# Introduction to Developing General Java Applications

The following short tutorial takes you through some of the basic steps of developing a Java SE application in [NetBeans IDE](http://testwww.netbeans.org/products/ide/index.html) 5.5. We will create a MyLib project with a utility class, then create a MyApp project with a main class that implements a method from the library project.

## Setting Up a Project

First we'll create a new Java class library to contain the utility classes we'll need later. Then we need to create a new Java application to use as our main project. Once the MyApp project has been created, we'll add MyLib's classes to its classpath.

### Creating a New Java Class Library

1. Choose File > New Project (Ctrl-Shift-N). Under Categories, select General. Under Projects, select Java Class Library and click Next.
2. Under Project Name, enter MyLib. Change the Project Location to any directory on your computer. From now on, we will refer to this directory as *NetBeans\_projects*.

**Note:** The path specified above should appear as follows in the Project Folder field of the wizard: /*NetBeans\_projects*/MyLib/

1. Click Finish. The MyLib project opens in both the Projects window and the Files window.

### Creating a New Java Application

1. Choose File > New Project. Under Categories, select General. Under Projects, select Java Application and click Next.
2. Under Project Name, enter MyApp. Make sure the Project Location is set to *NetBeans\_projects*.
3. Enter acrostic.Main as the main class.
4. Ensure that the Set as Main Project and Create Main Class checkboxes are checked.
5. Click Finish. The MyApp project is displayed in the Project window and Main.java opens in the Source Editor.

### Configuring the Compilation Classpath

1. In the Projects window, right-click the Libraries node for the MyApp project and choose Add Project.
2. Browse to *NetBeans\_projects*/ and select the MyLib project folder. The Project JAR Files pane shows the JAR files that can be added to the project. Notice that a JAR file for MyLib is listed even though we have not actually built the JAR file yet. This JAR file will get built when we build and run the MyApp project.
3. Click Add Project JAR Files.
4. Expand the Libraries node. Note that the MyLib project's JAR file is added to the MyApp project's classpath.

## Creating and Editing Java Source Code

Now we need to create a Java package and add the method that we'll use to construct our acrostic, after which we'll implement the acrostic method in the Main class.

### Creating a Java Package and Class File

1. Right-click the MyLib project node and choose New > Java Class. Type LibClass as the name for the new class, type org.me.mylib in the Package field, and click Finish. LibClass.java opens in the Source Editor.
2. In LibClass.java, place the cursor on the line between the class declaration (public class LibClass {) and the constructor (public LibClass() {). Type or paste in the following method code:
3. public static String acrostic(String[] args) {
4. StringBuffer b = new StringBuffer();
5. for (int i = 0; i < args.length; i++) {
6. if (args[i].length() > i) {
7. b.append(args[i].charAt(i));
8. } else {
9. b.append('?');
10. }
11. }
12. return b.toString();

}

1. If the code you pasted in is not formatted correctly, press Ctrl-Shift-F to reformat the entire file.
2. Press Ctrl-S to save the file.

### Editing a Java File

Now we will add some code to Main.java. In doing so, we will demonstrate the Source Editor's code completion and code template (abbreviation) features.

1. Select the Main.java tab in the Source Editor. If it isn't already open, expand MyApp > Source Packages > acrostic in the Projects window and double-click Main.java.
2. Delete the // TODO code application logic here comment in the main method and type the following:

String result = Li

1. Press Ctrl-Space to open the code completion box. The IDE offers code completion for all classes and methods in the project's compilation classpath. Select LibClass (org.me.mylib) and press Enter. The IDE fills in the rest of the class name and also automatically creates an import statement for the class.

**Note:** The IDE also opens a box above the code completion box that displays Javadoc information for the selected class or package. Since there is no Javadoc information for most packages, the box displays a "Cannot find Javadoc message" until you reach a Java class.

1. In the main method, enter a period after LibClass. The code completion box opens again. Select the acrostic(String[]args) method and press Enter. The IDE fills in the acrostic method and shows the method's parameters in a tip.
2. Press Ctrl-Space again to open the code completion box and select args, then press Enter.
3. Type a semicolon. Note that it gets entered after the closing parenthesis even though the insertion point was still inside the parenthesis. The final line should look like this:

String result = LibClass.acrostic(args);

1. Press Enter to start a new line. Then type sout and press space. The sout abbreviation expands to System.out.println(""); with the cursor positioned between the quotation marks. Type Result = inside the quotation marks and + result after the end quotation mark. The final line should look like this:

System.out.println("Result = " + result);

1. Press Ctrl-S to save the file.

## Compiling and Running a Project

Now we need to set the main class and execution arguments so we can run our project. We'll also take a look at the IDE's cleaning, building, and Javadoc generation features.

### Setting the Main Class and Execution Arguments

1. Right-click the MyApp project node, choose Properties, and select the Run node in the dialog's left pane. Notice that the main class is already set to acrostic.Main.
2. Enter However we all feel zealous in the Arguments field and click OK.

### Running the Main Project

1. Choose Run > Run Main Project (F6) from the Run menu. Double-click the Output window to maximize it so you can see all the output. Note that Ant builds MyLib.jar first and then compiles MyApp using it. Finally, it prints the output from the program, Result = Hello (the acrostic of the phrase that was passed to the program as an argument).
2. Select the Files window and expand the MyApp project folder. The built class files are in the build folder.
3. Press F6 to run the program again. Nothing new needs to be compiled, but the program is run.

### Cleaning and Building a Project

1. Choose Build > Clean and Build Main Project (Shift-F11). Both the MyLib project and the MyApp project are cleaned and rebuilt as part of the process.
2. Right-click the MyLib project node in the Projects window and choose Clean Project. Just the MyLib project is cleaned.

### Building an Individual Project

1. Right-click the MyLib project node in the Projects window.
2. Choose Build Project from the contextual menu.

### Generating Javadoc

1. Select the MyLib project.
2. Choose Build > Generate Javadoc for "MyLib" from the IDE's main menu.

The IDE displays Javadoc output in the Output window, and your web browser opens displaying the Javadoc.

## Testing and Debugging a Project

Now we'll create and run a test for our project using JUnit and then run it in the IDE's debugger to check for errors.

### Creating JUnit Tests

1. Right-click the LibClass.java node in the Projects window and choose Tools >Create JUnit Tests (Ctrl-Shift-U). Click OK to run the command with the default options. The IDE creates the org.me.mylib package and the LibClassTest.java file in a separate test folder. You can find this file by expanding the Test Packages node and the org.me.mylib subnode.
2. In LibClassTest.java, delete the body of the testAcrostic method and type or paste in the following:
3. System.err.println("Running testAcrostic...");
4. String result = LibClass.acrostic(new String[]
5. {"fnord", "polly", "tropism"});

assertEquals("Correct value", "foo", result);

1. Save the file by pressing Ctrl-S.

### Running JUnit Tests

1. Select the MyLib project node and choose Run > Test "MyLib" (Alt-F6). The MyLib (test) tab opens in the Output window. The JUnit test cases are compiled and run. The JUnit test result shows that the test passes.
2. You can also run a single test file rather than testing the entire project. Select the LibClass.java tab in the Source Editor and choose Run > Run File  > Test "LibClass.java" from the Run menu.

### Debugging a Project

1. In the LibClass.java file, go to the acrostic method and place the insertion point anywhere inside b.append(args[i].charAt(i));. Then press Ctrl-F8 to set a breakpoint.
2. Choose Run > Debug Main Project (F5). The IDE opens the Debugger windows and runs the project in the debugger until the breakpoint is reached.
3. Select the Local Variables window and expand the args node. The array of strings contains the phrase you entered as the command arguments.
4. Click Step Into (F7) in the toolbar to step through the program and watch the acrostic being constructed.

When the program reaches the end, the debugger windows close.

[*Send Us Your Feedback*](http://testwww.netbeans.org/about/contact_form.html?to=3&subject=Feedback:%20Introduction%20to%20Developing%20General%20Java%20Applications)

## Next Steps

For more information about using NetBeans IDE 5.5 to develop Java SE applications, see the following resources:

* [Importing Existing Java Source Code into NetBeans IDE 5.5](http://testwww.netbeans.org/kb/55/import_j2se.html)
* [Using NetBeans IDE 5.5](http://testwww.netbeans.org/kb/55/using-netbeans/index.html)
* [NetBeans IDE Support and Docs page](http://testwww.netbeans.org/kb/index.html)