the user to access the function of a menu item without activating the menu. In this case, the key combination of CTRL-S will activate the function for the Save menu item. Although you will not see the shortcut key indicated in the editor, when you run the file the shortcut key appears next to the menu item during runtime, as the following shows:



What Just Happened? You changed the names of the menus and menu items to comply with a naming convention (mentioned in Chapter 5) that requires a suffix to denote the type of each object. You also added mnemonics and an accelerator to assist users in accessing the menus and menu item functions.

III. Write the Menu Item Code

You do not need to write code to display menu elements because this function is performed by the control. However, you do have to write the event code that each menu item executes when it is selected.

Attaching code to a component is a three-step process:

1. Write a listener for the object that will wait for an event to occur on the object.
2. Identify the type of event the user will activate (such as mouse click, keypress, and focus-gained), and write a method for that event inside the listener definition. A generic event for menu items and button clicks is the *actionPerformed* event; this event occurs when the default action for the item occurs (for example, a button click or menu item selection). The event method calls another method defined in the class file.
3. Write the method that is called by the event method.

   For example, the Save menu item event will use a listener defined by the following code:

   ```
   saveMenuItem.addActionListener(new ActionListener()
   {
       public void actionPerformed(ActionEvent e) {
         saveMenuItem_actionPerformed(e);
       }
     }
   );
   ```

   This code creates a listener for the Save menu item that calls the `saveMenuItem_actionPerformed()` method when the *actionPerformed* event occurs; in this case, the action performed event corresponds to the user selecting the Save menu item. The `saveMenuItem_actionPerformed()` method contains the code that executes the commit action on the navigation bar as follows:

   ```
   private void saveMenuItem_actionPerformed(ActionEvent e)
   {
     // calls an action on the navigation bar
     hiddenNavBar.doAction(JUNavigationBar.BUTTON_COMMIT);
   }
   ```

Event Code

This section adds code to handle the events that occur when a menu item is selected. JDeveloper assists in creating the listener, its event method, and the method that the event method calls, as the following steps demonstrate:

1. Invoke the UI Editor to display the menu for the DepartmentsForm.java file if it is not already displayed.

TIP
At any point, if the UI Editor and Structure window are out of sync, click in the UI Editor, and then click in the Structure window. When you click the UI Editor again, the Structure window should refresh.

2. Click the icon for saveMenuItem in the Structure window. Click the Events tab in the Inspector.

   Additional Information: In the Events tab are event names (in the properties column) and the associated called method names (in the property values column). The method names do not appear until you define the event in the Property Inspector. You can specify the method name that you want the event to call in your code by filling out the value column for an event. The default name is a concatenation of the object name and the event name. For example, the saveMenuItem event for actionPerformed defaults to "saveMenuItem_actionPerformed."

3. Click the *actionPerformed* event and click the "…" button to display the actionPerformed dialog.

4. Leave the default name and click OK. The Code Editor will open to a method stub for `saveMenuitem_actionPerformed()` that was just added.

   Additional Information: JDeveloper also creates code to define the listener. Take a moment to search from the top of the file for "saveMenuItem_actionPerformed." The first occurrence of this string is in a section of code that sets properties for the Save item.

NOTE
If you change the name you assign to an existing event in the Property Inspector, the code for the method and listener will be modified automatically.

5. In the `saveMenuItem_actionPerformed()` method stub at the bottom of the file, add a line between the { } and enter the following code (all on one line):

   ```
   hiddenNavBar.doAction(JUNavigationBar.BUTTON_COMMIT);
   ```

   Additional Information: The JClient Form Wizard added a navigation bar object into the frame file (DepartmentsForm.java). This navigation bar (`hiddenNavBar`) is bound to the DepartmentsView view object and provides data manipulation functions such as `INSERT`, `DELETE`, `COMMIT`, and `ROLLBACK`. You can take advantage of these functions by calling a method, `doAction()`, that performs the same action as a specific button on the navigation bar (in this case the commit button). Calling pre-built data aware code is much easier than writing code from scratch code.

6. Return to the Menu Editor by clicking the DepartmentsForm.java UI Editor tab in the toolbar area.

7. Repeat steps 2–6 for the other menu items using the actionPerformed method code listed here:

Item	Method Code
exitMenuItem	`_popupTransactionDialog();` `System.exit(0);`
addMenuItem	`hiddenNavBar.doAction(` `JUNavigationBar.BUTTON_INSERT);`
removeMenuItem	`hiddenNavBar.doAction(` `JUNavigationBar.BUTTON_DELETE);`
findMenuItem	`hiddenNavBar.doAction(` `JUNavigationBar.BUTTON_FIND);`
executeMenuItem	`hiddenNavBar.doAction(` `JUNavigationBar.BUTTON_EXECUTE);`
undoMenuItem	`hiddenNavBar.doAction(` `JUNavigationBar.BUTTON_ROLLBACK);`
aboutMenuItem	`JOptionPane.showMessageDialog(this,` `"Department browse and edit window",` `"Department Browse",` `JOptionPane.INFORMATION_MESSAGE);`

TIP
You can bypass the event dialog by entering the event method name in the Property Inspector's event field instead of clicking the "..." button.

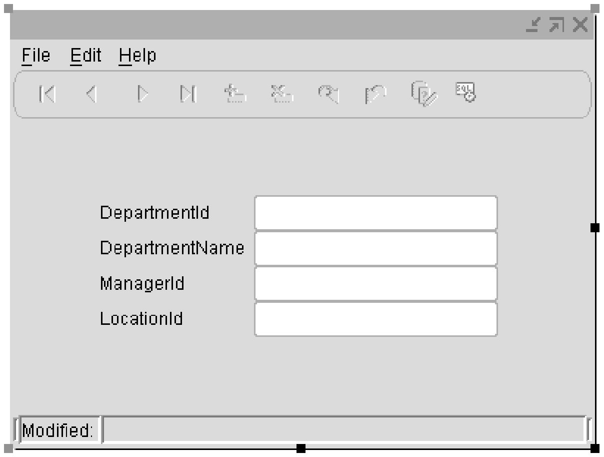
Additional Information: The About menu item calls JOptionPane to display a modal dialog. The Exit item calls the transaction dialog created by the JClient Form Wizard that asks the user if uncommitted changes should be saved. If there are no uncommitted changes, the dialog does not appear. The other menu items call button actions on the navigation bar.

8. Click Rebuild and Save All in the toolbar.
9. Run the DepartmentsForm file and test all menu items.

TIP
In addition to the Help About dialog, you might want to build an entire help system for your application. JDeveloper includes support for creating your own help files. Details about the Oracle Help for Java feature are contained in the help system Contents node "Developing Help With Oracle Help for Java."

What Just Happened? You wrote event-handler code for each menu item in the layout. Most of the items call code that emulates a button click on a hidden navigation bar.

The menu bar appears at the top of the frame in the UI Editor as shown here:



If you are interested in adding code that enables and disables menu items, generate an application using the JClient Form Wizard and specify that the wizard create a menu. The code in the `_updateButtonStates()` and `menuItemsUpdate()` methods that are generated will give you an idea about how to implement enabling and disabling menu options.

Hands-on Practice: Build a Popup Menu

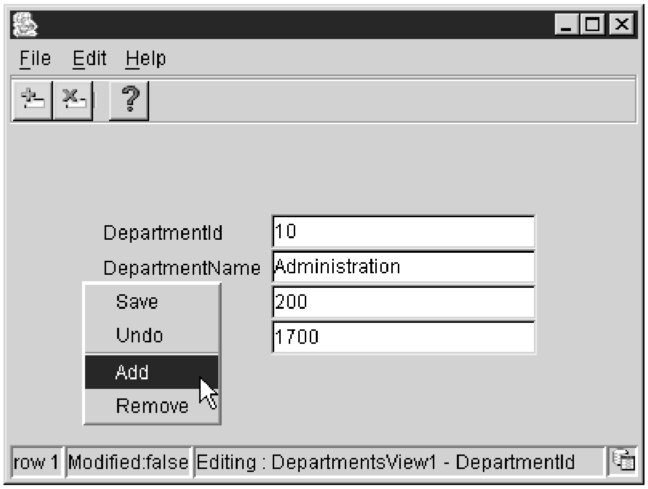
The method for creating a popup menu (shown in Figure 19-5) is similar to the method for creating a regular menu. It uses the sample application from the preceding practice as a starting point. As before, you can also use the application available in the sample files on the authors' websites.

Although completing the previous practice is not a requirement, this practice abbreviates many steps that were explained in that practice. If you need further explanation for a particular step in this practice, refer to the preceding practice. There are two main phases in creating a popup menu:

I. Lay out the elements

II. Write the menu code

- Display the popup menu
- Handle an event for each menu item



10
Administration
200
1700

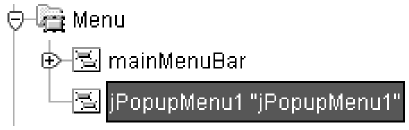
row 1 | Modified:false | Editing : DepartmentsView1 - DepartmentId

FIGURE 19-5. *Popup menu*

I. Lay Out the Elements

The first phase uses the UI Editor to place the objects on the frame. Be sure that the workspace for the previous practice or the sample application created in the "Hands-on Practice: Prepare a Sample Application" section is open.

1. Open the UI Editor for DepartmentsForm.java if it is not already open.
2. Drop a JPopupMenu component from the Swing Containers page of the Component Palette onto the Menu node of the Structure window. The Structure window will display the following:



3. Rename the popup menu to "mainPopupMenu" using the *name* property in the Property Inspector. Change the *label* property to "mainPopupMenu."

4. Add four items to the popup menu using the same techniques you used in the previous practice, and change the *name* properties as in the following table:

Item	***name***
Save	savePopupItem
Undo	undoPopupItem
Add	addPopupItem
Remove	removePopupItem

5. Select the Add item and add a separator using the Insert Separator toolbar button.

   Additional Information: Remember that separators and menu items will be added above the currently selected item. You can drag and drop objects to rearrange them in the Menu Editor.

TIP
When adding items to a menu, pressing ENTER *after entering the name will add an item below and move focus to the new item. All you need to do is start typing the new text at that point.*

6. Click Save All.

What Just Happened? You added the popup menu to the application frame. It appears in the Menu node of the Structure window but, unlike menu bars, will not be displayed until you write code to perform that action. In this phase, you also used the Menu Editor to lay out the menu items in a single menu. Unlike a menu bar, a popup menu does not have top level menus that display a pulldown with items. The remaining task is to write some code.

II. Write the Menu Code

You need to write event-handling code for displaying the popup and for performing a function when a menu item is selected. As with the main menu, JDeveloper helps you create the code for both purposes.

Display the Popup Menu

While a pulldown menu is usually displayed at the top of the window, a popup menu usually appears when the user clicks the right mouse button. (You can display a popup menu on a button click or other event as well.) Therefore, in this example, you need an event handler to display the popup menu on a mouse click. This handler consists of a listener on a panel in the application that calls a custom method. The following steps create this code:

1. Open the UI Editor for the DepartmentsForm.java file if it is not already open.

2. Expand the UI node in the Structure window and the "this" node under it. Click the first panel in the list (for example, topPanel).

3. Click the Events tab in the Property Inspector and click the "…" button on the *mouseReleased* event to display the mouseReleased dialog.

4. Notice that the event name is "topPanel_mouseReleased." This name is fine for this example, so click OK. The Code Editor will open, and a blank method stub will appear for the name you just entered.

5. Add a blank line between the curly brackets and enter the following in the blank line:

   ```
   if (e.isPopupTrigger()) {
     mainPopupMenu.show(this, e.getX(), e.getY());
   }
   ```

   Additional Information: This code shows the popup object inside the main frame object ("this"). You can programmatically attach the popup menu to other objects in the same way. The code also shows the popup menu at the mouse cursor's location (x and y positions) when the right mouse button was clicked.

   JDeveloper also adds a listener, `topPanel.addMouseListener(new MouseAdapter()`, that calls this new method. You do not need to modify the listener code.

6. Click Save All. You may want to run the DepartmentsForm file to check the popup menu display. Right-clicking anywhere in the main window will display the popup menu. The menu items will not yet be functional because you have not written the code.

Handle an Event for Each Menu Item

The code to handle events for each popup menu item works the same way as it does with a pulldown menu. The code you create for a popup menu item consists of a new listener that calls a wizard-generated method used by the main menu. This section uses the same methods created for the menu bar in the preceding practice. Most of those methods call functionality on the hidden navigation bar generated by the JClient Form Wizard. If you want to use functionality that is not on the navigation bar, you can write your own code instead of calling existing methods.

1. In the UI Editor for DepartmentsForm.java, click savePopupItem in the Structure window. In the Events tab of the Property Inspector, enter "saveMenuItem_actionPerformed" for the *actionPerformed* event and press ENTER.

   Additional Information: The focus will shift to the Code Editor and navigate to the existing method. If you did not complete the preceding practice, the method will be blank. If this is the case, return to the previous practice and fill in the code listed for the saveMenuItem. Notice that you did not need to display or interact with the event dialog because you typed an event name in the Property Inspector. The ENTER key completes the entry process and causes the Code Editor to be displayed. JDeveloper also creates a listener for the popup menu item to call the event handler method.

2. Repeat step 1 for each item in the following list:

Item	**Event Method**
undoPopupItem	undoMenuItem_actionPerformed
addPopupItem	addMenuItem_actionPerformed
removePopupItem	removeMenuItem_actionPerformed

3. Click Save All.
4. Compile the project and run it. Try all functions using the popup menu.

What Just Happened? You wrote the code for the main event handler that pops up the menu when the right mouse button is clicked. You also wrote code to execute the appropriate function from the navigation bar as in the previous practice. If you have new items that do not use existing code, the steps are the same except that you have to write the code into the code stub that the Property Inspector creates.

Hands-on Practice: Build a Toolbar

A toolbar is usually created as an extension of a menu, and it contains the most commonly used menu items. This practice demonstrates how to develop a toolbar that calls some of the same methods called by the menu. Although the toolbar only contains three buttons, you can use the same techniques to add more buttons. This practice will show how you can build a toolbar from scratch should you find that the navigation bar does not meet your needs. For example, you may want to add buttons of your own or to replace the navigation bar's native query-by-example find utility with one of your own.

This practice uses the sample data form application that is built in the earlier "Hands-on Practice: Prepare a Sample Application" section. You may also use the application that resulted from either of the preceding practices in this chapter, but it is not a requirement that you complete those practices.

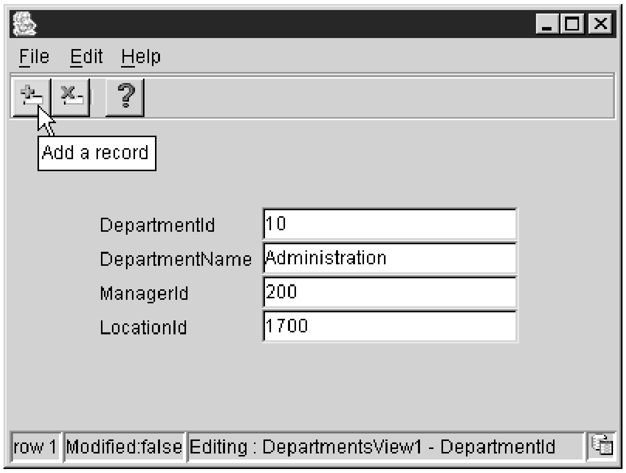
This practice uses phases that are similar to those in the menu practice as follows:

I. **Lay out the toolbar elements**
   - Add the toolbar
   - Import the image files
   - Create image icon objects
   - Add the buttons

II. **Set the button properties**

III. **Write the button code**

At the end of this practice, you will have created a toolbar that looks like the one shown here under the menu:



DepartmentId	10
DepartmentName	Administration
ManagerId	200
LocationId	1700

row 1 | Modified:false | Editing : DepartmentsView1 - DepartmentId

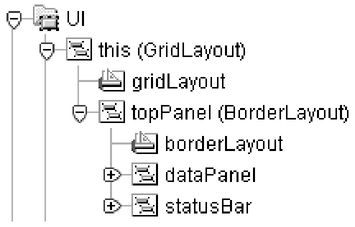
I. Lay Out the Toolbar Elements

As with the menu, the toolbar will be contained in the form file built from JFrame (the window). In this phase, you will drop a toolbar onto a panel that is assigned a BorderLayout manager. This layout allows you to fix the toolbar to the top of the parent container. (Chapter 20 describes layout managers in detail.)

Add the Toolbar

The first step is to add the toolbar container and buttons to the frame.

1. Open the UI Editor for the DepartmentsForm.java file if it is not already open.
2. Expand the UI node in the Structure window and the "this" node under it. The Structure window will look something like the following:



```
UI
  this (GridLayout)
    gridLayout
    topPanel (BorderLayout)
      borderLayout
      dataPanel
      statusBar
```

3. Drop a JToolbar component from the Swing Containers page of the Component Palette on top of the panel (in this example, topPanel) under the "this" node in the Structure window.

4. Change the following properties for the toolbar:
   name to "mainToolBar"
   constraints to "North"

   Additional Information: Although you will see the object in the Structure window, it will not be prominent in the editor. The toolbar will grow when you drop buttons into it later.

5. Be sure that the *constraints* property of dataPanel (under topPanel in the Structure window) is set to "Center" and the statusBar *constraints* property is set to "South." Change these if needed.

Import the Image Files

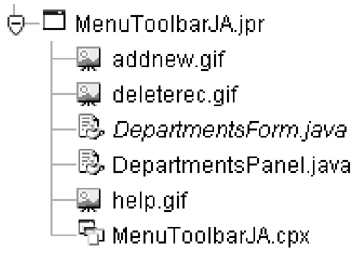
You have to load the icon files that you will be using for the toolbar images into the project. This practice uses .gif files already available in the JDeveloper directories but you can use the same techniques to import your own image files, if needed.

1. Click the project node and select **File | Import**.
2. Select Existing Sources and click OK to display the Import Existing Sources Wizard. Click Next if the Welcome page appears.
3. Click Add and navigate to the JDEV_HOME\BC4J\redist\ bc4j\webapp\images directory.

   Additional Information: If the specified directory does not exist, open the images directory under the webapp.war directory (actually an archive file that looks like a directory in the JDeveloper file dialog) in the JDEV_HOME\BC4J\redist directory.

NOTE
If the files or directories mentioned do not exist in your installation, search for similar files in the JDeveloper directories using Windows Explorer.

4. Select the files addnew.gif, deleterec.gif, and help.gif (using CTRL click to group select them).
5. Click Open to add the files to the *Refine Files to Be Added* pane.
6. Check the checkbox Copy Files to the Project Directory and click Browse. Navigate to the menutoolbar package directory under src and click Select. The directory, JDEV_HOME\jdev\mywork\MenuToolbarWS\MenuToolbarJA\src\menutoolbar, will appear in the text field.
7. Click Next and Finish. The files will be copied to the menutoolbar directory and will be displayed in the Navigator as shown here:


MenuToolbarJA.jpr
addnew.gif
deleterec.gif
DepartmentsForm.java
DepartmentsPanel.java
help.gif
MenuToolbarJA.cpx

Additional Information: When the project is built, the .gif files will be copied to the classes\menutoolbar directory so that they will be available to the compiled Java files at runtime.

NOTE
The icon files will be packaged with your other code files for distribution, and they will be easier to find if they are contained in the same directory. In addition, if the files are in the project directory, the code that the Property Inspector creates when you set the icon file properties will not include the path, which will make the code more portable.

8. Select one of the graphics files in the Navigator and check that the file name displayed in the JDeveloper IDE status bar includes the src\menutoolbar directory.
9. Click Save All.

Create Image Icon Objects

You will use the Swing JButton class to create buttons in this form. The JButton class contains an *icon* property that allows you to create an iconic button by defining an image file as the icon. The `setIcon()` method for a button object requires the parameter of an *image icon*, an object that is linked to an image file. Therefore, you need to programmatically create image icon objects for the graphics files to make the icons available to the button properties. This section adds the appropriate code to create the image icon objects.

1. Open the Code Editor for the DepartmentsForm.java file and find the following line that defines the class:

```
public class DepartmentsForm extends JFrame  {
```

2. There are a number of object declarations under this line. Add the following declarations directly under the class declaration line (after the opening curly bracket):

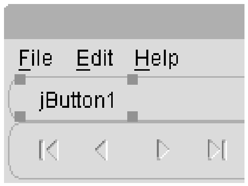
```
// button icons
private ImageIcon imageAdd = new ImageIcon(
  DepartmentsForm.class.getResource("addnew.gif"));
private ImageIcon imageDelete = new ImageIcon(
  DepartmentsForm.class.getResource("deleterec.gif"));
private ImageIcon imageHelp = new ImageIcon(
  DepartmentsForm.class.getResource("help.gif"));
```

Additional Information: This code creates image icon objects that can be assigned to the *icon* properties of the buttons.

Add the Buttons

You are now ready to add the buttons.

1. Return to the UI Editor window by clicking the appropriate file name in the Document Bar. Expand the UI node until you see mainToolbar. Shift click the JButton component in the Swing page of the Component Palette. The button will *pin* (appear outlined) so that you can draw more than one button without reselecting the component.
2. Click the mainToolBar node in the Structure window. A button will be added as shown in this excerpt from the UI Editor:



3. Click the mainToolBar node twice more to create two more buttons.
4. Click the Pointer (arrow) icon in the Component Palette to unpin the button tool.

What Just Happened? You added a toolbar component to the layout and placed buttons in it. You also prepared the project for the assignment of icon images by adding image files to the project and creating image icon objects in the code.

II. Set the Button Properties

You now need to set the button properties to refine the definitions.

1. Select the buttons as a group by clicking the top button in the Structure window and CTRL clicking the other button.
2. When the buttons are selected, apply the following properties' values to the group. Click ENTER after setting each property.
   maximumSize to "25,25"
   minimumSize to "25,25"
   preferredSize to "25,25"
   text to blank (no value)
3. Click the top button in the list in the Structure window to ungroup the buttons. Change the following properties:
   name to "addButton"
   icon to "imageAdd"
   toolTipText to "Add a record"

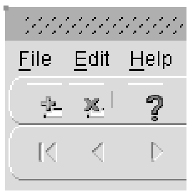
NOTE
The button image may not show correctly in the UI Editor. This will be corrected when you rebuild or run the project.

4. Repeat step 3 to set the properties for the second button as follows:
   name to "deleteButton"
   icon to "imageDelete"
   toolTipText to "Delete this record"
5. Repeat step 3 to set the properties for the third button as follows:
   name to "helpButton"
   icon to "imageHelp"
   toolTipText to "Help about"

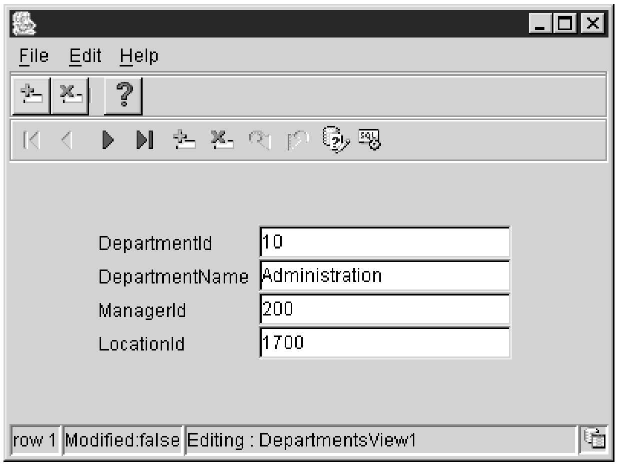
CAUTION
If you have a problem assigning the icon property, double check the location of the image files and the image icon object code as described in the earlier sections.

Additional Information: You can use a number of different properties to assign various icons to a button for different purposes. The *icon, pressedIcon, disabledIcon, disabledSelectedIcon, rolloverIcon,* and *rolloverSelectedIcon* properties define which icon appears on the button in different situations. For example, if you define a different icon for the rolloverIcon property, the icon will change if the user holds the mouse cursor over the button. For brevity, this practice only assigns an icon file to the *icon* property.

6. Click the JSeparator icon in the Swing page of the Component Palette, and drop it between the second and third buttons in the UI Editor. The right-hand button will move to the right. The separator serves the same purpose in the toolbar as it does in the menu— to create logical groupings of functions.
7. Click the separator and set the following properties:
   maximumSize to "10,25"
   orientation to "VERTICAL"
8. Click Rebuild and Save All. The image files will be copied into the classes\menutoolbar directory and the UI Editor will now display the correct icons as shown in this excerpt from the UI Editor window:



9. Click Save All and Run. The form will run and look something like the following. (The buttons you added will not yet be functional.)



DepartmentId 10
DepartmentName Administration
ManagerId 200
LocationId 1700

row 1 | Modified:false | Editing : DepartmentsView1

10. Notice that there are now two toolbars. The navigation bar in the DepartmentsPanel.java file and the toolbar you just created. Close the running form file. Open DepartmentsPanel in the Code Editor and find the code that adds the navBar object to the panel (`add(navBar, BorderLayout.NORTH);`). Comment this line of code so that it appears as follows:

```
// add(navBar, BorderLayout.NORTH);
```

11. Run the form file to confirm that the navigation bar is not displayed.

What Just Happened? This phase set the button properties for size and icon. You used the object grouping feature to apply common property values to a group of objects. You also used the button pinning feature to drop multiple copies of the same component. In addition, you added a separator to define a logical grouping of buttons.

The sidebar "Moving Objects Around" provides some techniques for repositioning objects.

Moving Objects Around

At this point, it is a good idea to take a few minutes to practice moving buttons around. The techniques you try here on toolbar buttons can be used with any visual object. Start this practice with the UI Editor. You can drag and drop buttons or the separator within the toolbar to reorder them. You can also drag them outside the toolbar, and they will reposition themselves in the Structure window nodes. Try this out, but be sure you restore the layout.

Another method for reordering objects is to cut and paste them in the Structure window. Whenever you paste an object on another object such as a pane, the pasted object will take a position either on the top or on the bottom of other objects inside that pane. Try cutting the delete button by selecting it in the Structure window and pressing CTRL-X. Paste it on top of the toolbar node by clicking that node and pressing CTRL-C. Try this on other buttons, but be sure to restore the layout when you are done.

III. Write the Button Code

The last step is to add the code to handle the button events. You use the same steps here as in the menu practices. The code calls the same methods as the menu items call.

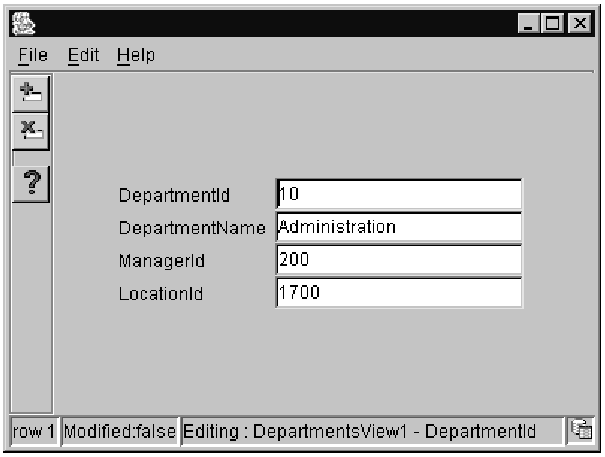
1. Open the DepartmentsForm.java file in the UI Editor if it is not already open. Select the addButton object. In the Property Inspector, click the Events tab and enter "addMenuItem_actionPerformed" in the *actionPerformed* event on the Events tab.
2. Press ENTER, and the focus will shift to the existing code in the Code Editor.

   Additional Information: If you did not complete the earlier menu practice, this method will have no code. Fill out the method body with the code contained in the menu practice. JDeveloper also creates the listener code for the toolbar button that calls this method.
3. Repeat steps 1 and 2 for the deleteButton and use the event name "removeMenuItem_actionPerformed".
4. Repeat steps 1 and 2 for the helpButton and use the event name "aboutMenuItem_actionPerformed".
5. Click Save All. Run the application.
6. Click the Help About button to test the code. Hold the mouse cursor above each button to see the tool tips. Try the add and remove buttons. The sidebar "Vertical and Floating Toolbars" contains details about toolbar features.
7. Exit the application.

What Just Happened? You added and tested the code that will be executed when the buttons are clicked. This code is similar to the code that you wrote for the menu practice. Once you have the code for the menu, creating the toolbar is just a matter of layout and some property settings.

Vertical and Floating Toolbars

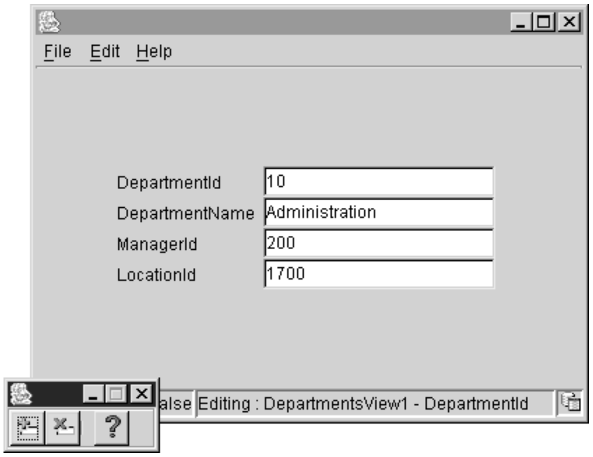
At runtime, try clicking a nonbutton part of the toolbar and dragging the toolbar to the right or left side of the window. The toolbar outline will change to a vertical orientation, and when you release the mouse button, the toolbar will stick to the side of the window, as the following shows:



DepartmentId 10
DepartmentName Administration
ManagerId 200
LocationId 1700

row 1 | Modified:false | Editing : DepartmentsView1 - DepartmentId

You can resize the outer window if you need to provide more room for the toolbar. You can also drag the toolbar out of the window, as in the following:



DepartmentId 10
DepartmentName Administration
ManagerId 200
LocationId 1700

alse | Editing : DepartmentsView1 - DepartmentId

This creates a separate floating toolbar window that acts in the same way as the windows in JDeveloper's Oracle Business Component Browser. Clicking the window close icon in the upper-right corner of the floating toolbar anchors it again inside the window. The toolbar component provides this functionality. You can disable floating by setting the *floatable* property of the toolbar to "False."