[sun.com](http://www.sun.com)

| http://java.sun.com/developer/onlineTraining/tools/netbeans\_part1/ | Jun 05, 2007 |
| --- | --- |

Tutorials

Getting Started With the NetBeans IDE Tutorial

|  | |  |  | |  |  | | --- | --- | | | --- | --- | --- | --- | --- |  | By [Dana Nourie](http://java.sun.com/features/authors.html#nourie), May 22, 2005, Updated May 2007 |  | |  | | --- | | | --- | --- | --- | --- |  |  | | --- | | [Download NetBeans Download NetBeans IDE](http://www.netbeans.org/downloads/index.html) |   Content  [NetBeans IDE Nuts and Bolts](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#nuts#nuts)  [Features and Tools](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#features#features)  [Getting Familiar With NetBeans](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#familar#familar)  [Using the Source Editor](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#source#source)  [Using Forms and Wizards to Create Code](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#forms#forms)  [Importing Existing Java Source Code Into NetBeans](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#importing#importing)  [Creating, Compiling, Debugging, and Running an Application](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#creatingcompiling#creatingcompiling)  [Creating the Pay Calculator Application](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#creating#creating)  [Refactoring](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#refactoring#refactoring)  [Checking Your GUI](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#checking#checking)  [Debugging](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#debugging#debugging)  [Adding Menus and Images](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#menus#menus)  [Adding Event Handlers](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#events#events)  [Documenting Your Work](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#documenting#documenting)  [Summary](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/#summary#summary)  Using an Integrated Development Environment (IDE) for developing applications saves you time by managing windows, settings, and data. In addition, an IDE can store repetitive tasks through macros and abbreviations. Drag-and-drop features make creating graphical user interface (GUI) components or accessing databases easy, and highlighted code and debugging features alert you to errors in your code.  The NetBeans IDE is open source and is written in the Java programming language. It provides the services common to creating desktop applications -- such as window and menu management, settings storage -- and is also the first IDE to fully support JDK 5.0 features. The NetBeans platform and IDE are free for commercial and noncommercial use, and they are supported by Sun Microsystems.  This tutorial is aimed at those who are new to using IDEs, fairly new to programming, and new to the Java platform. You'll learn to create a simple desktop application with a GUI interface and functionality that calculates overtime pay using basic features of the NetBeans IDE. This tutorial provides explanations for the code where appropriate, as well as links to the [Java API](http://java.sun.com/j2se/1.5.0/docs/api/index.html) and information about objects as they are introduced. Though the NetBeans environment also provides many rich features for the various Java platforms, such as Java Platform, Enterprise Edition (Java EE) and Java Platform, Micro Edition (Java ME), this article covers only Java Platform, Standard Edition (Java SE) technology, which is generally the entry point for new developers and programmers. Future tutorials will discuss more advanced features.  To follow this tutorial, you need to have downloaded and installed the [JDK and the NetBeans IDE](http://java.sun.com/javase/downloads/). Or you can download [JDK 5.0](http://java.sun.com/j2se/1.5.0/download.jsp) and [NetBeans 4.1](http://www.netbeans.org/downloads/index.html) separately to ensure that you have the latest versions.  NetBeans IDE Nuts and Bolts   |  | | --- |   Because the NetBeans IDE is open source and in continual improvement, you may notice slight differences between the screen captures in this article and the latest download. This tutorial is based on NetBeans 4.1 and may vary slightly from later versions as they become available.  Features and Tools   |  | | --- |   The NetBeans IDE has many features and tools for each of the Java platforms. Those in the following list are not limited to the Java SE platform but are useful for building, debugging, and deploying applications and applets:  Source Code Editor  Syntax highlighting for Java, XML, HTML, CSS, JSP, IDL  Customizable fonts, colors, and keyboard shortcuts  Live parsing and error marking  Pop-up Javadoc for quick access to documentation  Advanced code completion  Automatic indentation, which is customizable  Word matching with the same initial prefixes  Navigation of current class and commonly used features  Macros and abbreviations  Goto declaration and Goto class  Matching brace highlighting  JumpList allows you to return the cursor to previous modification  GUI Builder  Fully WYSIWYG designer with Test Form feature  Support for visual and nonvisual forms  Extensible Component Palette with preinstalled Swing and AWT components  Component Inspector showing a component's tree and properties  Automatic one-way code generation, fully customizable  Support for AWT/Swing layout managers, drag-and-drop layout customization  Powerful visual GridBagLayout Customizer  Support for null layout  In-place editing of text labels of components, such as labels, buttons, and text fields  JavaBeans support, including installing, using, and customizing properties, events, and customizers  Visual JavaBean customization -- ability to create forms from any JavaBean classes  Connecting beans using Connection wizard  Zoom view ability  Database Support  Database schema browsing to see the tables, views, and stored procedures defined in a database  Database schema editing using wizards  Data view to see data stored in tables  SQL and DDL command execution to help you write and execute more complicated SQL or DDL commands  Migration of table definitions across databases from different vendors  Works with databases, such as Oracle, IBM DB2, Microsoft SQL Server, PointBase, Sybase, Informix, Cloudscape, MySQL, and more  The NetBeans IDE also provides full-featured refactoring tools, which allow you to rename and move classes, fields, and methods, as well as change method parameters. In addition, you get a debugger and an Ant-based project system.  Getting Familiar With NetBeans   |  | | --- |   To get started, download the latest stable version from the [NetBeans.org](http://www.netbeans.org/downloads/index.html) web site and install it on whatever platform you use for programming and development. The NetBeans.org web site lists the computer requirements needed to run the IDE.  NetBeans can automatically upgrade its core and extension modules over the Internet, and it has a module that runs periodically on your behalf to check for updates to the version of NetBeans you're using. In addition, the Update Center can update and install any modules you okay or request.  When you start the application, you should get a welcome screen similar to Figure 1.   | | [Figure1: Opening Screen](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/image/OpenLrg.jpg) | | --- | | Figure 1: Welcome Screen Click to enlarge | | | --- | --- | --- |   You'll notice that this welcome screen makes getting started right away an easy process by displaying a Quick Start Guide up front, as well as the options to begin a project or open one. In addition, you can select the Sample Project for a quick example of how code is set up in this IDE. NetBeans is fairly intuitive to use, especially if you've used IDE software in the past.  You'll get familiar with three concepts right away: projects, nodes, and workspaces.  Within NetBeans, you work within the context of a project, which consists of an organized group of source files and associated metadata; project-specific properties files; an Ant build script and run settings; and all the tools you'll need to write, compile, test, and debug your application. You can create a main project with subprojects, and you can link projects through dependencies. So getting started is as easy as giving your project a name.  Once you tell NetBeans the name of a new project, it then  Creates a source tree with an optional skeleton class inside  Creates a folder for unit tests  Sets classpaths for compiling, running, and testing  Sets the Java platform the project is to run on  Creates an Ant build script (build.xml), which contains instructions that the IDE uses when you perform commands on your project, such as compile or run  Click File from the main menu and select New Project. The New Project wizard pops up and looks similar to Figure 2.   | | [Figure 2: New Project Wizard](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/NewProjectBg.jpg) | | --- | | Figure 2: New Project Wizard Click to enlarge | | | | --- | --- | --- | --- | | | Tabs Window | | --- | | Figure 3: Tabs Window | | |                       From there, you select a Category from General, Web, Enterprise, Samples. Notice that General creates a project containing an empty application. Choose General and select Java Application. Next, name the project and select a location within your file system. Note that the IDE automatically creates a main class for your application if you want it to. Click Finish.  The Project window displays only the files that are likely to be regularly edited, such as source files and tests. To see more details about your project, click on the Files tab. Here, if you click on the project folder you created, you'll see that the folder contains the Ant script and properties files that control how your project is built and run:  nbproject/build-impl.xml -- Contains the instructions for handling your project. The IDE generates this file, which you should not edit.  build.xml -- Imports targets from build-impl.xml. You can edit this file to override existing targets or create new targets.  The project folder also contains the output folder for compiled classes, JAR files (for Java SE projects) or WAR files (for web projects), and Javadoc.   | Tip Many menu commands such as Run and Build have keyboard shortcuts to the main project. You can also access all commands for individual projects by right-clicking their project nodes in the Projects window. |  | | --- | --- |     NetBeans allows you to see all your objects in a project represented as nodes of a tree, each having its own icon to represent the type of object the node represents. Within the Files tab, you can easily view the trees and representative nodes. If you double-click on a node, it opens up into a subtree that contains more detail. You can collapse or expand trees as necessary. Right-clicking on any node provides easy access to specific functions that you can perform and tools that you can use on that object. Expand the subtrees of the project node that you just created, and you will notice that the fields, constructors, methods, and bean patterns appear as node branches.  The Runtime tab shows you a view of what has happened when you tried to run your application. The Runtime view lists various facilities available to your project. In addition, you can see what operations have been performed and troubleshoot some types of runtime errors, such as when a [remote method invocation (RMI)](http://java.sun.com/developer/onlineTraining/rmi/index.html) connection is causing a problem. You'll get more familiar with this view after you have written, compiled, debugged, and run your applications.  Return to the Files tab. Double-clicking a source file automatically opens the file in the workspace to the right, bringing up the appropriate Source Editor. Notice the file system that has been created. Files and directories associated with an application project are organized in a logical fashion.  Double-click the Main.java source file to see the Source Editor similar to the one in Figure 4.   | | [Figure 4: Source Editor](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/SourceEditor1Bg.jpg) | | --- | | Figure 4: Source Editor Click to enlarge | | | --- | --- | --- |                           When your application is organized into several projects, the main project serves as the entry point to the application. Usually, the main project contains the main class that starts the application. To make a project the main project, right-click the project's node in the Projects window and choose Set Main Project.  Using the Source Editor   |  | | --- |   You can click right into the Source Editor and write code. When you write code this way, you'll notice that the editor automatically highlights code in color as appropriate and completes your code.  You can customize colors in the highlighting by going to Tools -> Options -> Editing -> Editor Settings -> Java Editor -> Fonts and Colors.  Code completion finishes package names, classes, interfaces, and common methods. As handy as code completion can be, you may sometimes not want it. You can easily turn off this feature by pressing Esc. Or you can turn it off more permanently by going to Tools -> Options -> Editing -> Editor Settings -> Java Editor and deselecting the Code Completion Instant Substitution checkmark.  You can also save time by assigning abbreviations that the Source Editor expands for you. Type the first few letters of an abbreviation and press the spacebar. The Source Editor then expands the abbreviation.  Turn on line numbering easily through the View menu.  Using Forms and Wizards to Create Code   |  | | --- |   The NetBeans IDE also provides forms, templates, and wizards that make creating code easy. In the Projects window, you can also add to your source code by right-clicking the fields, methods, and bean patterns on the nodes. By selecting Add on the menu that appears, you get forms similar to Figures 5 and 6.   | | Figure 5: Add New Field | | --- | | Figure 5: Add New Field Form | | | | --- | --- | --- | --- | | | Figure6: Add New Method | | --- | | Figure 6: Add New Method Form | | |                                     Importing Existing Java Source Code Into NetBeans   |  | | --- | | Managing the Classpath NetBeans automatically adds everything on your project's compilation classpath to the project's runtime classpath. To manage a project's classpath:   In the Projects window, right-click the project's node and choose Properties.   In the Project Properties dialog box, click the Libraries node in the left pane.   Select the Compile tab in the dialog box's right pane.   Add the necessary elements to the project's compilation classpath by clicking the appropriate button. You can add any of the following:  Project. The [JAR](http://java.sun.com/docs/books/tutorial/jar/) file or WAR file produced by another project, as well as the associated source files and Javadoc documentation  Library. A collection of JAR files or folders with compiled classes, which can optionally have associated source files and Javadoc documentation  JAR file. A JAR file somewhere on your system |   You can use two project templates to import your existing source code:  Java Project With Existing Sources -- Use this project template to create a standard Java project. A standard Java project uses the IDE to compile, run, and debug your application.  Java Project With Existing Ant Script -- Use this project template to create a free-form Java project. A free-form Java project uses your own Ant script to compile, run, and debug your project.  The following steps cover use of the first template to create a standard Java project without the use of existing Ant scripts.  Choose File -> New Project (Ctrl-Shift-N).  Choose General -> Java Project With Existing Sources.  Click Next.  In the Project Name field, type a name for your project. Notice that the IDE automatically suggests a name for the project folder.  Click the Browse button next to the Project Folder field and create a folder somewhere on your system for your NetBeans projects. Then select the newly created NetBeans projects folder. Be sure to select the Set as Main Project box, then click Next.  In the Source Packages Folder pane, click Add Folder, navigate to the directory of source files, and select the resources and src folders.  (Optional) In the Test Packages Folder pane, click Browse to select the folder containing the JUnit package folders.  Click Finish.  Your project will now be displayed in the Projects and Files windows.    Creating, Compiling, Debugging, and Running an Application   |  | | --- |   Now that you're familiar with the nuts and bolts of the NetBeans IDE, the next sections will step you through creating, compiling, debugging, and running an application that uses a simple GUI interface with some background functionality that calculates the rate of regular and overtime pay.  Your finished application will look something like Figure 7.   | | Figure 7: Pay Calculator Application | | --- | | Figure 7: Pay Calculator Application | | | --- | --- | --- |                         In creating this application, you will learn how to  Use the Form Editor  Use the Source Editor  Use the Palette, Inspector, and Properties features  Add event handlers  Debug your code  Refactor your code  Compile and run your application  Though you can do a great deal through wizards, forms, and drag-and-drop features, you still need to write some lines of code for the functionality of the application. The following sections explain some of these lines of code.  Creating the Pay Calculator Application   |  | | | --- | --- | | | [Figure8: New Project](http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/NewProjectBg.jpg) | | --- | | Figure 8: New Project Window Click to enlarge | | |   Click File from the main menu and select New Project. The Project wizard starts. Next, select General in the Categories window and select Java Application in the Projects pane. Click Next.  In the next window, you can leave the suggested project name or specify another name. Choose a Location and Folder for this application. Deselect the Create Main Class box.  Now click Finished, which takes you to the main program.  To begin creating the GUI, you are going to start the Form Editor by creating a top container for your application using a specific form. The Java API provides GUI components, often referred to as [Swing](http://java.sun.com/developer/onlineTraining/GUI/Swing1/shortcourse.html), and provides three useful top-level container classes: [JFrame](http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JFrame.html), [JDialog](http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JDialog.html), and [JApplet](http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JApplet.html). Every GUI component must be part of a containment hierarchy. A containment hierarchy is a tree of components that has a top-level container as its root. Each GUI component can be contained only once. If a component is already in a container and you try to add it to another container, the component will be removed from the first container and added to the second. Each top-level container has a content pane that, generally speaking, contains (directly or indirectly) the visible components in that top-level container's GUI. You have the option to add a menu bar to a top-level container.  Select the File menu entry, then select New File, and select Java GUI Forms from Categories. Notice the File Types that appear to the right. For this application, you will use a JFrame Form. Select JFrame Form, then click Next. Name the JFrame SamplesJFrame, and click Finish.  To see what the wizard has created, click on the Files tab. You'll see the files the NetBeans IDE has created, as well as the SamplesJFrame class. Double-click SamplesJFrame so that it opens in the Source Editor on the right. You'll notice that the code has been generated.  In the middle section of the workspace window, you should see three tabs as well as two buttons that allow you to toggle between the Design view, which gives you a visual view of your application, and Source view, which allows you to work with the raw code. In other words, you are switching from the forms GUI drag-and-drop editor to the code editor. Each editor gives you slightly different options on the menu bar, as shown in Figures 9 and 10.   | | Figure 9: Design Form Editor View | | --- | | Figure 9: Design Form Editor View | | | --- | --- | --- |      | | Figure 10: Source Editor View | | --- | | Figure 10: Source Editor View | | | --- | --- | --- |     Click into Design view to use the Form Editor as shown in Figure 9. Now you can begin the process of building the GUI interface. You've started this application with the JFrame container. Now you can easily add other components to it visually by selecting a component from the Palette pane and adding it to the application. Changing component properties is just as easy.   | | Figure 11: NetBeans Form Editor Design View | | --- | | Figure 11: NetBeans Form Editor Design View | | | --- | --- | --- |   From this editor, you will add and edit components by using three panes that are docked at the sides of the IDE (see Figure 11):  Palette -- Contains JFC/Swing, AWT, [Layout managers](http://java.sun.com/docs/books/tutorial/uiswing/layout/using.html), and [JavaBeans](http://java.sun.com/developer/onlineTraining/Beans/beans02/index.html) components.  Inspector -- Displays a tree hierarchy of all components contained in the currently active form, including visual components and containers, such as buttons, labels, menus, and panels. It can also include nonvisual components, such as timers and data sources.  Properties -- Displays the properties of the component currently selected in the Inspector, Editor, Projects, or Files windows. You can also right-click a component to pop up a properties box for that component.  As you use components from the Palette, NetBeans automatically generates code, instantiating those objects. If you change the components' properties by using the Properties pane, that code is also generated. You'll notice as you look at the Source Editor that this generated code appears as blue guarded blocks. It is recommended that you not change this code. However, you can modify the way initialization code is generated and even write custom code to be placed within the initialization code. For this tutorial, do not change the generated code.  So far, you have created a project and a JFrame class called SamplesJFrame. Next, you'll build on the JFrame component and add other components. Click on [JTabbedPane](http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JTabbedPane.html) in the Palette, then click on the blank square in the design form. Notice in the Inspector window that the components you've created so far are listed there.  Next, click on [JPanel](http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JPanel.html) in the Palette, then click in the main area of the JTabbedPane. Note that a new tabbed pane has been added to JTabbedPane. Add another JPanel, making sure that you select jTabbedPane1[JTabbedPane] in the Inspector window. Then hold down the Shift key and click on JPanel in the Palette, then on the frame in the workspace. Another tab appears. You can add as many tabs as you like, but this tutorial covers only two.  Click on any of the objects in the Inspector pane, and you'll see that the properties for each component appear in the Properties pane. In addition, you can right-click the components in the Inspector pane and select Properties from the menu to make the Properties pane pop up.  To change the print for each tab, click on the corresponding JPanel in the Inspector window. For instance, click on jTabbedPane1 to select it. Next, go to the Properties window on the right and scroll down until you see the Tab Title listed. Click on the three-dot ellipsis button (...) at the right, and a window pops up that allows you to replace the current text. Type in Pay Calculator and click OK. Notice that the text instantly appears in the Design view on the tab. Click first on jPanel2 in the Inspector window, then in the Properties window, to change that Tab Title to Images.  To see the code generated for your application so far, click on Source from the middle pane in the menu, as shown in Figure 10. The Source Editor reveals guarded code, which by default is shaded in blue background, as well as code that you can add to or change. As you look through the code, you may notice that you can collapse or expand trees of code as you add to this file.   | | Figure 12: Guarded Generated Code | | --- | | Figure 12: Guarded Generated Code | | | --- | --- | --- |     You may have observed that a layout manager also has been added just beneath the JPanel you added. You can easily change the layout manager. Click on jPanel1, and select Set Layout Manager to GridBagLayout.   | What's a Layout Manager? A layout manager is an object that implements the LayoutManager interface and determines the size and position of the components within a container. Although components can provide size and alignment hints, a container's layout manager has the final say on the size and position of the components within the container. See [A Visual Guide to Layout Managers](http://java.sun.com/docs/books/tutorial/uiswing/layout/visual.html). |  | | --- | --- |     Return to the Design view, and you should now have an application that reveals two tabs with the titles that you input in the Properties box for the JPanel object. For this application, you'll need to add eight labels, two text fields, and a button. The labels contain information for what the user should enter and for what will appear when the user clicks the button. The text fields are editable for user input. The components need to be arranged on this panel in a way that makes sense. The usual layout managers are available to you. For this arrangement, [GridBagLayout](http://java.sun.com/j2se/1.5.0/docs/api/index.html) is ideal. The GridBagLayout manager allows you to arrange components in a number of ways, and the new customizer makes this really easy to do.  To add the components, first make sure you have selected jPanel1 in the Inspector pane. Add the components to the panel by holding down the Shift key and clicking on [JLabel](http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JLabel.html), then on the workspace as many times as you need to label objects. Do the same with [JTextField](http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JTextField.html). Lastly, add a [JButton](http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JButton.html) component. Don't worry about where the components appear on the workspace for now or what variable names they've been given. You are going to work on those in a moment.  Once you've added the components, you'll need to arrange them on the panel. Right-click the GridBagLayout object in the Inspector pane and select Customize. The Customizer window pops up.  Within the GridBagLayout Customizer, you can drag and drop the representation of your components in the window on the right and arrange them in the order you want. In the left side of the Customizer, use the buttons to anchor objects, add padding and fill, and so forth. You can also change the field entries in the upper part of that area to make the same changes if you know what to enter. Play around with the Customizer until the components are aligned the way you want them. Once you have arranged the components the way you want them, click Close. Now you can change the names of some of those variables to more appropriate titles. To rename the variables, right-click each in the Inspector pane and select Rename. Type in the new name and click OK.  Once you have laid out the component the way you want, you may want to change the properties of some of those objects and check the properties of the text fields to be sure that the user can enter information in them. In addition, you need to have text appear on the labels so that the user knows what to enter or what to expect to appear after clicking the button. In the Properties pane for each label, enter the text for each variable as follows:   | Variable | Text | | --- | --- | |  |  | | jLabel1 | Hours Worked: | | jLabel2 | Rate per Hour: | | jLabel3 | Regular Pay: | | jLabel4 | Overtime Pay: | | jLabel5 | Total: |  | To edit a component's properties, click on that property in the Inspector window. That component's properties automatically appear in the Properties pane. Scroll through the items, changing the text, changing the background or foreground colors as you like, setting column width, and changing any other properties you feel appropriate. Experiment with various properties, changing back to the default whenever you need to. Once the components are exactly as you want them, move on to the next step.  Because the user will enter information into the two text fields, it would be better to give those objects variable names that reflect the information being entered. Go to the Inspector window, click on the first JTextField object, right-click, and select Rename. Type in the variable name hoursWorked. Then rename the second JTextField object rate.  Refactoring | | --- |  |  | | --- |   The application calculates the amount of regular pay, overtime pay, and the total pay the user is to receive and writes these amounts to JLabel objects. Rename those variables appropriately as well, such as overtimepayField, regularpayField, and finalTotal.  The refactoring tool goes to work and makes all the necessary changes based on your single entry. Refactoring is the restructuring of code, using small transformations, where the result does not change any program behavior. Just as you factor an expression to make it easier to understand or modify, you refactor code to make it easier to read, simpler to understand, and faster to update. A refactored expression must produce the same result, and the refactored program must be functionally equivalent with the original source.  Some common reasons for refactoring code include the following:  To make the code easier to change or easier to add a new feature  To reduce complexity for better understanding  To remove unnecessary repetition  To enable use of the code for other needs or more general needs  NetBeans provides the following features for refactoring:  Finds where classes, variables, and methods are used throughout the source code in your project.  Enables you to move a class to another package or into another class and enables you to move a static field or a static method from one class to another. In addition, it updates all source code in your project to reference the element in its new location.  Enables you to change the name of a class, variable, or method to something more meaningful. In addition, it updates all source code in your project to reference the element by its new name.  Generates a getter method and a setter method for a field and optionally updates all referencing code to access the field using the getter and setter methods.  Enables you to add parameters to a method and change the access modifier.  Enables you to roll back all the changes in all the files that were affected by the refactoring.  Checking Your GUI   |  | | | --- | --- | | | Figure 13: Test Form Button | | --- | | Figure 13: Test Form Button | | |        | | Figure 14: Build Menu | | --- | | Figure 14: Build Menu | | | --- | --- | --- |   When writing code in a simple text editor, you would have to compile the code frequently to see what your GUI looks like. NetBeans has a preview feature that allows you to see how your application looks with a simple click of the Test Form button located on the center menu as shown in Figure 13. Click the Test Form button, and a test application pops up, showing your creation. You can see from this test whether you need to readjust any of the objects on the panel by going back to GridBagLayout Customizer.  If you would like to compile and run the application, that also is a simple process. Go to the Build menu (see Figure 14) at the top of the IDE and select Clean and Build Main Project. Notice the output at the bottom window. Once the build is complete, you can run the application by returning to the Source view or clicking in the Inspector window on SamplesJFrame, then going to the Run menu and selecting Run File. Select Run SamplesJFrame. The application will run and pop up on the screen. A quick way to compile is to press F9.    Debugging   |  | | --- |   You may have had errors show up in the output window at the bottom of the screen from typos in your code. In an application this small, errors tend to be fairly easy to troubleshoot. In longer programs, though, you may want to use some of the debugging features that NetBeans provides. One such feature is called breakpoints.  Within the NetBeans IDE, you debug by setting breakpoints and watches in your code and running it in the debugger. You can execute your code one line at a time and examine the state of your application in order to discover any problems. When you start a debugging session, all of the relevant debugger windows appear automatically at the bottom of your screen. You can debug an entire project, any executable class, and any JUnit tests. The IDE also lets you debug applications that are running on a remote machine by attaching the debugger to the application process.  A breakpoint is a flag in the source code that tells the debugger to stop execution of the program. When your program stops on a breakpoint, you can perform actions like examining the value of variables and single-stepping through your program. The Source Editor indicates a breakpoint by highlighting the line in red and placing an annotation in the left margin.  Except for line breakpoints, all Java breakpoints are defined globally for all IDE projects. For example, if you set a breakpoint on a class, the IDE will stop execution every time it encounters that class during a debugging session regardless of what project you are debugging.  To set a line breakpoint, click the left margin of the line in the Source Editor or press Ctrl-F8. You can set other types of breakpoints as well, such as the following:  Method -- Program execution stops every time the method is executed.  Exception -- You can break whenever a specific exception is caught, whenever a specific exception is not handled in the source code or whenever any exception is encountered regardless of whether the program handles the exception or not.  Variable -- You can stop execution of your program whenever a variable in a specific class is accessed (for example, the method was called with the variable as an argument) or modified.  Thread -- You can break program execution whenever a thread starts, stops, or both.  Class -- You can break when the class is loaded into the virtual machine, unloaded from the virtual machine, or both.  This sample application is small for taking advantage of these rich features, but as you add to this application, you should consider experimenting with breakpoints for debugging. You can set a few breakpoints at the method you are going to write soon to see how they work. For more details on the NetBeans debugging tool, see For More Information at the end of this tutorial.  Adding Menus and Images   |  | | --- |   So far, this application is a simple GUI and no more. It doesn't yet have functionality. You will add an event handler for the button soon, but first create a top menu for the application.  Though menus are often one of the first objects created in an application, you're creating the menuing system now to demonstrate how easy it is to move around within your application, creating in whatever order you need, without messing up code. This is one of the big advantages of the Inspector window: the way you can click on any object within your application, add to it, or move around within it.  Return to Design view, go to the Inspector window, and select the JFrame listed. Notice in the Properties window that you can change the properties listed. For a title, type in the pop-up box Pay Calculator. Change background colors if you like.  Next, in the Inspector pane, right-click the JFrame listed , select Add From Palette, then select Swing and [JMenuBar](http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JMenuBar.html). When you see that the object has been added to the Inspector box, right-click the JMenuBar, then select Add JMenu. Right-click the JMenu that is added and rename it FileMenu. Now right-click FileMenu and select Add JMenuItem. Rename the item OpenMenu. Add as many items as you like. If you want to add another menu, right-click JMenuBar again, select Add JMenu, and rename it HelpMenu. In the Design view, you should see menus appear. Click Test Form if you'd like a more realistic view.  The last item to complete in the GUI portion of this application is the Images pane (see Figure 15), so that you know how to add images easily. One way to display images is by decorating Swing components, such as labels, buttons, and tabbed panes, with an icon -- a fixed-sized picture. An icon is an object that adheres to the [Icon](http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/Icon.html) interface. Swing provides a particularly useful implementation of the Icon interface: [ImageIcon](http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/ImageIcon.html), which paints an icon from a GIF, JPEG, or (as of version 1.3) PNG image.  One statement creates the image icon, and two more statements include the image icon on each of the two labels:   | ImageIcon icon = createImageIcon("images/middle.gif",  "The name of an image.");  label1 = new JLabel("Image and Text", icon, JLabel.CENTER);  ...  label2 = new JLabel(icon); | | --- |      | | Figure 15: Images Pane | | --- | | Figure 15: Images Pane | | | --- | --- | --- |     NetBeans allows you to display images in similar fashion, but you can do it through the Palette and Properties windows. You don't have to write the code for it. For the Images pane, you'll use a JLabel, and change the properties of that object to display an image.  Begin either by clicking on the Images tab in the Design view or by clicking on the JPanel object for this pane in the Inspector pane. Set the layout to BorderLayout. In the Palette pane, click on JLabel, and click in the west side of the layout. This label is just a spacer. To be sure the JLabel is where you want it, check the label's Direction property. You can move it anywhere from within the Properties pane. Set the preferredSize of this label to 70, 14, and remove any text for the label.  Next, create another label and place it in the center of the BorderLayout. In the properties for this label, click on the icon button. A window appears that allows you to enter information about image type. Make sure the drop-down menu is on Select Mode: Icon Editor. Notice the many Image Source Types you can choose from (see Figure 16). For this example, select URL and type in the following url into the Name text area:  http://java.sun.com/developer/onlineTraining/new2java/newjava.gif  or select File and type in the path to the file on your system in the Name text area.  Note: If you're on a system that is behind a firewall, you'll have to have your system HTTP proxy settings correct before NetBeans will be able to display the preview of this .gif file (when using the URL method).  You should see an image appear in the preview window. Click OK. If you return to Source view and compile the application, you will see the following lines of code, including [exception handling statements](http://java.sun.com/docs/books/tutorial/java/nutsandbolts/exception.html):   | jLabel6.setIcon(new javax.swing.JLabel() {  public javax.swing.Icon getIcon() {  try {  return new javax.swing.ImageIcon(  new java.net.URL(  "http://java.sun.com/developer/onlineTraining/new2java/newjava.gif")  );  } catch (java.net.MalformedURLException e) {  }  return null;  }  }.getIcon());  jLabel6.setToolTipText("This is a label with an image"); | | --- |     Now the GUI is complete, but the application still doesn't do anything. Next, you need to give it functionality. Click back into Design view and then on to the Pay Calculator pane.   | | Figure 16: The Icon Property Window | | --- | | Figure 16: The Icon Property Window | | | --- | --- | --- |   Adding Event Handlers   |  | | --- |   Java technology programs rely on events that describe user actions. These events are represented by objects that a user initiates, such as text entered, a button pushed, or a mouse moved over a component. The component must listen for an event if you want to handle the event with an action, such as displaying text or writing to a database. Without an IDE, you'd have to follow these steps to create event handlers:  Decide which type of event is fired by a component and implement the right interface  Register the component as a listener for that type of event  Implement handler methods from the interface   | | Figure 17: How Events Work | | --- | | Figure 17: How Events Work | | | --- | --- | --- |   When a user fires an event through a GUI component, a method call is made to any objects that have been registered with the component as listeners for the type of event that took place. An event listener is an object that is notified when an event occurs. Events generated from buttons and text fields need the class to implement the ActionListener interface. When implementing this interface, you must also implement a listener method.  NetBeans simplifies creating event handlers by creating much of the code for you. You need to know which interface you'll need to implement, and then you'll need to write the code for the logic that goes into the listener method.  Go to Design view and click the button you created, then right-click. Select Events, then Action -> actionPerformed(computePayActionPerformed).  In Source view, you'll see that the following lines of code have been added to register the button with a listener:   | computePay.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  computePayActionPerformed(evt);  }  }); | | --- |     NetBeans has also added the necessary method:   | private void computePayActionPerformed(java.awt.event.ActionEvent evt) {  // TODO add your handling code here: | | --- |     Now, add code where indicated to provide the logic of what the application needs to do to compute the amount of regular pay and overtime pay. First, you'll need set up a few variables for the following:  Hours Worked  Pay Rate  Later, you'll also need variables for  Regular Pay  Overtime Pay  Because the user is entering the information into the text field, that data comes in as a String. You'll need to parse these strings into workable types, such as int or double. You'll then need to perform the calculations to determine how many hours are regular and how many are overtime. Once those amounts are determined, you can calculate regular pay and overtime pay, then total those two amounts. Lastly, you'll need to convert the numbers to dollar amounts.  The classes you'll need to be familiar with are [Integer](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Integer.html), [Double](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Double.html), and [NumberFormat](http://java.sun.com/j2se/1.5.0/docs/api/java/text/NumberFormat.html).  A common need is to convert strings to numerical values. Once these are converted, you can manipulate that value like any other data primitive (for instance, an int, float, or double). The problem is that when you read from a file, get a command line argument, get an applet parameter, or just prompt for input from a text field on a form, you always start with a String object. Treating that String as a primitive requires an extra step, the conversion.  To convert a String you use a wrapper class. Every primitive has an associated wrapper class. These wrapper classes allow you to treat primitives as objects. In addition, these classes contain methods that permit manipulation of these objects, appropriate for the data type.  The following list shows the mapping from primitive type to wrapper class. In almost all cases, the wrapper class has the same name as the primitive data type, with the first letter capitalized:   | byte - Byte  short - Short  int - Integer  long - Long  char - Character  float - Float  double - Double  boolean - Boolean | | --- |     Each of these wrapper classes, except the Character class, has a method that allows you to convert from a String to the specific primitive type. Simply call the method from the appropriate wrapper class, and your String is converted to a primitive type:   | String myString = "12345";  int myInt = Integer.parseInt(myString); | | --- |     converts the contents of the String variable myString to an int named myInt. The conversion is that easy. The only trick is that the conversion for each data type involves a uniquely named method in each wrapper class. All but the Boolean conversions are done by similarly named methods, but still all the method names are different:   | byte - Byte.parseByte(aString)  short - Short.parseShort(aString)  int - Integer.parseInt(aString)  long - Long.parseLong(aString)  float - Float.parseFloat(aString)  double - Double.parseDouble(aString)  boolean - Boolean.valueOf(aString).booleanValue(); | | --- |     There is one exception: The Character class has no such method, so you have to ask String for the character with the charAt method:   | // The variable e is the character e in the string Hello  String hello = "Hello";  char e = hello.charAt(1); | | --- |     If the String contents cannot be converted to the requested primitive type, then a NumberFormatException is thrown. This is a runtime exception, so the conversion code does not have to be in a try-catch block.  The [NumberFormat](http://java.sun.com/j2se/1.5.0/docs/api/java/text/NumberFormat.html) class is used to print the numbers correctly. Printing numbers to the screen can produce some odd or undesired results.  For instance,   | public class PrintNum {  public static void main(String args[]) {  double amount = 400000.00;  System.out.println("He paid " + amount  + " for the house.");  }  } | | --- |     results in   | He paid 400000.0 for the house. | | --- |     You can control display format and arrange input into the needed output format by using the NumberFormat abstract class in the java.text package. This class provides the interface to format and parse numbers, and it includes methods to determine which locales have number formats and what their names are.  The class has methods for three standard formats:  getInstance or getNumberInstance gets a format for the normal number format, such as 600,000.  getCurrencyInstance gets a format for the currency number format, such as $600,000.00.  getPercentInstance gets a format for displaying percentages, such as 56%.  To format a primitive, start by returning an object of type NumberFormat by calling on one of the above methods.   | NumberFormat nf = NumberFormat.getCurrencyInstance(); | | --- |     To be certain the amount is formatting correctly for a specific country, you specify the locale this way:   | NumberFormat nf =  NumberFormat.getCurrencyInstance(Locale.US); | | --- |     The result is   | He paid $400,000.00 for the house. | | --- |     To compile these, you'll need to import the following packages and add them to the top of the file:   | import java.text.NumberFormat;  import java.util.Locale; | | --- |     Read the comments in the code, which gives some explanation. If you are new to using if-statements, read [if/else and switch statements](http://java.sun.com/developer/onlineTraining/new2java/supplements/2001/sept01.html#basics), which is a part of the [Java Technology Fundamentals](https://softwarereg.sun.com/registration/developer/en_US/subscriptions) newsletter.  Add the following code to your computePayActionPerformed() event handler method:   | //Gets how many hours worked from the text field, and then parses it to type int.  int hw = Integer.parseInt(hoursWorked.getText());  //Gets the pay rate entered in the text field, and parses it to type double.  double rateEntered = Double.parseDouble(rate.getText());  //Creates two variables of type double for later use.  double regularPay = 0;  double overTimePay = 0;  //Simple if statement for calculations  if (hw > 40) {  regularPay = 40 \* rateEntered;  overTimePay = (hw - 40) \* 1.5 \* rateEntered;  } else {  regularPay = hw \* rateEntered;  overTimePay = 0.0;  }  //Creates a variable of both types of pay combined.  double total = regularPay + overTimePay;  //Creates variables for number formatting  NumberFormat nfRegular = NumberFormat.getCurrencyInstance(Locale.US);  NumberFormat nfOverTime = NumberFormat.getCurrencyInstance(Locale.US);  NumberFormat nfTotal = NumberFormat.getCurrencyInstance(Locale.US);  //Writes the totals in the correct format to the labels  regularpayField.setText(" " + nfRegular.format(regularPay));  overtimepayField.setText(" " + nfOverTime.format(overTimePay));  finalTotal.setText(" " + nfTotal.format(total));  } | | --- |     Compile and run the application. Test it out by entering hours worked and pay rate, such as 33.00, and so forth. Your application should look something like Figure 18. The blue type was created simply by editing the properties of that label so that the foreground appears blue.   | | Figure 18: Final Application | | --- | | Figure 18: Final Application | | | --- | --- | --- |     The menus are still nonfunctional as well, and you'd follow the same procedure to add events to your menu items. In the Inspector or Project window, select the menu ExitItem, right-click, select Events, choose Action -> actionPerformed. You'll be sent to the Source Editor, where you can enter the following code:   | // TODO add your handling code here:  System.exit(1); | | --- |     Select Clean and Build Project, then run your application, closing it with the Exit menu. Follow the same procedure for each menu item and any other objects that require functionality.  If there are any objects you don't want in your application, select the object in the Inspector window. Right-click and select Delete. You'll be asked if you're sure that you want to delete the item. Say yes, and the item will disappear from view. Build and Clean, then run your application. Try this by deleting OpenItem from the menu. You'll see that cleanup is an easy process.  Documenting Your Work   |  | | --- |   Documenting your application is always a good idea, especially if many people are working on one project. Again, this sample application is a bit small for a good demonstration of the Javadoc tool feature, but you can get a good idea of the kind of information that goes into the documentation and see how NetBeans does all the hard work for you.  Click on the Files tab, then on your SamplesJFrame. Go to the Build Menu and select Generate Javadoc for JavaApplication1 (or whatever name you gave your project). You'll see that the Javadoc is being generated in the output window. Then a browser window pops up with the documentation formatted in HTML. Click on SamplesJFrame, and scroll down to see the documentation. This application will not require much documentation, but if you continue building on this application or move on to creating larger project, Javadoc will come in handy.  Summary   |  | | --- |   You have completed a small desktop application and learned some of the basics of the NetBeans IDE. A big advantage of using the NetBeans IDE platform is that you won't outgrow it. As your development knowledge and skills build, you can delve into other Java technologies and platforms, such as web development using Java Platform, Enterprise Edition or Java Platform, Micro Edition. In addition, NetBeans is extensible, meaning that many of the extensions, or modules, that work in NetBeans also work seamlessly within other products, such as Sun Java Studio Creator and Sun Java Studio Enterprise. The IDE evolves with your needs and experience, and you build on the experience you gain using a common IDE platform, rather than having to learn an entirely new toolset as your needs change.  For More Information   |  | | --- |   [NetBeans.org](http://netbeans.org)      Top of Form     | | Rate and Review | | --- | | Tell us what you think of the content of this page. | | Excellent   Good   Fair   Poor | | Comments: | | Your email address (no reply is possible without an address): [Sun Privacy Policy](http://sun.com/privacy/)  Note: We are not able to respond to all submitted comments. | |  | | | --- | --- | --- | --- | --- | --- | --- |   Bottom of Form |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

|  |
| --- |

copyright © Sun Microsystems, Inc

|  |
| --- |

