

2 Getting Started

For the examples in this section, Microsoft Windows and Java will be used. However, much of the information applies to other operating systems and supported languages for which you have installed a compiler (e.g., Ada, C, C++, Java) unless noted otherwise. In any of the language specific steps below, simply select the appropriate language and source code. For example, in the “Creating a New File” below, you may select C++ as the language instead of Java, and then enter a C++ example. If you have installed jGRASP on your personal computer, you should see the jGRASP icon on the Windows desktop.

Objectives – When you have completed this tutorial, you should be comfortable with editing, compiling, and running Java programs in jGRASP. In addition, you should be familiar with the pedagogical features provided by the Control Structure Diagram (CSD) window, including using interactions, generating the CSD, folding your source code, numbering the lines, and stepping through the program in the integrated debugger.

The details of these objectives are captured in the hyperlinked topics listed below.

- 2.1 Starting jGRASP
- 2.2 Quick Start - Opening a Program, Compiling, and Running
- 2.3 Creating a New File
- 2.4 Saving a File
- 2.5 Building Java Programs - - Recap
- 2.6 Interactions (Java only)
- 2.7 Generating a Control Structure Diagram
- 2.8 Folding a CSD
- 2.9 Line Numbers
- 2.10 Compiling a Program – A Few More Details
- 2.11 Running a Program - Additional Options
- 2.12 Using the Debugger (Java only)
- 2.13 Opening a File – Additional Options
- 2.14 Closing a File
- 2.15 Exiting jGRASP
- 2.16 Review and Preview of What’s Ahead
- 2.17 Exercises

2.1 Starting jGRASP



jGRASP

If you are working in a Microsoft Windows environment, you can start jGRASP by double clicking its icon on your Windows desktop.

If you don't see the jGRASP icon on the desktop, try the following: click *Start > All Programs > jGRASP (folder) > jGRASP*.

Depending on the speed of your computer, jGRASP may take between 10 and 30 seconds to start up. The jGRASP virtual **Desktop**, shown below, is composed of a Control Panel with a menu and toolbar across the top and three resizable panes. The *left pane* has tabs for **Browse**, **Debug**, **Find**, and **Workbench** (Project tab is combined with the Browse tab beginning in version 1.7). The Browse tab, which is the default when jGRASP is started, lists the files in the current directory. The large *right pane* is for UML and CSD Windows. The *lower pane* has tabs for jGRASP messages, Compile messages, Run I/O, and Interactions. The panes can be resized by moving the horizontal or vertical partitions that separate them. Select the partition with the mouse (left-click and hold down) then drag the partition to make a pane larger or

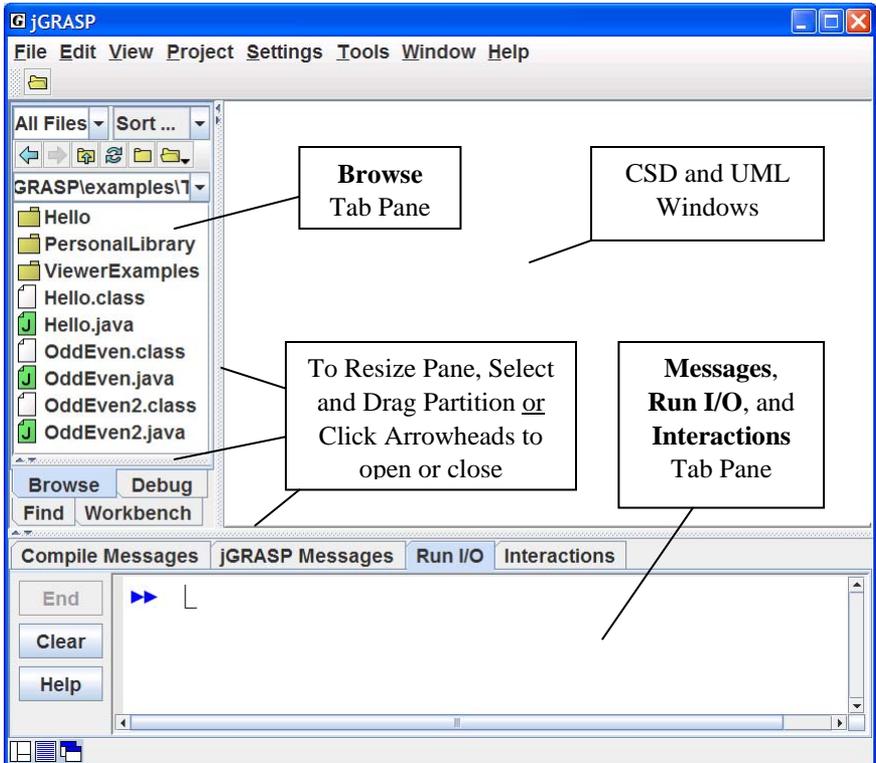


Figure 2-1. The jGRASP Virtual Desktop

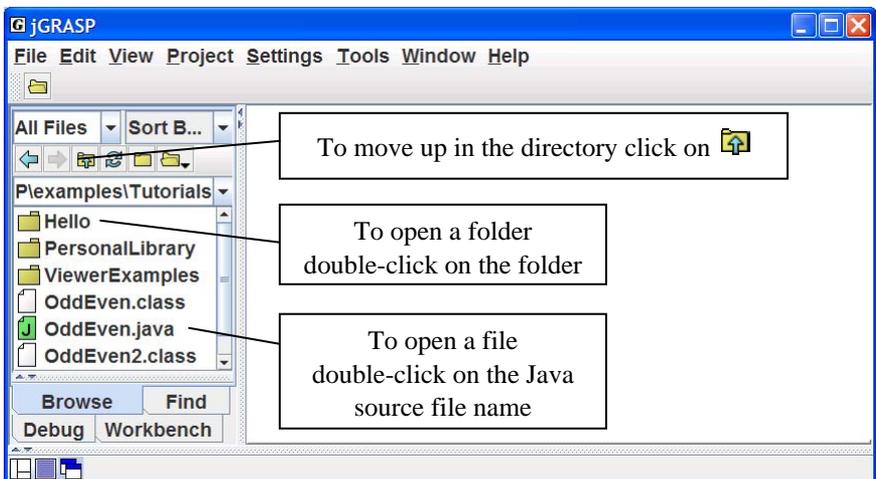
smaller. You can also click the arrowheads on the partition to open and close the pane.

2.2 Quick Start - Opening a Program, Compiling, and Running

Example programs are available in the jGRASP folder in the directory where it was installed (e.g., c:\Program Files\jgrasp\examples\Tutorials). You should copy the tutorial folder to one of your personal folders (e.g., in your *My Documents* folder) so that any changes you make will not be lost when a new version of jGRASP is installed.

Note: If you already have example programs with which you are familiar, you may prefer to use them rather than the ones included with jGRASP as you work through this first tutorial.

Clicking the Open File button  on the toolbar pops up the Open File dialog. However, the easiest way to open existing files is to use the **Browse** tab (below). The files shown initially in the Browse tab will most likely be in your home directory. You can navigate to the appropriate directory by double-clicking on a folder in the list of files or by clicking on  as indicated in the figure below. The refresh button  updates the Browse pane. Below, the Browse tab is displaying the contents of the Tutorials folder.



Double-clicking on the Hello folder, then the Hello.java file, as shown in **Step 1** below, opens the program in a CSD window. The CSD window is a full-featured editor for entering and updating your programs. Notice that opening the CSD window places additional buttons on the toolbar. Once you have opened a program or entered a new program (**File > New File > Java**) and saved it, you are ready to compile the program and run it. To compile the program, click on the **Build** menu then select **Compile**. Alternatively, you can click on the Compile button indicated by **Step 2** below. After a successful compilation – no error messages in the Compile Messages tab (the lower pane), you are ready to run the program by clicking on the Run button as shown in **Step 3** below, or you can click the **Build** menu and select **Run**. The standard input and output for your program will be in the Run I/O tab of the Message pane.

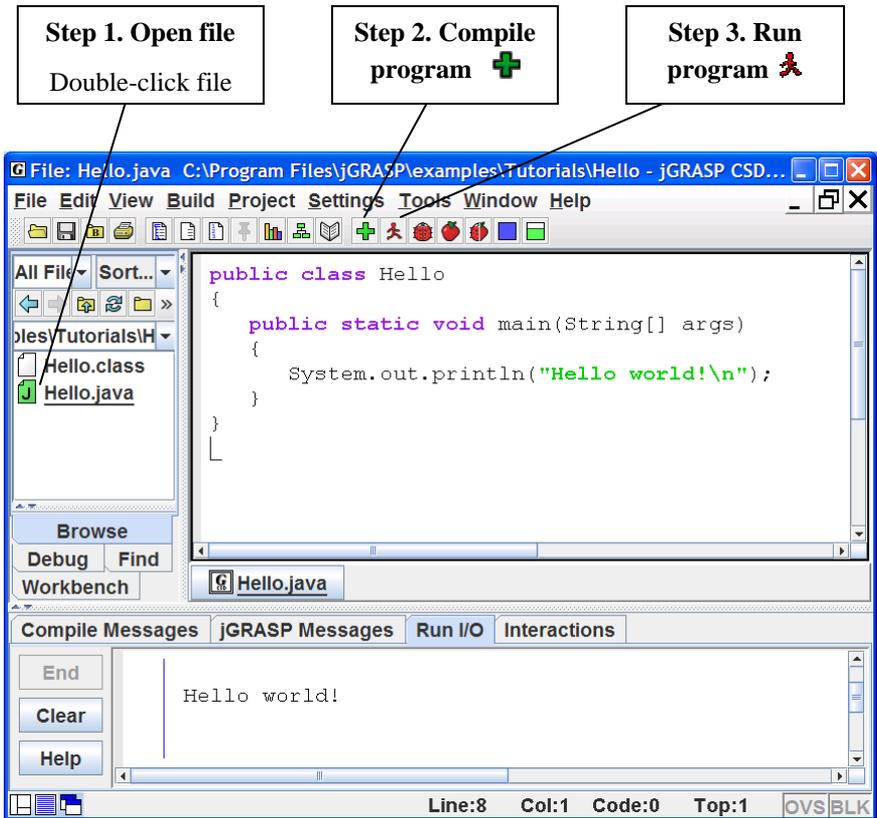


Figure 2-2. After loading file into CSD Window

2.3 Creating a New File

To create a new Java file within the Desktop, click on **File > New File > Java**. Note that the list of languages displayed by **File > New File** will vary with your use of jGRASP. If the language you want is not listed, click **Other** to see the additional available languages. The languages for the last 25 files opened will be displayed in the initial list; the remaining available languages will be under **Other**.

After you click on **File > New File > Java**, a CSD window is opened in the right pane of the Desktop as shown in Figure 2-4 below. Notice the title for the frame, jGRASP CSD (Java), which indicates that the CSD window is Java specific. If Java is not the language you intend to use, you should close the window and then open a CSD window for the correct language. Notice that the *button* for each open file appears below the CSD windows in an area called the windowbar (similar to a taskbar in the Windows OS environment). Later when you have multiple files open, the windowbar will be quite useful for popping a particular window to the top. The buttons can be reordered by dragging them around on the windowbar.

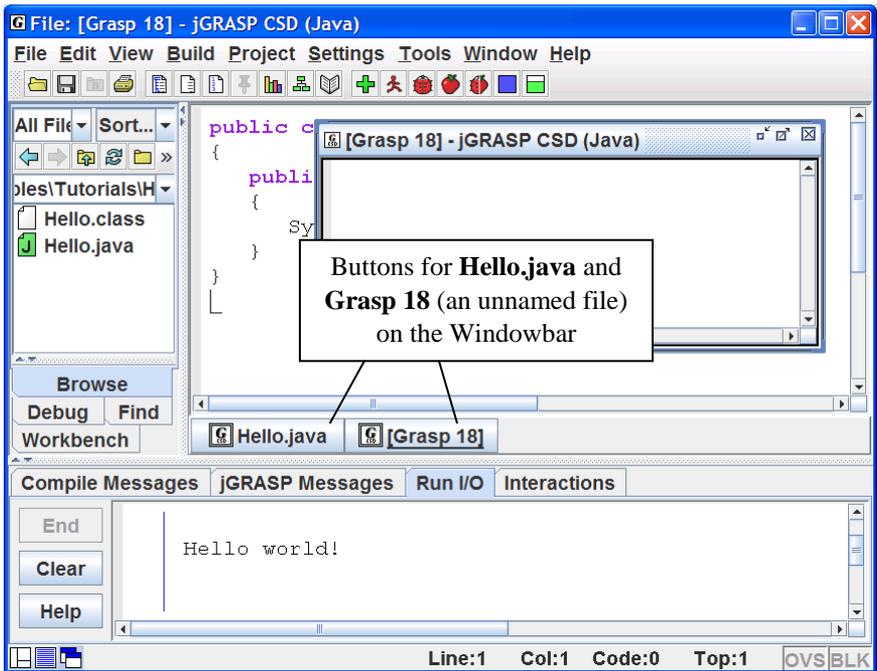


Figure 2-4. After opening a new CSD Window for Java



In the upper right corner of the CSD window are three buttons that control its display. The first button minimizes the CSD window; the second button maximizes the CSD window or if it is already maximized, restores the CSD window to its previous size. The third button closes the CSD window. You may also make the Desktop itself full screen by clicking the appropriate button in the upper corner of it.

Figure 2-5 shows the CSD window maximized within the virtual Desktop. The “L” shaped cursor in the upper left corner of the empty window indicates where text will be entered.

TIP: If you want all of your CSD windows to be maximized automatically when you open them, click **Settings > Desktop**, and then click **Open Desktop Windows Maximized** (a check mark indicates that this option is turned ON).

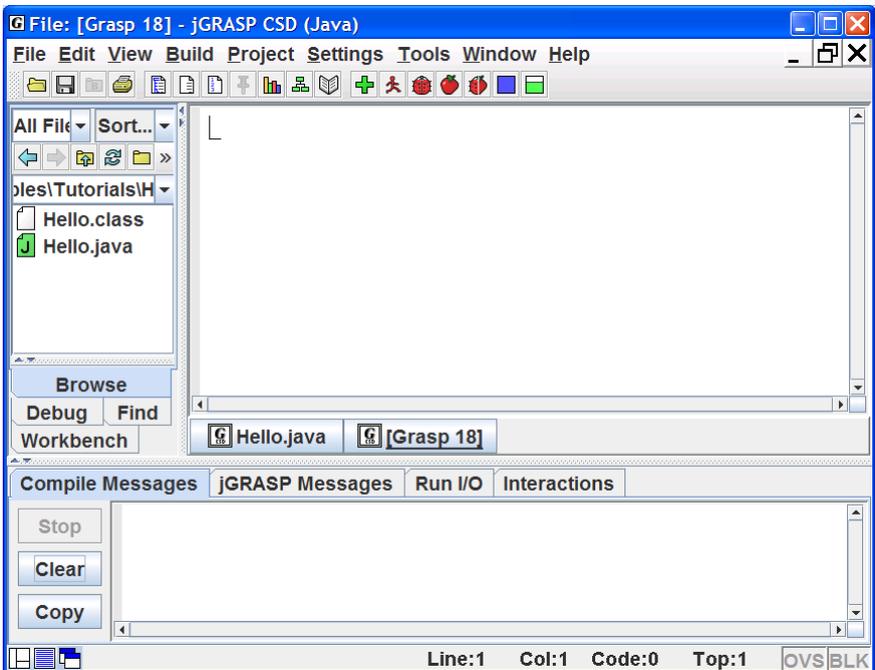


Figure 2-5. CSD Window maximized in Desktop

Type the following Java program in the CSD window, exactly as it appears. Remember, Java is case sensitive. Alternatively, you may copy/paste the Hello program into this window, then change the class name to Hello2 and add the “Welcome...” line.

```
public class Hello2
{
    public static void main(String[] args)
    {
        System.out.println ("Hello world!");
        System.out.println ("Welcome to jGRASP!");
    }
}
```

After you have entered the program, your CSD window should look similar to the program shown in Figure 2-6.

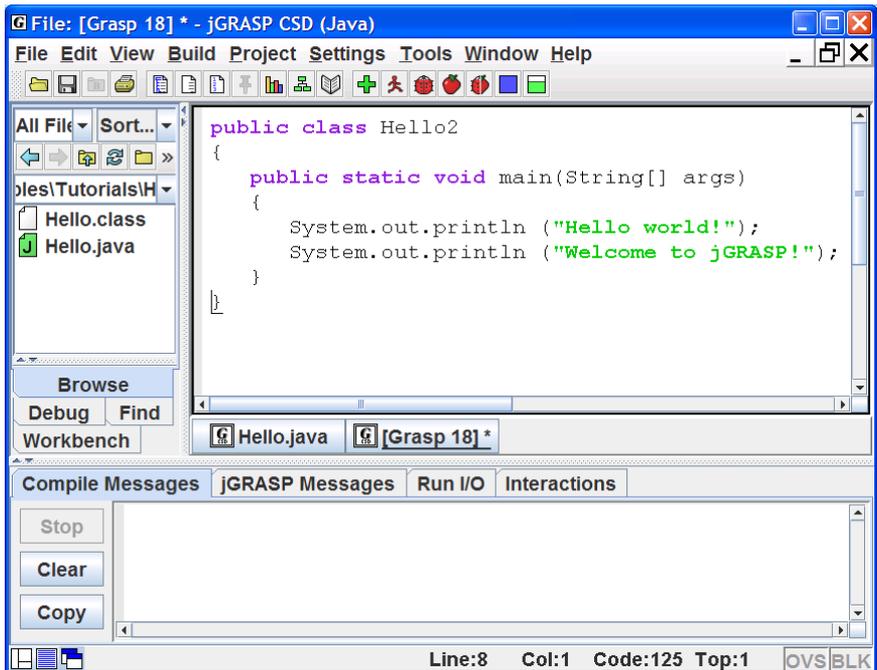


Figure 2-6. CSD Window with program entered

2.4 Saving a File

You can save the program as "Hello2.java" by doing any of the following:

- (1) Click the Save button  on the toolbar, or
- (2) Click **File > Save** on menu (see Figure 2-7), or
- (3) Press Ctrl-S (i.e., while pressing the Ctrl key, press the "s" key).

If the file has not been saved previously, the Save dialog box pops up with the name of the file set to the name of the class file. Note, in Java, the file name must match the class name (i.e., class Hello2 must be saved as Hello2.java). Be sure you are in the correct directory. If you need to create a new directory, click the folder button on the top row of the Save dialog. When you are in the proper directory and have the correct file name indicated, click the *Save* button on the dialog. After your program has been saved, it should be listed in the Browse tab (see Figure 2.8 on the next page). If you do not see the program in the Browse tab, you may need to navigate to the directory where the file was saved or click  on the toolbar to change the **Browse** tab to the directory of the current file.

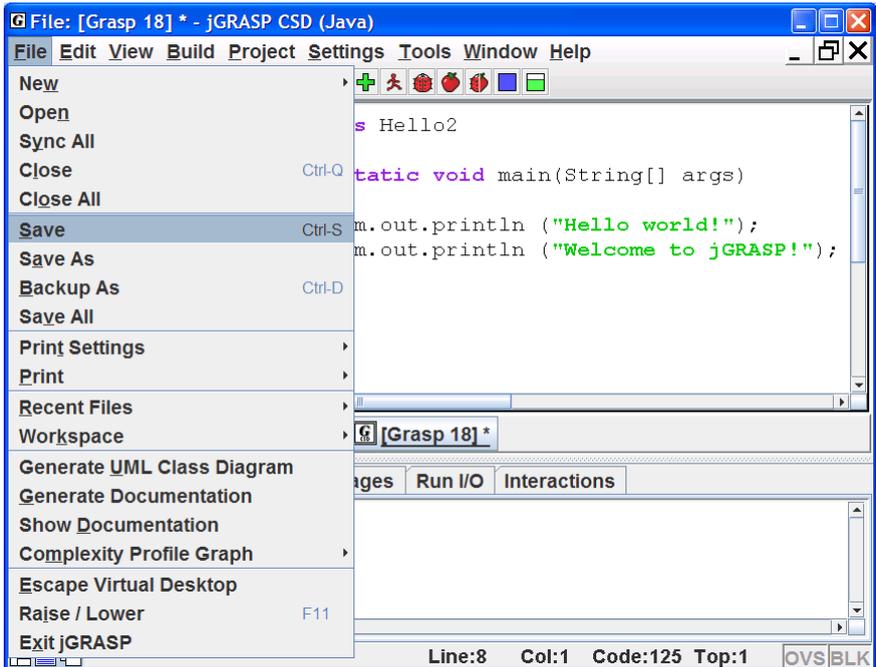


Figure 2-7. Saving a file from the CSD Window

2.5 Building Java Programs - - Recap

As seen in the previous sections, Java programs are written in an edit window, saved, compiled, and run. A somewhat more detailed description of steps for building software in Java is as follows.

(1) Enter your source code into a CSD window and then save the program in a file whose name consists of the Java class name and the “.java” extension (e.g., MyProgram.java). You should try to enter your program in chunks so that it will always be compilable.

(2) **+** Compile the source program (e.g., MyProgram.java) to create the byte code file with a “.class” extension (e.g., MyProgram.class). After attempting to compile your program, you may need to make corrections via the edit window (step 1) based on the messages provided by the compiler and then compile the program again. Note that the .class file is not created until you have a “clean” compile (i.e., no error messages).

(3) **⚠** Run your program. In this step, the byte code or .class file produced by the compiler is executed by the Java Virtual Machine. After you run your program, you should inspect the output (if any) to make sure the program did what you intended. At this point, you may need to find and correct mistakes (bugs). After making the corrections in the edit window (step 1), you will need to compile your program again (step 2). Later, we will use the debugger to step through a program so we can see what happens after each individual statement is executed.

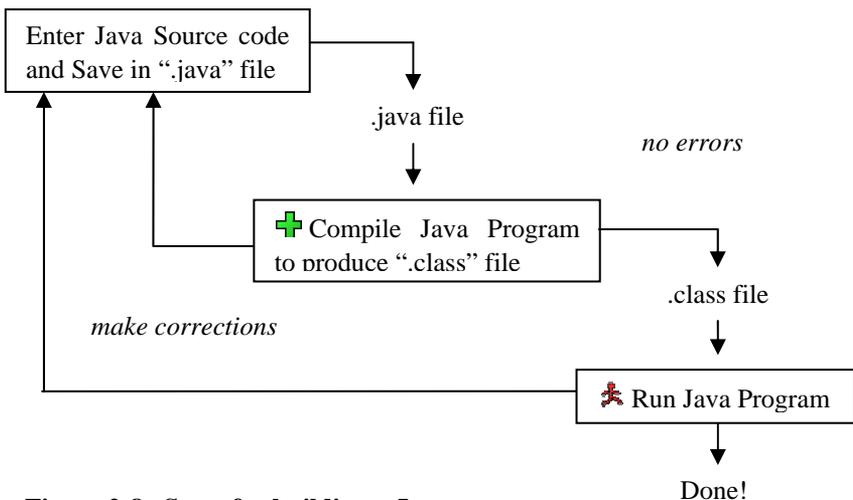


Figure 2-8. Steps for building a Java program

2.6 Interactions (Java only)

While all of your Java programs will be built using the steps described above, or some variation of them, jGRASP provides an Interactions feature which can be very useful along the way. The **Interactions** tab, located next to the **Run I/O** tab in the lower window of the desktop, allows you to enter most Java statements and expressions and then execute or evaluate them immediately when you press ENTER. Interactions can be especially helpful when learning and experimenting with new elements in the Java language.

Consider the following statement that prints a String which includes escape sequences for newline (\n) and tab (\t).

```
System.out.println("Hello \n\tfrom \n\t\tInteractions!");
```

Of course you could write a short program that includes this statement, save it, compile it, and run it in order to see the results of executing the statement. However, it may be more convenient to type this statement into the Interactions tab, press ENTER, and quickly see the results as shown below in Figure 2-8.

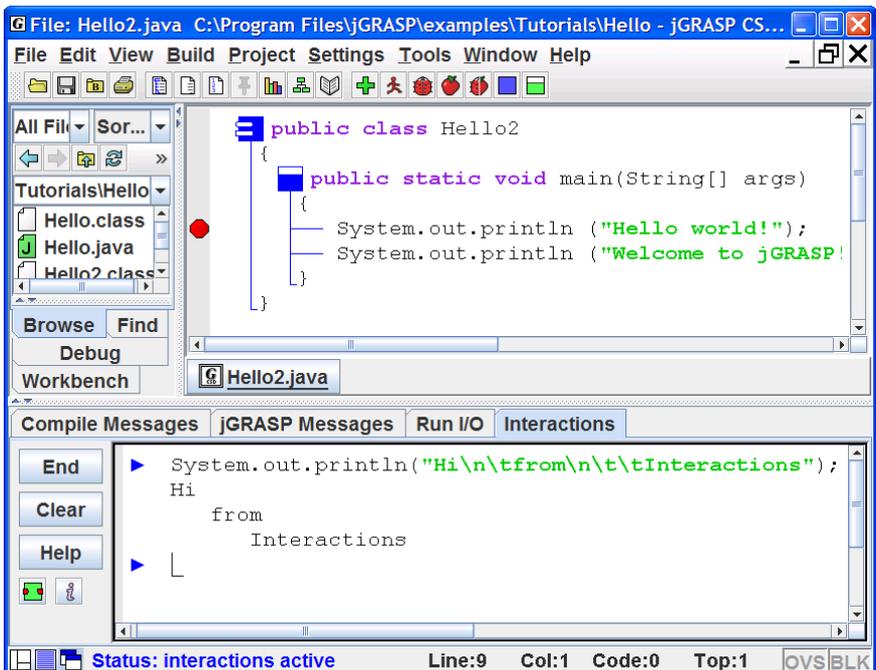


Figure 2-8. Using Interactions

2.7 Generating a Control Structure Diagram

You can generate a Control Structure Diagram in the CSD window whenever you have a syntactically correct program, such as the Hello2.java program described above. Note that CSD generation checks only the structure of a program, so even though the CSD may generate successfully, the program may not compile. Generate the CSD for the program by doing any of the following:

- (1) Click the Generate CSD button , or
- (2) Click **View > Generate CSD** on the menu, or
- (3) Press the F2 key.

If your program is syntactically correct, the CSD will be generated as shown for the Hello2.java program in Figure 2-10. After you are able to successfully generate a CSD, go on to the next section below.

If a syntax error is detected during the CSD generation, jGRASP will highlight the vicinity of the error and describe it in the message window.

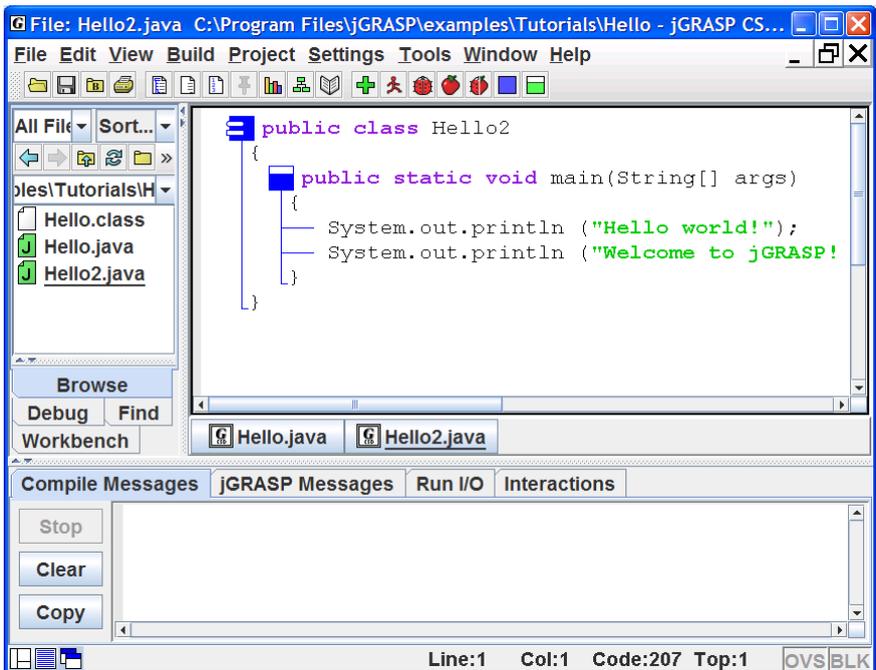


Figure 2-10. After CSD is generated

If you do not find an error in the highlighted line, be sure to look for the error in the line just above it. For example in Figure 2-11, the semi-colon was omitted at the end of the first `println` statement. As you gain experience, these errors will become easier to spot.

If you are unable find and correct the error, you should try compiling the program since the compiler may provide a more detailed error message (see [Compiling](#) below).

You can remove the CSD by doing any of the following:

- (1) Click the Remove CSD button , or
- (2) Click **View > Remove CSD** on the menu, or
- (3) Press Shift-F2.

Note that it is not necessary to remove the CSD before compiling or saving a program. Your programs will always be saved as plain text. Many users never remove the CSD. In fact, many turn on Auto Generate (**View >** then check **ON Auto Generate CSD**) so that they will always have the CSD with their code.

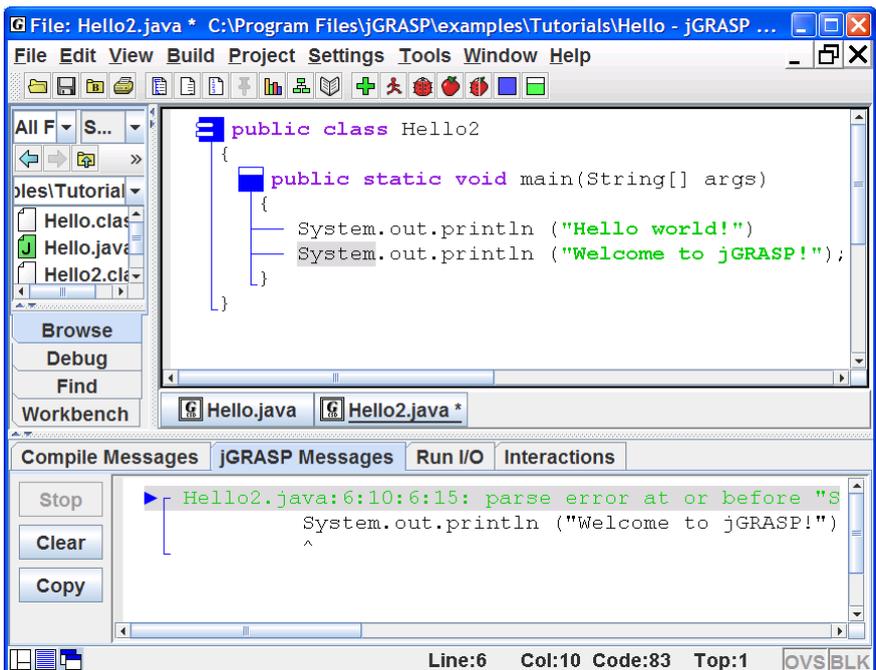


Figure 2-11. Syntax error detected

Remember, the purpose of using the CSD is to improve the readability of your program. While this may not be obvious on a simple program like the example above, it should become apparent as the size and complexity of your programs increase.

TIP: As you enter a program, try to enter it in “chunks” that are syntactically correct. For example, the following is sufficient to generate the CSD.

```
public class Hello
{
}
```

As soon as you think you have entered a syntactically correct chunk, you should generate the CSD. Not only does this update the diagram, it catches your syntax errors early.

2.8 Folding a CSD

Folding is a CSD feature that becomes increasingly useful as programs get larger. After you have generated the CSD, you can fold your program based on its structure. For example, if you double-click on the class symbol in the program, the entire program is folded (Figure 2-12). Double-clicking on the class symbol again will unfold the program completely. If you double-click on the “plus” symbol, the first layer of the program will be unfolded. Large programs can be unfolded layer by layer as needed. Although the example program has no loops or conditional statements, these may be folded by double-clicking the corresponding CSD control constructs. For other folding options, see the **View > Fold** menu.

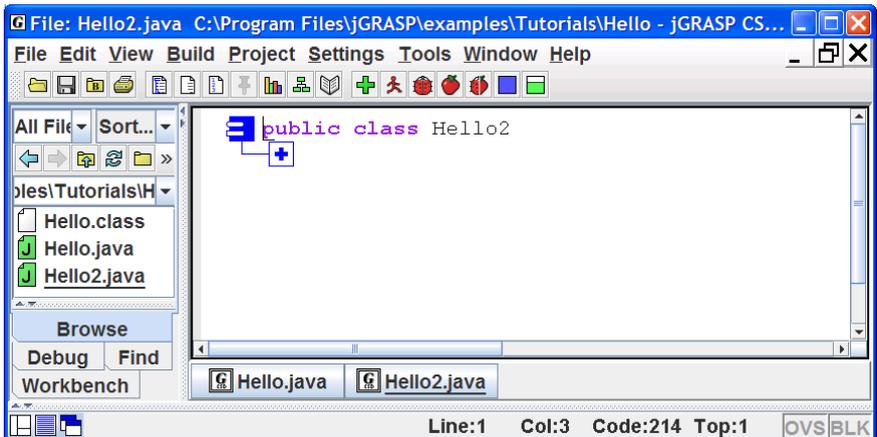


Figure 2-12. Folded CSD

2.9 Line Numbers

Line numbers can be very useful when referring to specific lines or regions of a program. Although not part of the actual program, they are displayed to the left of the source code as indicated in Figure 2-13.



Line numbers can be turned on and off by clicking the line numbers toggle button on the CSD window toolbar or via the View menu.

With Line numbers turned on, if you insert a line in the code, all line numbers below the new line are incremented.



You may “freeze” the line numbers to avoid the incrementing by clicking on the Freeze Line Numbers button. To unfreeze the line numbers, click the button again. This feature is also available on the View menu.

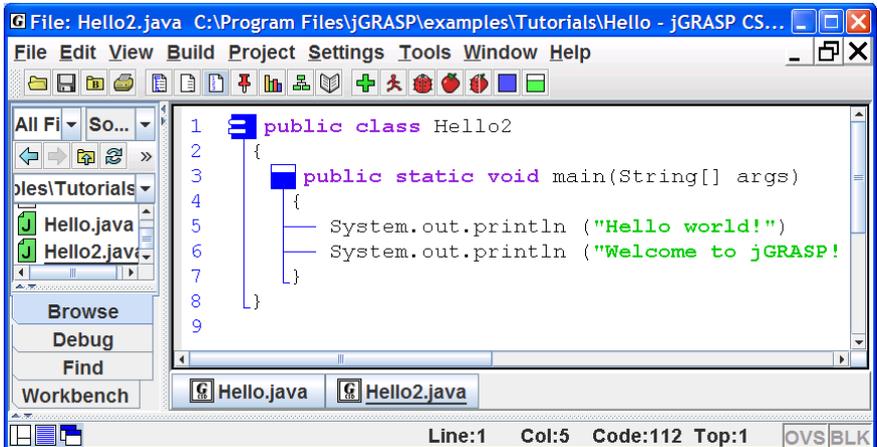


Figure 2-13. Line numbers in the CSD Window

2.10 Compiling a Program – A Few More Details

When you have a program in the CSD window, either through loading a source file or typing in the program and saving it, you are ready to compile the program. When you compile your program, the file is automatically saved if Auto Save is ON, which it is by default. Auto Save can be turned on/off by clicking **Settings > Auto Save**. If you are compiling a program in a language other than Java, you will typically need to “compile and link” the program.

 Compile a Java program in jGRASP by clicking the Compile button or by clicking on the Compiler menu: **Build > Compile** (Figure 2-14).

 Compile and Link the program (if you are compiling in a language other than Java) by clicking on the Compile and Link button or by clicking on the Build menu: **Build > Compile and Link**. Note that this option will not be visible on the toolbar and menu in a CSD window for a Java program.

In the figure below, also note that **Debug Mode** is checked ON. This should always be left on so that the `.class` file created by the compiler will contain information about variables in your program that can be displayed by the debugger and Object Workbench.

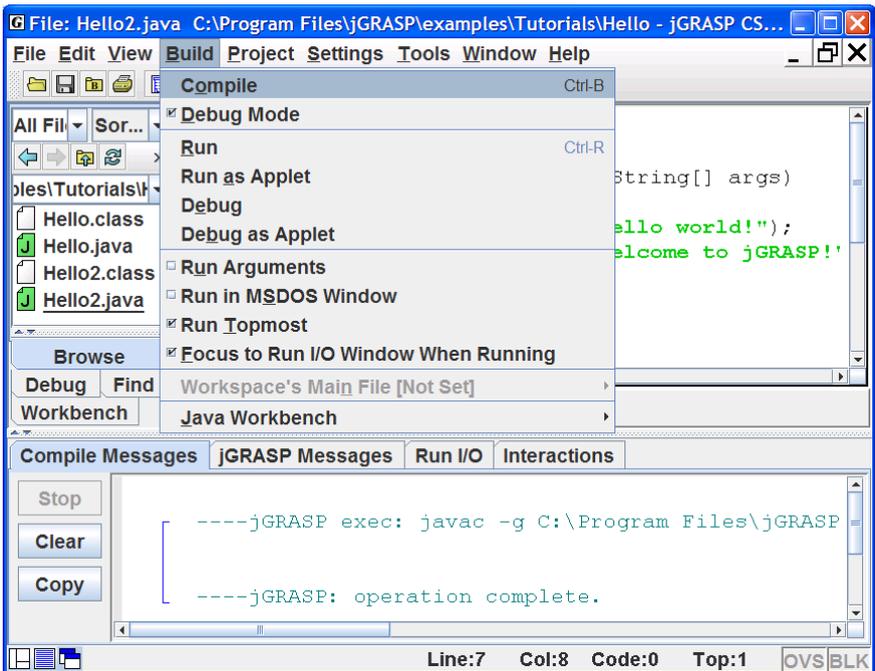


Figure 2-14. Compiling a program

The results of the compilation will appear in the **Compile Messages** tab in the lower window of the Desktop. If your program compiled successfully, you should see the message “operation complete” with no errors reported, as illustrated in Figure 2-14. Now you are ready to “Run” the program (see Section 2.11 *Running A Program – Additional Options*).

Error Messages – An error message indicating “file not found,” generally means jGRASP could not find the compiler. For example, if you are attempting to compile a Java program and the message indicates that “javac” was not found, this means the Java compiler (javac) may not have been installed properly. Go back to Section 1, Installing jGRASP, and be sure you have followed all the instructions. Once the Java JDK is properly installed and set up, any errors reported by the compiler should be about your program.

Figure 2-15 shows a program with a missing “)” in the first println statement. The error description is highlighted in the Compiler Message tab, and jGRASP automatically scrolls the CSD window to the line where the error most likely occurred and highlights it. If multiple errors are indicated, you should correct all that are obvious and then compile the program again. Sometimes correcting one error can clear up several error messages.

After you have “fixed” all reported errors, your program will successfully compile, which means a .class file will be created for your .java file. After this .class file has been created, you can “Run” the program as described in the next section.

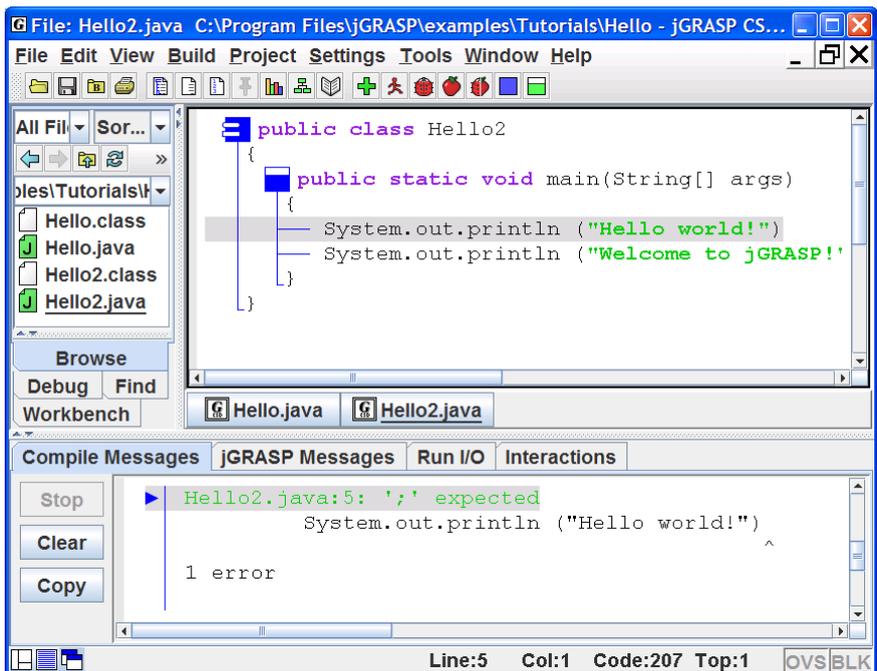


Figure 2-15. Compile time error reported

2.11 Running a Program - Additional Options

At this point you should have successfully compiled your program. Two things indicate this. First, there should be no errors reported in the Compile Messages window. Second, you should have a Hello2.class file listed in the Browse pane, assuming the pane is set to list “All Files.”

To run the program, click **Build > Run** on the toolbar (Figure 2-16). The options on the Build menu allow you to run your program: as an application (**Run**), as an Applet (**Run as Applet**), as an application in debug mode (**Debug**), and as an Applet in debug mode (**Debug as Applet**). Other options allow you to pass Run arguments, Run in an MS-DOS window rather than the jGRASP Run I/O message pane, and Run Topmost to keep frames and dialogs of the program on top of jGRASP components.



You can also run the program by clicking the Run button on the tool bar.

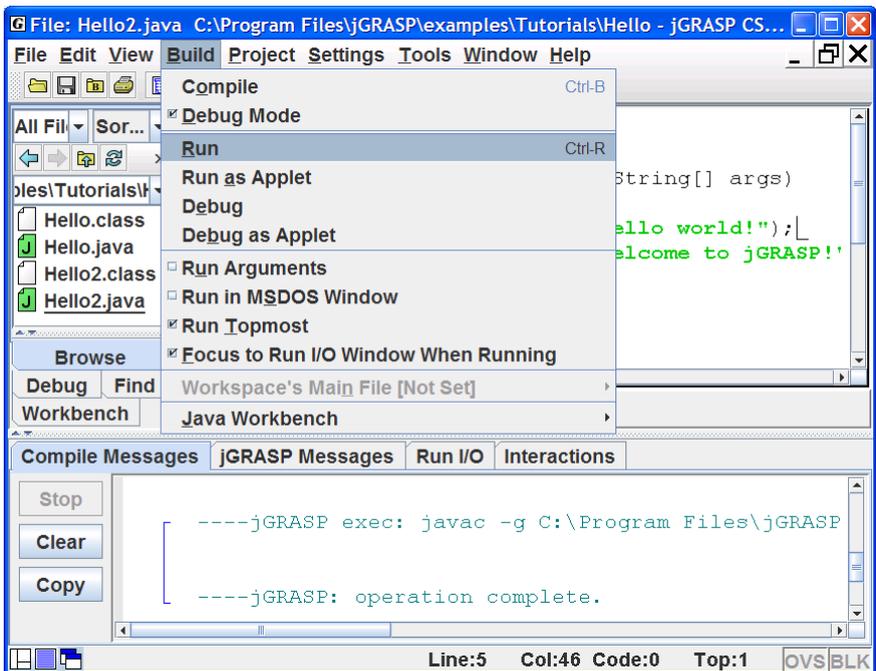


Figure 2-16. Running a program

Output – If a program has any standard input and/or output, the Run I/O tab in the lower pane pops to the top of the Desktop. In Figure 2-17, the output from running the Hello2 program is shown in Run I/O tab.

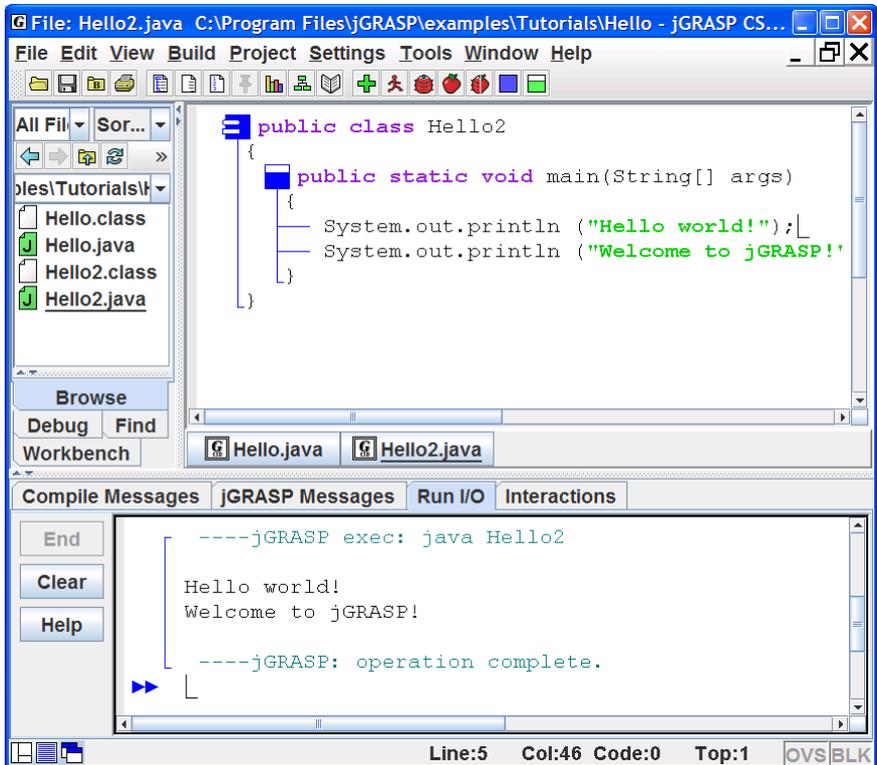


Figure 2-17. Output from running the program

2.12 Using the Debugger (Java only)

jGRASP provides an easy-to-use visual Debugger for Java that allows you to set one or more breakpoints in your program, run the debugger, then after the program reaches a breakpoint, step through your program statement by statement. To set a breakpoint, hover the mouse over the gray column to the left of the line where you want to set the breakpoint. When you see the red breakpoint symbol, left-click the mouse to set the breakpoint. You can also set a breakpoint by left-clicking on the statement where you want your program to stop, then right-clicking to select **Toggle Breakpoint** (Figure 2-18). Alternatively, after left-clicking on the line where you want the breakpoint, click **View > Breakpoints > Toggle Breakpoint**. You should see the red octagonal

breakpoint symbol  appear in the gray area to the left of the line. The statement you select must be an executable statement (i.e., one that causes the program to do something). In the Hello2 program below, a breakpoint has been set on the first of the two `System.out.println` statements, which are the only statements in this program that allow a breakpoint.

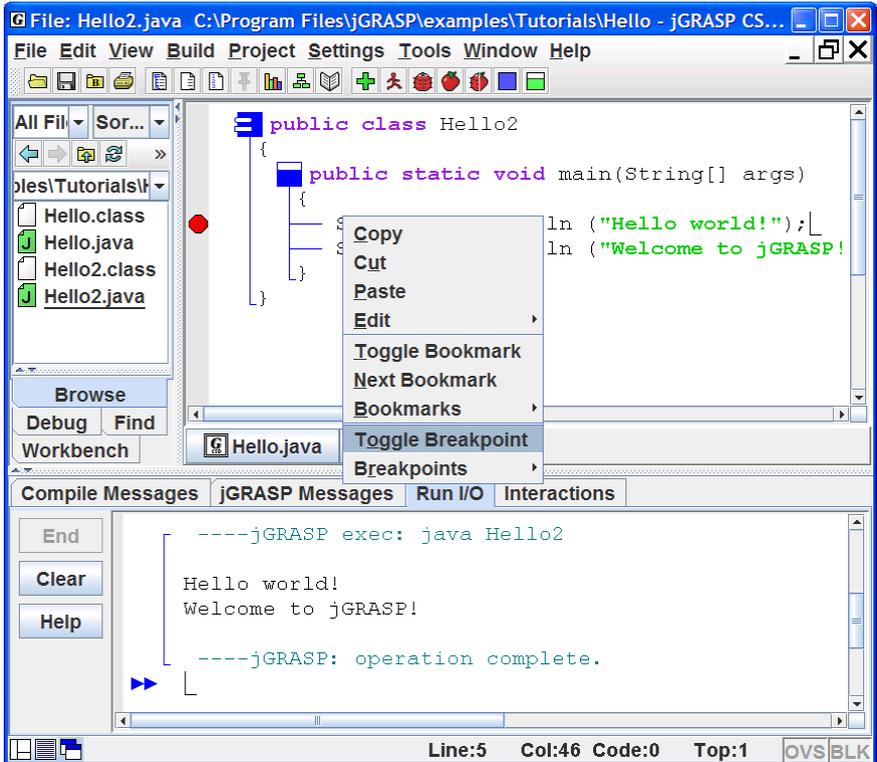


Figure 2-18. Setting a breakpoint

To start the debugger on an application, click the debug button  on the toolbar. Alternatively, you can click **Build > Debug**. When the debugger starts, the Debug tab with control buttons (Figure 2-19) should pop up in place of the Browse tab, and your program should stop at the breakpoint as shown in Figure 2-20 below.

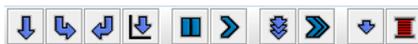


Figure 2-19. Debugger control buttons

Only the “step” button  of the debugger control buttons, located at the top of the Debug tab, is needed in this section. Each time you click the “step”

button , your program should advance to the next statement. After stepping all the way through your program, the Debug tab pane will go blank to signal the debug session has ended. When a program contains variables, you will be able to view the values of the variables in the Debug tab as you step through the program.

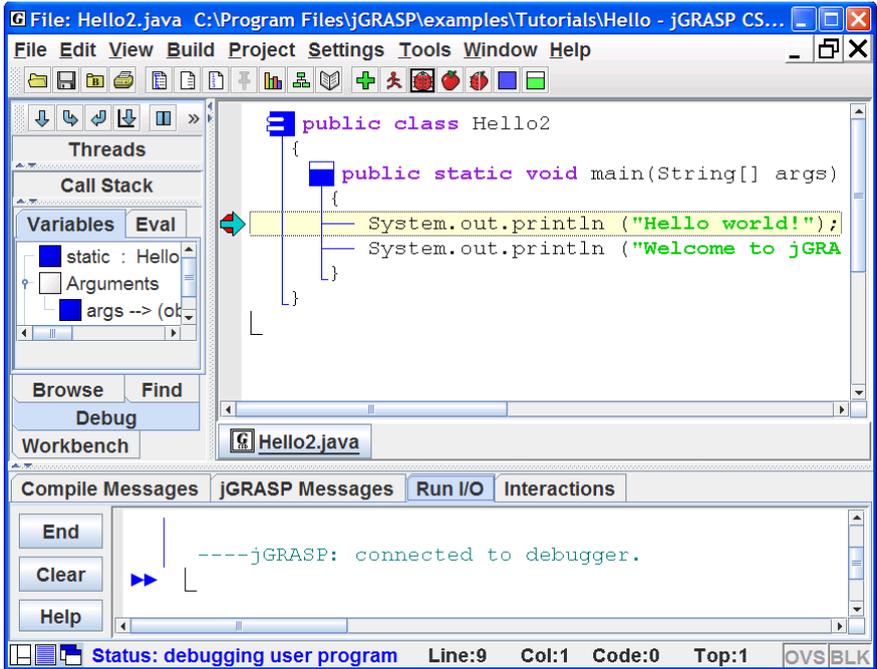


Figure 2-20. Stepping with the Debugger

In the example below, the program has stopped at the first output statement. When the step button is clicked, this statement will be executed and “Hello world!” will be output to the Run I/O tab pane. Clicking the step button again will output “Welcome to jGRASP!” on the next line. The third click on the step button will end the program, and the Debug tab pane should go blank as indicated above. When working with the debugger, remember that the highlighted statement with the blue arrow pointing to it will be the next statement to be executed. For a complete description of the other debugger control buttons, see the tutorial on the *Integrated Debugger*.

2.13 Opening a File – Additional Options

A file can be opened in a CSD window in a variety of ways. Each of these is described below.

- (1) **Browse Tab** - If the file is listed in jGRASP Browse tab, you can simply double click on the file name, and the file will be opened in a new CSD window. We did this back in section **2.1 Quick Start**. You can also drag a file from the Browse tab and drop it in the CSD window area.
- (2) **Menu or Toolbar** - On the menu, click **File > Open** or Click the Open File button  on the toolbar. Either of these will bring up the Open File dialog shown in Figure 2-21.

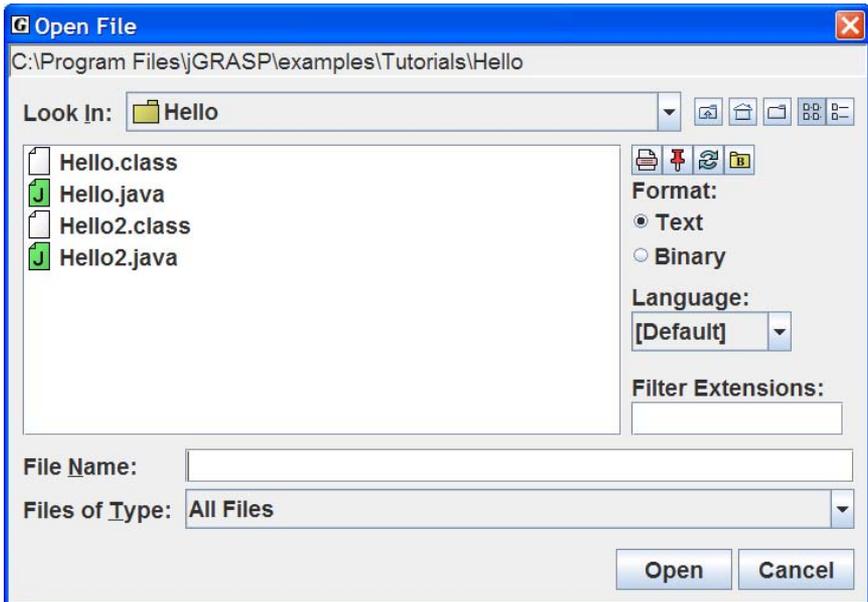


Figure 2-21. Open File dialog

- (3) **Windows File Browser** - If you have a Windows file browser open (e.g., My Computer, My Documents, etc.), and the file is marked as a jGRASP file, you can just double click the file name.
- (4) **Windows File Browser (drag and drop)** - If you have a Windows file browser open (e.g., My Computer, My Documents, etc.), you can drag a file from the file browser to the jGRASP Desktop and drop it in the area where the CSD window would normally be displayed.

In all cases above, if a file is already open in jGRASP, the CSD window containing it will be popped to the top of the Desktop rather than jGRASP opening a second CSD window with the same file.

Multiple CSD Windows – When you have multiple files open, each is in a separate CSD window. Each program can be compiled and run from its respective CSD window. When multiple windows are open, the single menu and toolbar go with the top window only, which is said to have “focus” in the desktop. In Figure 2-22, two CSD windows have been opened. One contains Hello.java and the other contains Hello2.java. If the window in which you want to work is visible, simply click the mouse on it to bring it to the top. If you have many windows open, you may need to click the **Window** menu, then click the file name in the list of the open files. However, the easiest way to give focus to a window is to click the window’s button on the *windowbar* below the CSD window. As described earlier, these buttons can be reordered by dragging/dropping them on the windowbar. In the figure below, the windowbar has buttons for Hello and Hello2. Notice that Hello2.java is underlined both on the windowbar and in the Browse tab to indicate that it has the current focus. Hello2.java is also displayed in the desktop’s blue title bar.

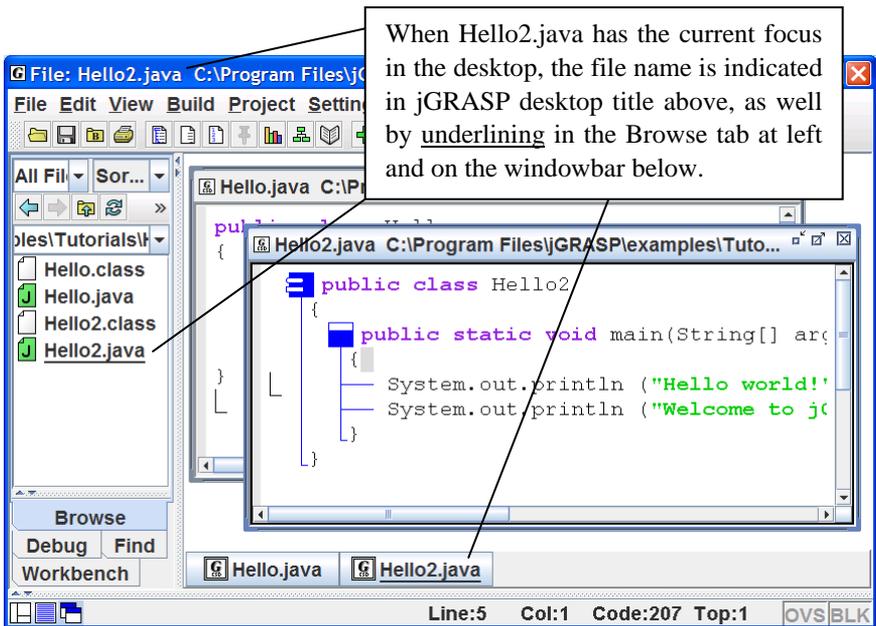


Figure 2-22. Multiple files open

2.14 Closing a File

The open files in CSD windows can be closed in several ways.

- (1)  If the CSD window is maximized, you can close window and file by clicking the Close button at the right end of the top level Menu.
- (2)  If the CSD window is not maximized, click the Close button in the upper right corner of the CSD window itself.
- (3) **File Menu** – Click **File > Close** or **Close All Files**.
- (4) **Window Menu** – Click **Window > Close All Windows**.

In each of the scenarios above, if the file has been modified and not saved, you will be prompted to *Save and Exit*, *Discard Edits*, or *Cancel* before continuing. After the files are closed, your Desktop should look like the one shown in Figure 2-23, which is essentially how we began this tutorial.

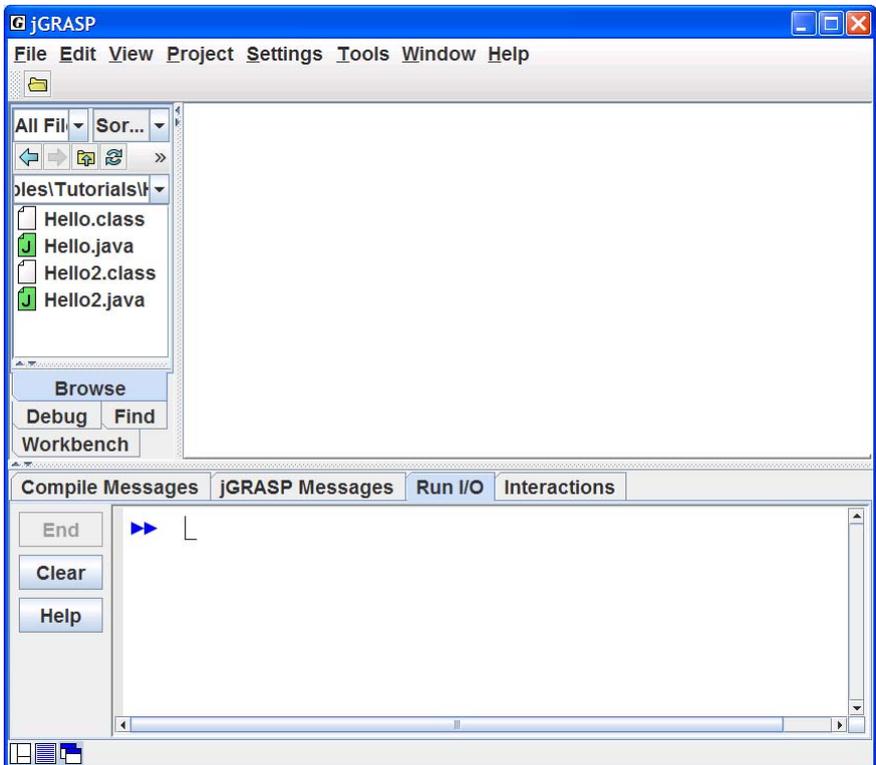


Figure 2-23. Desktop with all CSD Windows closed

2.15 Exiting jGRASP

When you have completed your session with jGRASP, you should always close (or “exit”) jGRASP rather than let your computer close it when you log out or shut down. However, you don’t have to close the files you have been working on before exiting jGRASP. When you exit jGRASP, it remembers the files you have open, including their window size and scroll position, before closing them. If a file was edited during the session, jGRASP prompts you to save or discard the changes. The next time you start jGRASP, it will open your files, and you will be ready to begin where you left off. For example, open the Hello.java file and then exit jGRASP by one of the methods below. After jGRASP closes down, start it up again and you should see the Hello.java program in a CSD window. This feature is so convenient that many users tend to leave a few files open when they exit jGRASP. However, if a file is really not being used, it is best to go ahead and close the file to reduce the clutter on the windowbar.

Close jGRASP in either of the following ways:

- (1) Click the Close button  in the upper right corner of the desktop;
or
- (2) On the File menu, click **File > Exit jGRASP**.

2.16 Review and Preview of What’s Ahead

As a way of review and also to look ahead, let’s take a look at the jGRASP *toolbar*. Hovering the mouse over a button on the toolbar provides a “tool hint” to help identify its function. Also, **View > Toolbar Buttons** allows you to display *text* labels on the buttons. Figure 2-24 provides a brief description of the each button.

While many of these buttons were introduced in this section, some were assumed to be self-explanatory (e.g., Print, Cut, Copy, etc.), and several others will be covered in the next section along with Projects and the Object Workbench (e.g., Generate UML, Generate Documentation, Create Object, and Invoke Method). Section 9 provides an in depth look at the CSD, which can be read at any time, but is most relevant when control structures are studied (e.g., *if*, *if-else*, *while*, *for*, *try-catch*, etc).

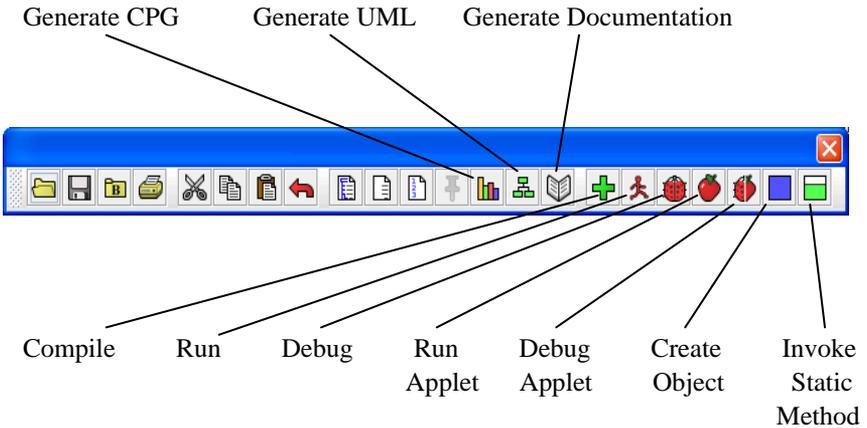
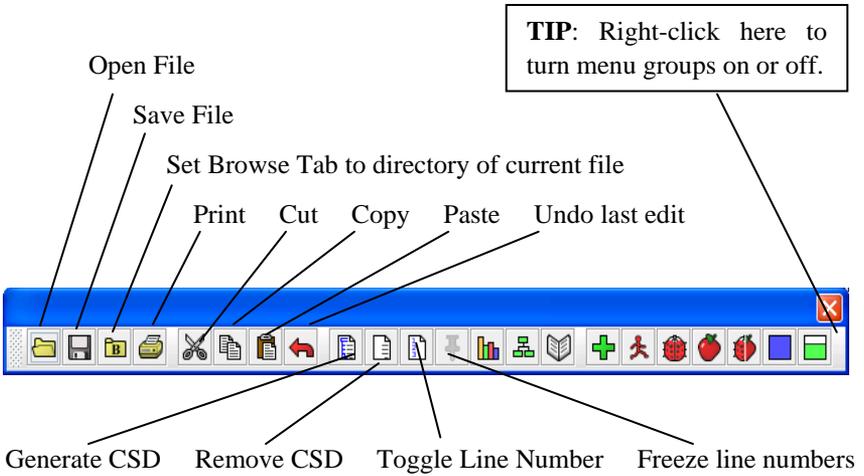


Figure 2-24. Toolbar

2.17 Exercises

- (1) Create your own program then save, compile, and run it.
- (2) Enter several statements and expressions in **Interactions** to immediately see the results of their execution and/or evaluation.
- (3) Generate the CSD for your program. On the View menu, turn on Auto Generate CSD (**Settings > CSD Window Settings** – then (checkbox) **Auto Generate CSD**).
- (4) Display the line numbers for your program.
- (5) Fold up your program then unfold it in layers.
- (6) On the Build menu, make sure Debug Mode is ON (indicated by a check box). [Note that Debug Mode should be ON by default, and we recommend that this be left ON.] Recompile your program.
- (7) Set a breakpoint on the first executable line of your program then run it with the debugger. Step through each statement, checking the Run I/O window for output.
- (8) If you have other Java programs available, open one or more of them, then repeat steps (1) through (5) above for each program.

Notes