Powersoft PowerJ Learning Edition
Quick Start

VERSION 98.04.22

Powersoft®

Contents

About PowerJ Learning Edition

PowerJ Learning Edition provides JavaBean-based visual development and deployment for the Java programming language. Its drag-and-drop programming, in-context debugging, and syntax-highlighting tools save time and make Java easier to learn.

The PowerJ Learning Edition environment includes the following features:

- The Sun Java Development Kit (JDK) 1.1.5
- Four available targets (Applet, Application, Classes, and Beans) that you can create
- Online documentation (this *Quick Start*, the *PowerJ Programmer's Guide*, component references for Sun JDK 1.1.5, *PowerJ and Power++ Keyboard Shortcuts* references, and *Thinking in Java*)
- PowerJ Samples

PowerJ Learning Edition provides support for: Sun JDK 1.1.5; importing existing Java code into PowerJ; the Sun Virtual Machine (VM); running programs in web browsers; and debugging programs in the Applet Viewer and under Internet Explorer.

About this guide

This guide describes the PowerJ development environment for the Java programming language. The guide assumes that you are familiar with the basic principles of using Windows, including how to:

- Start a Windows program.
- Reposition, resize, and close windows.
- Create, open, copy, and delete files and folders.
- Point, click, double-click, and drag with a mouse or other pointing device.

If you are not familiar with such features, consult the Windows documentation (for example, *Introducing Microsoft Windows 95*).

This guide also assumes that you are familiar with, or in the process of learning, the Java programming language. To help you to learn Java, PowerJ Learning Edition includes *Thinking in Java*, an introduction to Java by Bruce Eckel.

There is a PDF version of this Quick Start, available in the PowerJ 2.1 Learning Edition folder, for printing.

A quick start

About this chapter

This chapter includes introductory tutorials for PowerJ Learning Edition, a list of product features with helpful references to the *PowerJ Programmer's Guide*, and a list of sample programs.

The introductory tutorials will guide you through the creation of simple Java programs using PowerJ, a tool for writing Java applications and applets. The tutorials demonstrate, respectively, how to create a command line (console) Java application and how to create a Java applet with PowerJ.

Contents

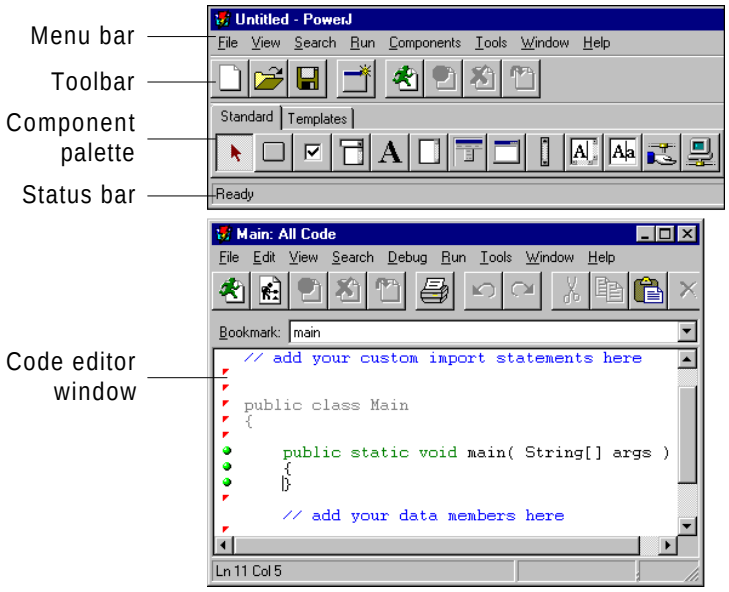
Tutorial 1: Creating a command line application

This tutorial shows you how to write a simple, command line, Java application called ‘‘Hello World’’ with the help of PowerJ.

♦ **Start PowerJ:**

1. Double-click on the **PowerJ icon** in the `PowerJ Learning Edition` folder.

When you first open PowerJ, you will be presented with a new, untitled *code editor window*:



By default, the code editor window contains the startup code for a Java application. To create a command line application, you simply enter the remainder of the necessary Java code in the code editor and run the application in a Java console.

♦ **Create the Hello World application:**

1. After the opening brace of the main method, start a new line and type the following Java code:

```
System.out.println("Hello World");
```

The final code should look like this:

```
public class Main
{
    public static void main( String[] args )
    {
        System.out.println("Hello World");
    }
}
```

Now you can use PowerJ to save your code. In PowerJ, code is saved as a *project*. Every project has an associated project file which lists information about the project. PowerJ project files use the extension `.WXJ`.

A single PowerJ *target* (a Java applet, standalone Java application, or Java package) has many associated source files, which are automatically generated by PowerJ. (Source files can be Java source code files, HTML files, and other files.) You must save each target in a separate folder to prevent the files of one target from overwriting the files of another.

♦ **Save your Hello World application:**

1. On the **File** menu of the main PowerJ menu bar, click **Save Project**. This displays the contents of the `Projects` folder in your `PowerJ21le` folder.
2. Under **Folder name**, type `Hello`. Click **Save**. This creates a folder called `Hello` for the target.
3. Next to **File name**, leave `Hello.wxj` as the project file name, and click **Save**. PowerJ creates a project file named `Hello.wxj` in the folder named `Hello`, along with other target files.

Your application is complete and can now be run on a Java console.

♦ **Run your Hello World application:**

1. On the **Tools** menu, click **Show Console** so that it is checked.

   The Java console appears. Position it on the screen so that you can see its upper-left corner.
2. On the **Run** menu of the main PowerJ window, click **Run**.

   PowerJ runs the `Hello` target. The text `Hello World` appears on the console.

Note: Clicking the Run menu item (or the Run button) in PowerJ is equivalent to compiling your Java code with a command line compiler (e.g. javac) and then executing the application (e.g. by typing `java classname`).

When you no longer want to view the Java console, go to the **Tools** menu and click **Show Console** again.

- **Close the Hello project:**

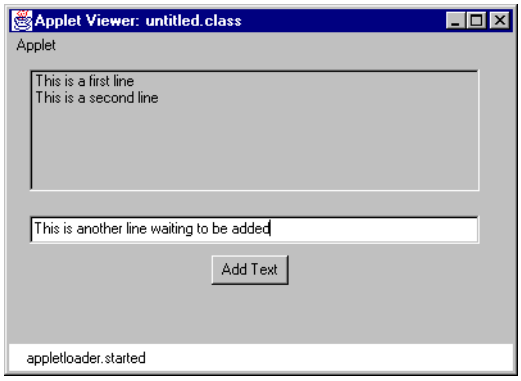
  1. On the **File** menu of the main PowerJ window, click **Close Project**.

Tutorial 2: Creating an applet

To design the user interface of an applet using the PowerJ *framework*, you lay out one or more objects on the *form design window*, creating a user interface that can appear on a web page or on an *Applet Viewer*. Together, the form design window, component palette objects, and project views allow you to manage forms and use the drag-and-drop programming features.

Important: When you use the PowerJ framework to create an applet or application, you will need to access classes in the `powerjle.zip` file. When you deploy your target, make sure that `powerjle.zip` is in the same directory as the target's class file. For more information on classes and deployment, see the following sections of the *PowerJ Programmer's Guide*: *How PowerJ uses the CLASSPATH environment variable* in Chapter 2, *Classpath options* in Chapter 3, and *CLASSPATH and CODEBASE* in Chapter 5.

The applet you are about to create has a user interface that consists of a list, a text field, and a command button. When you run the applet in the Applet Viewer, you will see a form similar to the following:



When the program is running, you can type a line of text into the text field, then add that line to the list by pushing the command button.

- **Start a new project:**

  1. On the **File** menu of the main PowerJ menu bar, click **New Project**.

     The code editor window for an untitled new project appears. To create a framework applet, you also need to use the PowerJ form design window.

  2. On the **File** menu of the main PowerJ menu bar, point to **New** and click **Target**. This opens the Target Wizard, which takes you step-by-step through the creation of the new target.

  3. Click **Java - Applet**, then click **Next**.

     The Target Wizard asks you if you wish to use the PowerJ framework. Using the PowerJ framework allows PowerJ to automatically generate code that implements the Applet's main form. It also lets you use PowerJ's design-time facilities to make the user interface.

  4. **Use the PowerJ framework** is selected by default. Click **Next**.

  5. Click **Next** to use the **Powersoft Java AWT 1.10** class library.

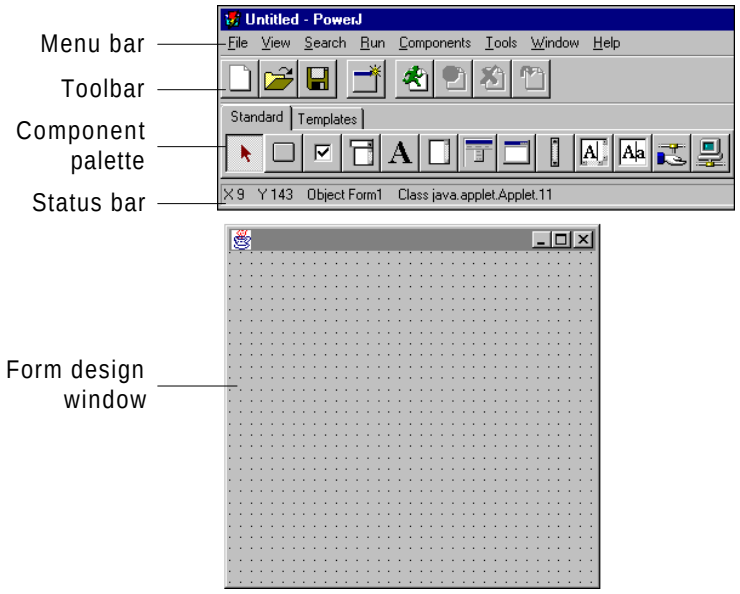
  6. Under **Target Name**, type `List`, and specify where to store the files associated with the new target. Typically, you will accept the default to create a new folder under the PowerJ Learning Edition Projects folder, using the name you just typed for the new target. Click **Next**.

     The Target Wizard asks you what you want to call the new form.

  7. Click **Finish** to accept the default form name.

     PowerJ creates the new target and all of the necessary source files.

  PowerJ now displays a form design window instead of a code editor:



You can drag from the objects (components) on the *component palette* to the form design window to create a user interface. The first step is to click one of the objects on the component palette. Then you can move the cursor to the form design area, press and hold down the left mouse button, and drag the cursor diagonally to specify the object's position and size. If you click on the form without dragging the cursor, PowerJ adds a default-sized component.

The first step in the design of this applet is to add a list at the top of the form.

♦ **Add a list:**

1. Click the List button (indicated by the **java.awt.List.11** tooltip) on the **Standard** page of the component palette.
2. Point anywhere in the upper-left quarter of the form design window. The cursor changes from an arrow to a crosshair.
3. Hold down the left mouse button and drag the cursor diagonally across the form. While you are dragging, a rectangular outline shows you the size that the list object will be.
4. Release the mouse button.

This creates a list on the form. You can move the list by dragging it and resize it by dragging one of the resizing handles on the list's corners and edges.

The next step in designing the form is to add a text field.

- **Add a text field:**

  1. Click the Text Field button (**java.awt.TextField.11**) on the **Standard** page of the component palette.
  2. In the form design window, drag diagonally to create a text field below the list.

     The form now has a list and text field. You can change the size and position of the text field the same way you change the size and position of the list.

The next step in designing the form is to add a button.

- **Add a button:**

  1. Click the Button button (**java.awt.Button.11**) on the **Standard** page of the component palette.
  2. In the form design window, click under the text field to create a standard-size button (which can also be moved and resized).

You can now run your applet to see the default behavior of the objects that you have added.

- **Run your applet and test the program interface:**

  1. On the **Run** menu of the main menu bar, click **Run** to prepare your program for execution.
  2. Wait for the applet to be compiled and executed in the Applet Viewer. Once the Applet Viewer has loaded your applet, the objects you placed on the form will appear in the Applet Viewer window. This is the program that you have created.
  3. Experiment with your new program. Notice that you can edit text in the text field and click the button, but the list does not change.
  4. Close the Applet Viewer to terminate your program.

The next step in creating a program is to adjust the properties of the objects you placed on the form. The properties affect the appearance and behavior of the objects.

- **Label the button:**

  1. Use the right mouse button to click the button you placed on the form. Then click **Properties** to display the properties for the button.
  2. On the **General** page, click in the **Label** text field, then type `Add Text`.
  3. Click **OK**.

  The next step in creating this program is to write code that will respond to user actions. You need to add the code that will be invoked when a user clicks the button. When this happens, the program should retrieve the text that is currently in the text field and add that text to the list object.

  You can use *drag-and-drop programming* to create the code with very little typing. Using drag-and-drop programming means dragging from objects placed on the form design window to the code editor window. This opens the *Reference Card*, which allows you to easily apply methods and properties to objects.

  Although you can always type code directly into the code editor, drag-and-drop programming will assist you when you do not know the methods available for a class, or when you cannot remember the syntax for a method. Typically, you will use the drag-and-drop coding facilities less as you become more familiar with the Java classes you are using.

- **Create code to be invoked when the user clicks the button:**

  1. Use the right mouse button to click the button. Point to **Events**, then click **java.awt.event.Action.actionPerformed**.

     This opens a code editor window for the **actionPerformed** event handler. The **actionPerformed** event will be triggered when a user clicks on the button, which will termed the *command button* for the remainder of this tutorial.
  2. Move the code editor window to the bottom right of the screen so that you can see at least part of each object in the form design window.

  You can now create the code that responds to a user clicking the command button. When the user clicks the command button, you want the applet to respond by retrieving the text that is currently in the text field.

- **Generate code to retrieve the text from the text field:**

  1. Drag from the text field in the form design window to the blank line below the opening brace of the **actionPerformed** event handler in the code editor window, then release the mouse button. The Reference Card opens.

     The first time that you open the Reference Card by dragging from an object, it automatically opens to that object's class on the **Contents** page. The **Contents** page has a hierarchical view of packages, classes, methods, and properties.

  2. Click the **Find** page on the Reference Card to find a method for **java.awt.TextField.11**.

  3. In step 2 of the **Find** page, type **getText**.

  4. In step 3 of the **Find** page, click **text**.

  5. In step 4 of the **Find** page, click **java.lang.String getText()**.

  6. Click **Parameters** to open the Parameter Wizard for the **getText** method.

     The selected **Object Prefix** should be `textf_1`. Make sure that **Store in a variable** is checked and that the variable is `text`.

  7. Click **Finish** to generate code in the editor.

  The **actionPerformed** event handler now has an additional variable, which is assigned a copy of the string present in the text field when the user clicks the command button. The code added by the Parameter Wizard to retrieve the text is:

```
java.lang.String                text;
text = textf_1.getText();
```

  The next step is to add code to put the text from the text field into the list.

- **Generate code to copy the text to the list:**

  1. Drag from the list on the form design window to the code editor window, leaving a blank line after the **getText** function call of the **actionPerformed** event handler.

     The Reference Card for **java.awt.List.11**, opens to the **Find** tab.

  2. In step 2 of the **Find** page, type `add`, since you are looking for a method to add text to the list. The list will show only the entries containing `add`.

3. In step 3 of the **Find** page, click **addItem**.
4. In step 4 of the **Find** page, click the version of **addItem** that only takes a single string argument (the first one shown). This method will add the text to the end of the list.
5. Click **Parameters** to open the Parameter Wizard so that you can specify the parameters for **addItem**.
6. The selected **Object Prefix** should be `lb_1`. Use the right mouse button to click the **java.lang.String item** text field, then click **Variables** on the context menu. This shows a list of the variables that are defined in the event handler, so you can pick a variable to use as the parameter.
7. In the Variables dialog box, click **text** (the variable holding the string), then click **OK**. This specifies that the `text` variable will be used as the parameter.
8. Click **Finish** to generate code in the editor.

The code is now complete and should look like this:

```
    public void cb_1_actionPerformed(
java.awt.event.ActionEvent event )
    {
        java.lang.String                text;
        text = textf_1.getText();

        lb_1.addItem( text );

    }
```

If there are any differences, change your code to match the above sample.

- **Run your program:**

1. In the **Run** menu of the code editor window, click **Run** to start your program.
2. Wait for your applet to be compiled and started.
3. Type text in the text field, then click the command button. Notice that the text is added to the list.
4. Modify the text in the text field, then click the command button. The modified text is added to the end of the list.

Close the Applet Viewer when you are finished experimenting with your program.

- **Save your project:**

  1. On the **File** menu of the main menu bar, click **Save Project**. This displays the contents of the `Projects` folder.
  2. Under **Folder name**, type `List` and click **Save**. This creates a folder called `List` for your project.
  3. Under **File name**, leave `List.wxj` as the project file name and click **Save**.

     PowerJ creates a project file named `List.wxj` in the new folder named `List`, along with other files and folders of your project.

Improving your applet

This section shows you how to improve your applet in several ways. The new version of the program you create will clear the text field after adding a new line to the list. It will also check to see if the text field actually contains text; if the text field is empty, the program will not try to add a blank line to the list.

The new source code that will do this work will be added to the existing **actionPerformed** event handler for the command button.

- **Generate code to determine if there is text in the text field:**

  1. Drag from the text field on the form design window to the blank line after the **getText** function call in the code editor window. This opens the Reference Card.
  2. In step 1 of the **Find** page, in the drop-down list, click **java.lang.String.11 Functions**. This changes the Reference Card to show methods for the Java **String** class.
  3. In step 2 of the **Find** page, type `length` to find method names that include “length”.
  4. In step 3 of the **Find** page, click **length** to select the **length** method of the **String** class. This method returns the length of the string.
  5. Click **Parameters**. This opens the Parameter Wizard for the **length** method.
  6. In the **Object Prefix** drop-down list, click **text**. This specifies the object of the length method.
  7. Make sure **Store in a variable** is checked and that the variable is `result`.

8. Click **Finish** to generate code in the editor.

The code editor window now contains a declaration for an integer named `result` and a statement that assigns the value of `text.length()` to `result`.

To avoid adding empty lines to the list, the next lines of the code should make having a string of at least one character in the text field a condition for adding the text field contents to the list.

♦ **Generate code to check for a non-zero string length:**

1. After the line calling the **length** method, start a new line and type:
   ```
   if ( result > 0 ) {
   ```
2. Select the next line (containing **addItem**). On the **Edit** menu, click **Indent** to indent the line.

The code in the code editor window should look like this:

```
    public void cb_1_actionPerformed(
java.awt.event.ActionEvent event )
    {
        int                             result;
        java.lang.String                text;
        text = textf_1.getText();
        result = text.length();

        if ( result > 0 ) {
            lb_1.addItem( text );

    }
```

The `if` statement checks whether the length of the text that the user typed is greater than zero. If it is, your code can retrieve the text.

Next you add code to clear the text field for new input. The following steps show you how to do this using the Reference Card and Parameter Wizard.

♦ **Generate code to clear the text field:**

1. Drag from the text field to the blank line after the **addItem** statement in the code editor window. This opens the Reference Card for **java.awt.TextField.11**.
2. In step 2 of the **Find** page for **TextField**, type `text`. This will display method names that contain “text”.
3. In step 3 of the **Find** page, click **text.**
4. In step 4 of the **Find** page, click **void setText(java.lang.String)**, then click **Parameters** to open the Parameter Wizard.

5. The **Object Prefix** should be `textf_1`. Type two double quotes (`""`) in the **java.lang.String t** text field to specify an empty string.
6. Click **Finish** to generate code in the editor.

The editor now has code to clear the text field. Note that it is at the same level of indentation as the preceding code.

- **Complete the if statement:**

1. In the code editor, type a closing curly brace (`}`), then start a new line.

   The closing curly brace returns to the same level of indentation as the beginning of the `if` statement.

The next step is to set the input focus to the text field, to make it easier for a user to enter new text after clicking the **Add Text** button.

- **Generate code to return the focus to the text field:**

1. In the **Help** menu, click **Reference Card**. In step 2 of the **Find** page for **TextField**, type `focus`. In step 3, click **requestFocus**, then double-click **void requestFocus( )** in step 4 to set the parameters.
2. In the **ObjectPrefix** drop-down list, click **textf_1**.
3. Click **Finish** to generate code in the editor.

The editor now has code to set the focus to the text field.

Your final code should look like this:

```
    public void cb_1_actionPerformed(
java.awt.event.ActionEvent event )
    {
        int                             result;
        java.lang.String                text;
        text = textf_1.getText();
        result = text.length();

        if ( result > 0 ) {
            lb_1.addItem( text );
            textf_1.setText( "" );
        }
        textf_1.requestFocus();


    }
```

Run the program as usual. Notice that the input focus returns to the text field after you click the command button. Click the command button when the text field is empty to see that the program does not try to add the blank line to the list. Close the Applet Viewer before continuing.

Debugging your program

This section examines how to use the debugging features of PowerJ with your program. In particular, this section demonstrates the use of a *breakpoint*. If you set a breakpoint on a statement in your program's source code, program execution stops when it reaches that statement. While the program is stopped at the breakpoint, you can perform a variety of operations to examine the current state of your program.

Note: Make sure that your program is *not* running before you try to set a breakpoint.

- **Set a breakpoint in your code:**

  1. If the code editor window is not open, use the right mouse button to click on the command button, then point to **Events** and click **java.awt.event.Action.actionPerformed**. This opens a code editor window showing the **actionPerformed** event handler for the command button.
  2. Use the right mouse button to click on the line:

     ```
     result = text.length();
     ```

  3. Click **Toggle Breakpoint** on the context menu.

  The **Toggle Breakpoint** action sets a breakpoint on the line.

  Note: If there is already a breakpoint on a line, clicking **Toggle Breakpoint** removes the breakpoint.

  When you set a breakpoint on a statement, PowerJ marks the breakpoint with a red stop sign icon to the left of the statement. You can also double-click on a line's icon to toggle whether it has a breakpoint or not.

  Once you have a breakpoint in your code, you are ready to run your applet again. The breakpoint will only work if you are running the applet with a *Virtual Machine (VM)* that supports the debugging facilities needed by PowerJ.

- **Turn on debugging:**

  1. On the **Run** menu of the main menu bar, click **Run Options**. This opens the Run Options dialog.

     By default, your program is run using Sun's Java Interpreter.

  2. Click the **Debug** tab and make sure **Run with the debugger** is selected.

3. Click **OK**.

   This specifies that PowerJ will activate the debugger when you run your program.

You can now run the program to see what happens when execution reaches the breakpoint.

♦ **Run the program up to the breakpoint:**

1. On the **Run** menu, click **Run**.
2. When the form appears, type some text in the text field, then click the **Add Text** command button.

When you click the command button, the program executes the **actionPerformed** event handler and hits the breakpoint that you set. Program execution then "freezes", or suspends, so that you can examine the current state of your program.

In the code editor window showing the **actionPerformed** event handler, you will see a yellow arrow in the left margin, pointing to the statement where you placed the breakpoint. This shows the point of execution. In this case, execution is suspended at the breakpoint.

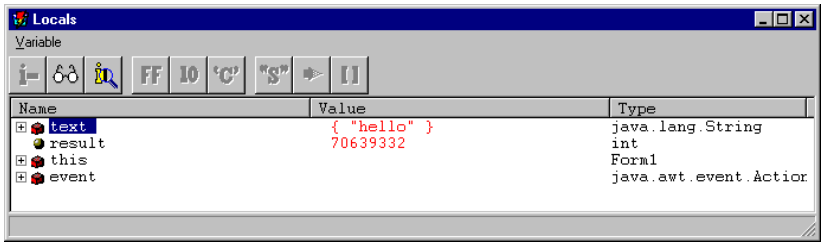
Note: When you set a breakpoint, execution stops just *before* executing the statement where the breakpoint is set.

When your program is stopped at a breakpoint, you can examine the local variables of your program using the Locals view.

♦ **Open the Locals view:**

1. On the **Debug** menu of the code editor window, click **Locals**.

This displays the following window:



The Locals window shows the local variables defined at this point in the program. It also shows `this`, referring to the form itself. If you click on the + sign to expand `this`, you see the objects on the form. By expanding other items in the Locals view, you can examine the contents of all the objects on the form. For example, you can see the text that is stored in the text field.

After stopping at a breakpoint, you can continue executing your program one statement at a time.

- **Step through the code:**

  1. On the toolbar of the code editor window, click

  This is the **Step Over** button. You will see the yellow arrow move to the next statement in your program. This means that PowerJ has executed the statement at the breakpoint and has "stepped over" to the next statement.

  As you execute the statements, the Locals view updates to display changed values. For example, when a new value is assigned to `result`, the Locals view displays the new value.

  If you click the **Step Over** button again, execution moves on to the next statement. By repeatedly clicking the button, you can execute your program one statement at a time.

- **Return to normal execution:**

  1. On the **Run** menu of the code editor window, click **Run**.

  This starts your program running normally again, instead of following the statement-by-statement operation used in the previous section.

  The next time you click the command button, your program executes the **actionPerformed** event handler and runs into the breakpoint again. The program will stop as before, giving you a chance to examine local data again.

  Many other debugging features are available through the **Debug** and **Run** menus of the code editor window while you are stopped at a breakpoint.

> **Note:** To remove a breakpoint, close your program, then right-click on the line where the breakpoint has been set and click **Toggle Breakpoint**.

Using a web browser

By default, PowerJ Learning Edition runs your applet using the Applet Viewer and Sun's implementation of the Java VM. If you have a web browser that supports JDK 1.1 installed on your system, you can run your applet under the browser instead of on the Applet Viewer.

- **Run your applet under a web browser:**

  1. On the **Run** menu of the main menu bar, click **Run Options**. This opens the Run Options dialog.
  2. On the **General** page of the Run Options dialog, click **Use a web browser**, then click **Configure**. This opens a configuration dialog box.
  3. Click **Netscape Navigator** or **Internet Explorer**, then click **OK**.
  4. If the field under **Initial URL** is blank, click the associated **Browse** button. This opens a file dialog box for selecting the name of an HTML file.
  5. Open the `Release` folder, click `List.html`, and then click **Open**.
  6. Click **OK** to close the Run Options dialog.
  7. On the **Run** menu of the main menu bar, click **Run**.

     PowerJ launches the web browser you configured and runs your program under it.

> **Note:** You can debug your program under a browser only if you are using Internet Explorer. PowerJ Learning Edition does not allow debugging in Netscape Navigator. When you configure the run options to run your applet under a browser, PowerJ automatically turns off debugging. Therefore, if you are using Internet Explorer and wish to debug your program, you have to turn debugging on again.

- **Configure Run Options to debug your program under Internet Explorer:**

  1. Open the Run Options dialog, as before.
  2. Click the **Debug** tab, then click **Run with the debugger**.
  3. Click OK to close the Run Options dialog.

Note: When debugging your program under a browser, you follow the same procedures for starting your program, toggling breakpoints, and viewing the debugging features of PowerJ as you do when debugging your program under the Applet Viewer.

You have now created two simple Java programs (a command line application and a framework applet) and explored the basic capabilities of PowerJ as a tool for writing Java code.

Product features

PowerJ Learning Edition replaces the usual command line interface with a JavaBean-based visual development environment for the Java programming language. Its drag-and-drop programming, in-context debugging, and syntax-highlighting tools save time and make Java easier to learn.

PowerJ Learning Edition includes a subset of the features found in PowerJ Enterprise Edition. The following table provides references to detailed descriptions of the features of PowerJ.

For more information on:	**See the following section of the *PowerJ Programmer's Guide*:**
Basic concepts of Java	*Java* in Chapter 1
Using forms in PowerJ	*Forms* in Chapter 1
Using targets in PowerJ	*Targets* in Chapter 2
Using projects in PowerJ	*Projects* in Chapter 2
Contents and organization of PowerJ target folders and files	*Target folder contents* in Chapter 2
Properties of Java applets	*Java applets* in Chapter 2
Properties of Java applications	*Java applications* in Chapter 2
Using the PowerJ form design window	*Using the form design window* in Chapter 3
Adding component palette objects to a form	*Adding objects to a form* in Chapter 3
Using PowerJ templates as form design shortcuts	*Templates* in Chapter 3

PowerJ startup options	*Startup options* in Chapter 3
Using the PowerJ code editor window	*The PowerJ code editor* in Chapter 3
Importing classes	*Importing classes* in Chapter 3
Using drag-and-drop programming	*Using drag-and-drop programming* in Chapter 3
Debugging with PowerJ	*Debugging* in Chapter 4
Importing Java code into PowerJ	*Importing from other Java environments* in Chapter 7
Writing PowerJ code	*Using JDK and PowerJ components* in Chapter 9
Layout managers	*Layout managers* in Chapter 13
The **Resize Percentages** property	*Resize percentages property* in Chapter 13
Creating JavaBeans components	*Creating JavaBeans Components* in Chapter 18

PowerJ Enterprise Edition includes all the components of PowerJ Learning Edition, plus many additional features. The extra features of PowerJ Enterprise Edition are:

- Support for JDK 1.02
- Support for the Microsoft VM
- Support for CORBA
- Database Components (including Transaction Object, Query Object, Data Navigator, and databound versions of standard controls)
- Client-side caching for high-performance database applications
- Sybase's jConnect for JDBC
- Development version of Powersoft Jaguar CTS, a high- performance component transaction server designed for delivering scalable transactional applications for WebOLTP.
- Support for importing Visual Café projects
- Development version of Sybase Dynamo
- Sybase SQL Anywhere, a scalable SQL database solution for workgroups of any size.
- Support for the included revision control system

- Support for Powersoft ObjectCycle and other systems
- Powersoft Components, including Tab Control, Grid Control, and Picture Box, and many other integrated components, classes, and drivers from JScape, KL Group, ObjectSpace, Visigenic, and XDB
- Support for collection targets (ZIP, JAR, and CAB), Java servlets, and Jaguar CTS
- Support for Web Application target

Samples

PowerJ comes with a number of sample projects that you may like to examine or experiment with. The sample projects available are:

Sample Name	Description	Keywords
Application	Queries and displays the Java System Properties.	Java System Properties
Button	Demonstrates the Button component.	Button
CheckBox	Demonstrates the use of Checkbox component and a CheckboxGroup. A CheckBoxGroup guarantees that, at most, one item is selected from the group of Checkboxes.	CheckBox
Choice	Demonstrates the Choice component.	Choice
JSocket	Demonstrates some capabilities of the PowerJ socket component. Used with JServerSocket.	Socket
JServerSocket	Demonstrates socket-based communication. Used with JSocket.	ServerSocket
ResizePercentages	Demonstrates using the resize percentage properties of components to handle window resizing.	Resize Percentage
ScrollBar	Demonstrates the ScrollBar component. Note: Runs best in a web browser. In the development environment you may have to uncheck the "Vertical" property on each ScrollBar's property sheet.	ScrollBar Scroll events
SortableVector	Demonstrates the SortableVector class.	TextArea Sortable

		Vector
TextArea	Demonstrates the TextArea component, menus, and a simple dialog.	TextArea Menu Dialog
Thermometer	Uses a scroll bar to simulate a thermometer.	Scroll bar
Threads	Uses various methods to control three different threads. Each thread can be started by the user, then stopped, suspended, and resumed. You can also change the priority of the thread.	Thread

- **Load a sample project:**

  1. In the **File** menu of the main PowerJ menu bar, click **Open Project**. This displays the Open Project dialog box that lets you choose the project you want to open.
  2. Use the dialog box to find the main `PowerJ21LE` folder.
  3. Double-click the folder icon for **PowerJ Samples**. This displays a new list of folders, corresponding to various sample programs.
  4. Double-click one of the folder icons. This displays the contents of the associated folder.
  5. Double-click the project file in this folder. (The name of this file should be the same as the name of the folder, with the extension .WXJ added.)

  When you double-click this project file, PowerJ loads the associated sample project. You can run this program to see how it works. You can also examine the properties of each object on the program's forms and examine the event handlers for these objects.