8 UML Class Diagrams

Java programs usually involve multiple classes, and there can be many dependencies among these classes. To fully understand a multiple class program, it is necessary to understand the interclass dependencies. Although this can be done mentally for small programs, it is usually helpful to see these dependencies in a class diagram. jGRASP automatically generates a class diagram based on the Unified Modeling Language (UML). In addition to providing an architectural view of your program, the UML class diagram is also the basis for the Object Workbench which is described in a separate section.

Objectives – When you have completed this tutorial, you should be able to generate the UML class diagram for your project, display the members of a class as well as the dependencies between two classes, and navigate to the associated source code.

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

8.1 Opening the Project

The jGRASP project file is used to determine which user classes to include in the UML class diagram. The project should include all of your source files (.java), and you may optionally include other files (e.g., .class, .dat, .txt, etc.). You may create a new project file, then drag and drop files from the Browse tab pane to the UML window.

To generate the UML, jGRASP uses information from both the source (.java) and byte code (.class) files. Recall, .class files are generated when you compile your Java program files. Hence, you <u>must</u> compile your .java files in order to see the dependencies among the classes in the UML diagram**.** Note that the .class files do not have to be in the project file.

If your project is not currently open, you need to open it by doing one of the following:

(1) On the Desktop tool bar, click **Project** > **Open Project**, and then select the project from the list of project files displayed in the Open Project dialog and click the **Open** button.

(2) Alternatively, in the files section of the Browse tab, double-click the project file.

When opened, the project and its contents appear in the open projects section of the Browse tab, and the project name is displayed at the top of the Desktop. If you need additional help with opening a project, review the previous tutorial on *Projects*.

The remainder of this section assumes you have created your own project file or that you will use PersonalLibraryProject from the examples that are included with jGRASP.

TIP: Remember that your Java files must be compiled before you can see the dependencies among your classes in the UML diagram. When you recompile any file in a project, the UML diagram is automatically updated.

8.2 Generating the UML

In Figure 8-1, PersonalLibraryProject is shown in the Open Projects section of the Browse tab along with a **UML** symbol and the list of files in the project. To generate the UML class diagram, double-click the UML symbol. Alternatively, on the Desktop menu, click on **Project** > **Generate/Update UML Class Diagram**.

The UML window should open with a diagram of all class files in the project as shown below. You can select one or more of the class symbols and drag them around in the diagram. In the figure, the class containing *main* has been dragged to the upper left of the diagram and the legend has been dragged to the lower center.

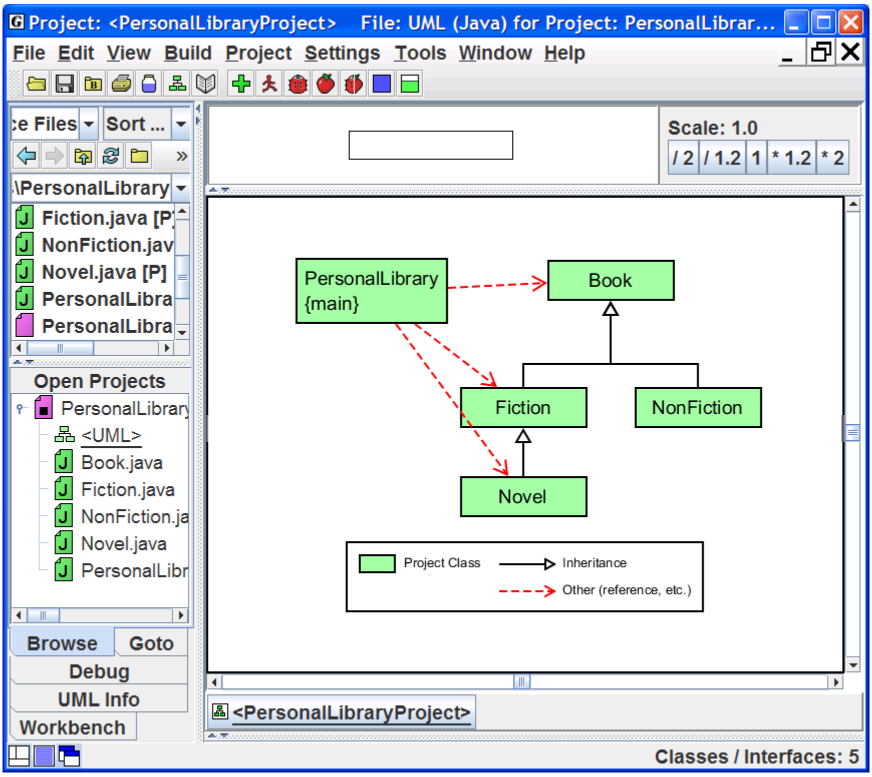


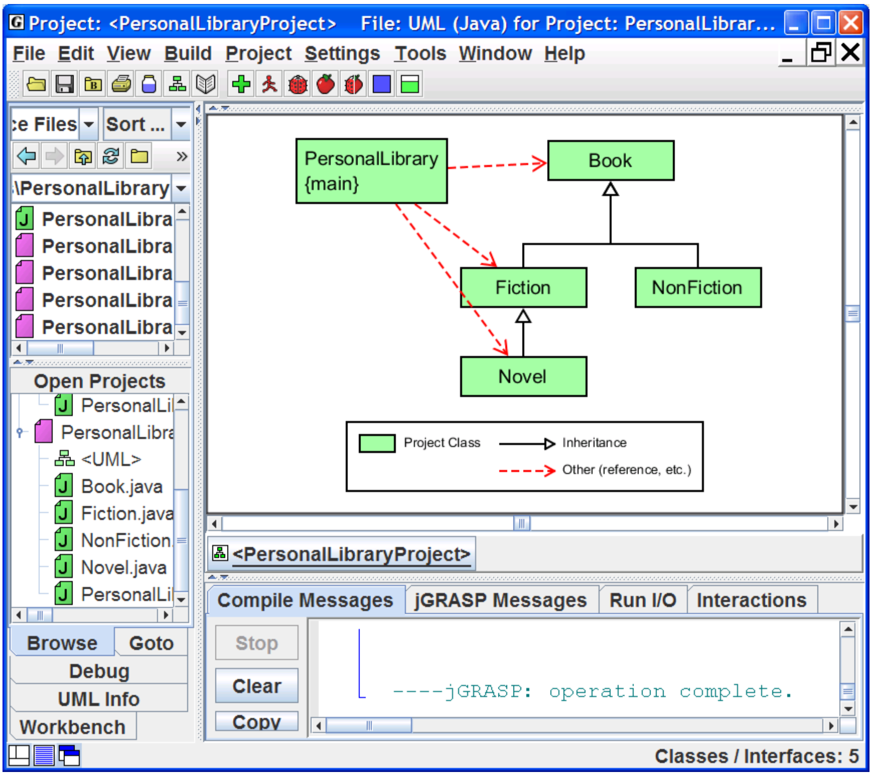
Figure 8-1. Generating the UML

The UML window is divided into three panes. The top pane contains a **panning rectangle** that allows you to reposition the entire UML diagram by dragging the panning rectangle around. To the right of the panning rectangle are buttons for scaling the UML: divide by 2 (/2), divide by 1.2 (/1.2), no scaling (1), multiply by 1.2 (*1.2), and multiply by 2 (*2). In general, the class diagram is automatically updated as required; however, the user can force an update by clicking the Update UML diagram button on the desktop menu.

If your project includes class inheritance hierarchies and/or other dependencies as in the example, then you should see the appropriate red dashed and solid black dependency lines. The meaning of these lines is annotated in the *legend* as appropriate.

8.3 Compiling and Running from the UML Window

The Build menu and buttons on the toolbar for the UML window are essentially the same as the ones for the CSD window. For example, clicking the Compile button compiles all classes in the project (Figure 8-2). When a class needs to be recompiled due to edits, the class symbol in the UML diagram is marked with red crosshatches (double diagonal lines). During compilation, the files are marked and then unmarked when done. Single red diagonal lines in a class symbol indicate that another class upon which the first class depends has been modified. Clicking the **Run** button on the toolbar will launch the program as an application if there is a *main()* method in one of the classes. Clicking on the **Run as Applet** button will launch the program as an applet if one of the classes is an applet. Similarly, clicking the **Debug** button or the **Debug Applet** button will launch the program in debug mode. Note that for running in debug mode, you should have a breakpoint set somewhere in the program so that the debugger will stop.



```
----jGRASP: operation complete.
```

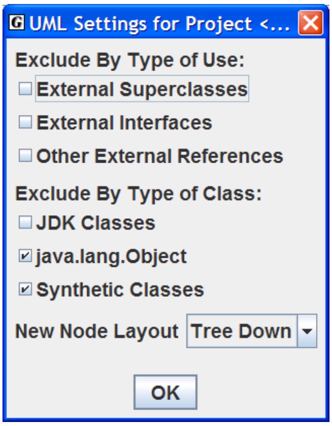
Figure 8-2. Compiling Your Program

8.4 Determining the Contents of the UML Class Diagram

jGRASP provides one group of options to control the contents of your UML diagram, and another group to determine which elements in the diagram are actually displayed. **Settings** > **UML Generation Settings** allows you to control the contents of the diagram by excluding certain categories of classes (e.g., external superclasses, external interfaces, and all other external references). **The View menu** allows you to make visible (or hide) certain categories of classes and dependencies that are actually in the UML diagram. Both options are described below.

Most programs depend on one or more JDK classes. Suppose you want to include these JDK classes in your UML diagram (the default is to exclude them). You will need to change the UML generation settings in order to not exclude these items from the diagram. Also, if you do not see the red and black dependency lines expected, then you may need to change the View settings. These are described below.

Excluding (or not) items from the diagram - On the UML window menu, click on **Settings** > **UML Generation Settings**, which will bring up the **UML Settings** dialog. Generally you should leave the top three items unchecked so that they are not excluded from the UML diagram. Now for our example of not excluding the JDK classes, under **Exclude by Type of Class**, uncheck (turn OFF) the checkbox that excludes **JDK Classes,** as shown in Figure 8-3. Note that synthetic classes are created by the Java compiler and are usually not included in the UML diagram. After checking (or unchecking) the items so that your dialog looks like the one in the figure, click the OK button. This should close the dialog and update the UML diagram. All JDK classes used by the project classes should now be visible in the diagram as gray boxes. This is shown in Figure 8-4 after the JDK classes have been dragged around. To remove them from the diagram, you will need to turn on the exclude option. If you want to leave them in the diagram but not always display them, see the next paragraph. For more information see UML Settings in jGRASP **Help**.


UML Settings for Project <...

Exclude By Type of Use:
- [ ] External Superclasses
- [ ] External Interfaces
- [ ] Other External References

Exclude By Type of Class:
- [ ] JDK Classes
- [✓] java.lang.Object
- [✓] Synthetic Classes

New Node Layout: Tree Down

Figure 8-3. Changing the UML Settings

Making objects in the diagram visible (or not) - On the UML window menu, click on **View** > **Visible Objects**, then check or uncheck the items on the list as appropriate. In general, you will want all of the items on the list in **View** >

Visible Objects checked ON as shown in Figure 8-4. For example, for the JDK classes and/or other classes outside the project to be visible, **External References** must be checked ON. Clicking (checking) ON or OFF any of the items on the Visible Objects list simply displays them or not, and their previous layout is retained when they are redisplayed. Note that if items have been *excluded* from the diagram via **Settings** > **UML Generation Settings**, as described above, then making them visible will have no effect since they are not part of the diagram. For more information see View Menu in jGRASP **Help**.

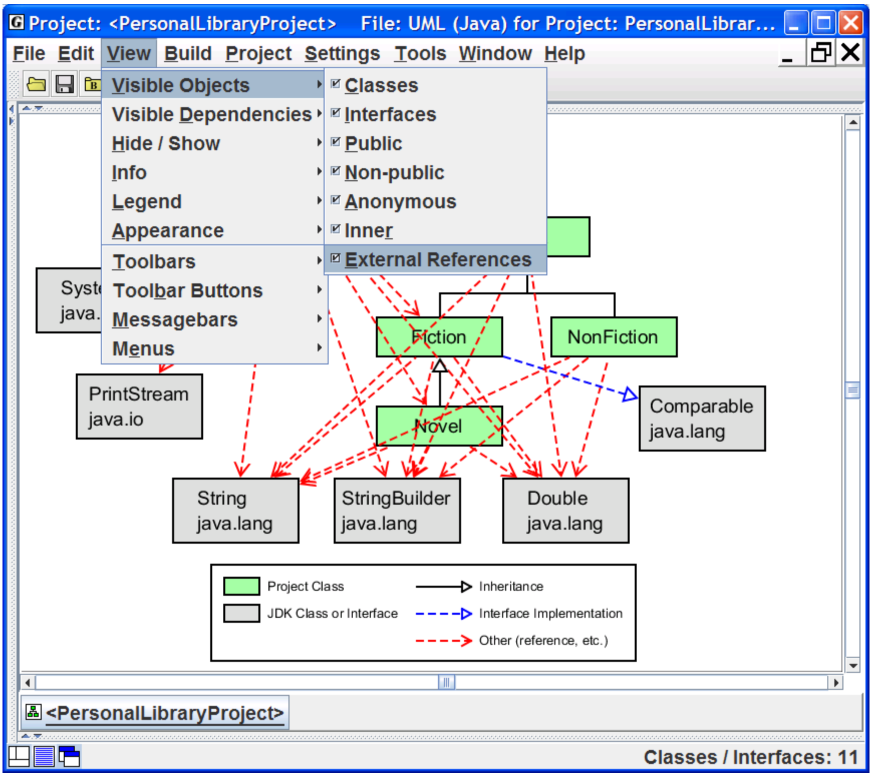


Figure 8-4. Making objects visible

Making dependencies visible - On the UML window menu, click on **View** > **Visible Dependencies**, then check or uncheck the items on the list as appropriate. The only two categories of dependencies in the example project are **Inheritance** and **Other**. **Inheritance dependencies** are indicated by black lines with closed arrowheads that point from a child to a parent to form an *is-a* relationship. Red dashed lines with open arrowheads indicate **other dependencies**. These include *has-a* relationships that indicate that a class uses fields, methods, or constructors of another class. The red dashed arrow is drawn from the class where an object is declared or referenced to the class where the item is actually defined. In general, you probably want to make all dependencies visible as indicated in Figure 8-5.

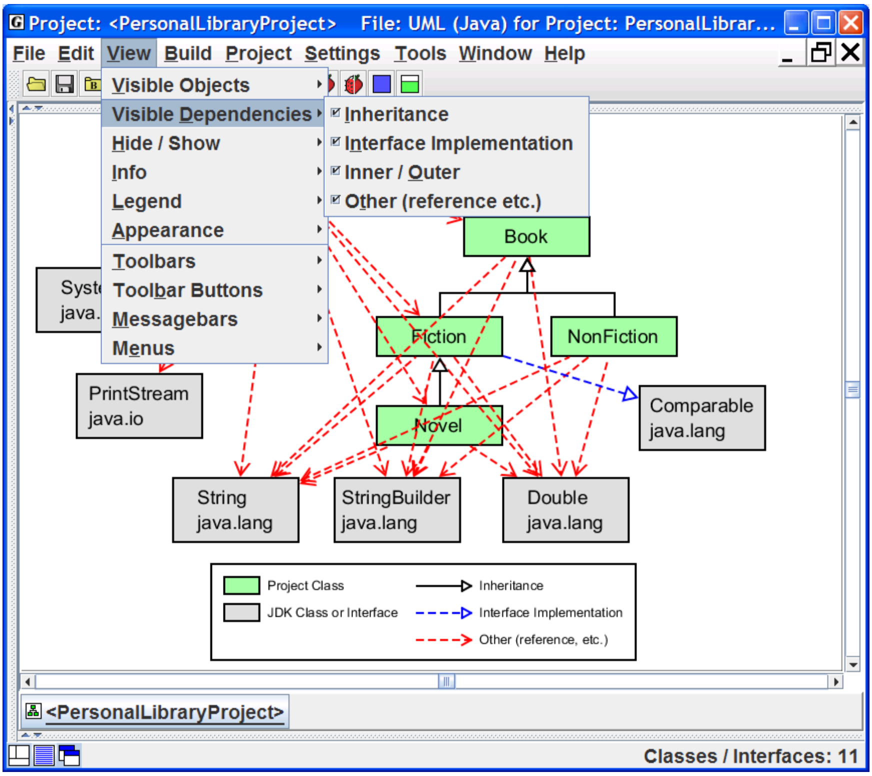


Figure 8-5. Making dependencies visible

Displaying the Legend - The legend has been visible in each of the UML diagrams (figures) in this tutorial. To set the options for displaying the legend, click **View** > **Legend**. Typically, you will want the following options checked ON: Show Legend, Visible Items Only, and Small Font. Notice that if “Visible Items Only” is checked ON, then an entry for JDK classes appears in the legend only if JDK classes are visible in the UML diagram. Experiment by turning on/off the options in **View** > **Legend**. When you initially generate your UML diagram, you may have to pan around it to locate the legend. Scaling the UML down (e.g., dividing by 2) may help. Once you locate it, just select it and drag to the location where you want it as described in the next section.

8.5 Laying Out the UML Class Diagram

Currently, the jGRASP UML diagram has limited automatic layout capabilities. However, manually arranging the class symbols in the diagram is straightforward, and once this is done, jGRASP remembers your layout from one generate/update to the next.

To begin, locate the class symbol that contains *main*. In our example, this would be the PersonalLibrary class. Remember that the project name should reflect the name of this class. Generally, you want this class near the top of the diagram. Left click on the class symbol and then, while holding down the left mouse button, drag the symbol to the area of the diagram where you want it, and then release the mouse button. Now repeat this for the other class symbols until you have the diagram looking like you want it. Keep in mind that class–subclass relationships are indicated by the *inheritance arrow* and that these should be laid out in a tree-down fashion. You can do this automatically by selecting all classes for a particular class–subclass hierarchy (hold down SHIFT and left-click each class). Then click **Edit** > **Layout** > **Tree Down** to perform the operation; alternatively, you can right-click on a selected class or group of classes, then on the pop up menu select **Layout** > **Tree Down**. Finally, right-clicking in the background of the UML window with no classes selected will allow you to lay out the entire diagram.

With a two or more classes selected, you can move them as a group. Figure 8-5 shows the UML diagram after the PersonalLibrary class has been repositioned to the top left and the JDK classes have been dragged as a group to the lower part of the diagram. You can experiment with making these external classes visible by going to **View** > **Visible Objects** > then uncheck **External References**.

Here are several heuristics for laying out your UML diagrams:

(1) The class symbol that contains *main* should go near the top of the diagram.

(2) Classes in an inheritance hierarchy should be laid out *tree-down*, and then moved as group.

(3) Other dependencies should be laid out with the red dashed line pointing downward.

(4) JDK classes, when included, should be toward the bottom of the diagram.

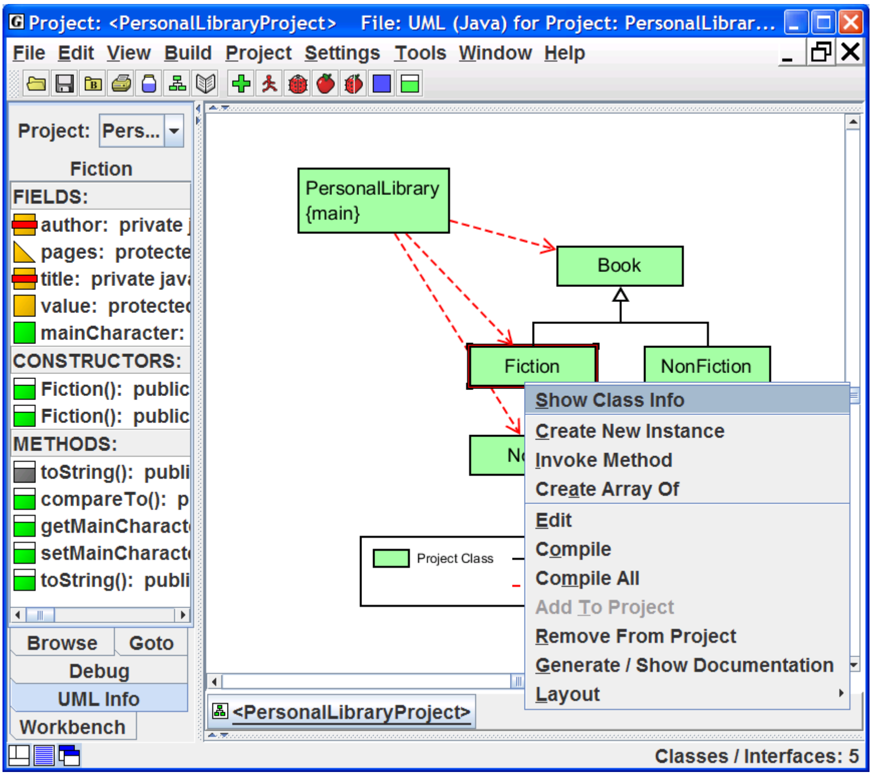
(5) Line crossings should be minimized.

(6) The legend is usually below the diagram.

8.6 Displaying the Members of a Class

To display the fields, constructors, and methods of a class, right-click on the class, then select **Show Class Info** which will pop the UML Info tab to the top in the left tab pane. Also, in the left tab pane, you can click on the **UML Info** tab to pop it to the top. Once the Info tab is on top, each time you select a class its members will be displayed.

In Figure 8-6, external classes are not visible (**View** > **Visible Objects** > then uncheck **External References**). Class Fiction has been selected and its fields, constructors, and methods are displayed in the left pane. This information is only available when the source code for a class is in the project. In the previous example, the System class from package java.lang is an external class, so selecting it would result in a "no data" message. If the only field you are seeing is mainCharacter, click **View** > **Info** > **Show Inheritance within Project**. You should now see the fields that are inherited by Fiction (i.e., author, pages, and title).



Fiction

FIELDS:
author: private j
pages: protecte
title: private jav
value: protecte
mainCharacter:

CONSTRUCTORS:
Fiction(): public
Fiction(): public

METHODS:
toString(): publi
compareTo(): p
getMainCharact
setMainCharact
toString(): publi

Figure 8-6. **Displaying class members**

8.7 Displaying Dependencies Between Two Classes

Let’s reduce the number of classes in our UML diagram by not displaying the JDK classes. Click **View** > **Visible Objects** and uncheck **External References**. Now to display the dependencies between two classes, right-click on the arrow, then select **Show Dependency Info**. You can also click on the UML Info tab to pop it to the top. Once the Info tab is on top, each time you select an arrow, the associated dependencies will be displayed.

In Figure 8-7, the edge drawn from PersonalLibrary to Fiction has been selected as indicated by the large arrowhead. The list of dependencies in the Info tab includes one constructor (Fiction) and one method (getMainCharacter). These are the resources that PersonalLibrary uses from Fiction. Understanding the dependencies among the classes in your program should provide you with a more in-depth comprehension of the source code. Note that clicking on the arrow between PersonalLibary and the PrintStream class in Figure 8-6 would show that PersonalLibary is using two println() methods from the PrintStream class. Make the **External References** visible again and try this**.**

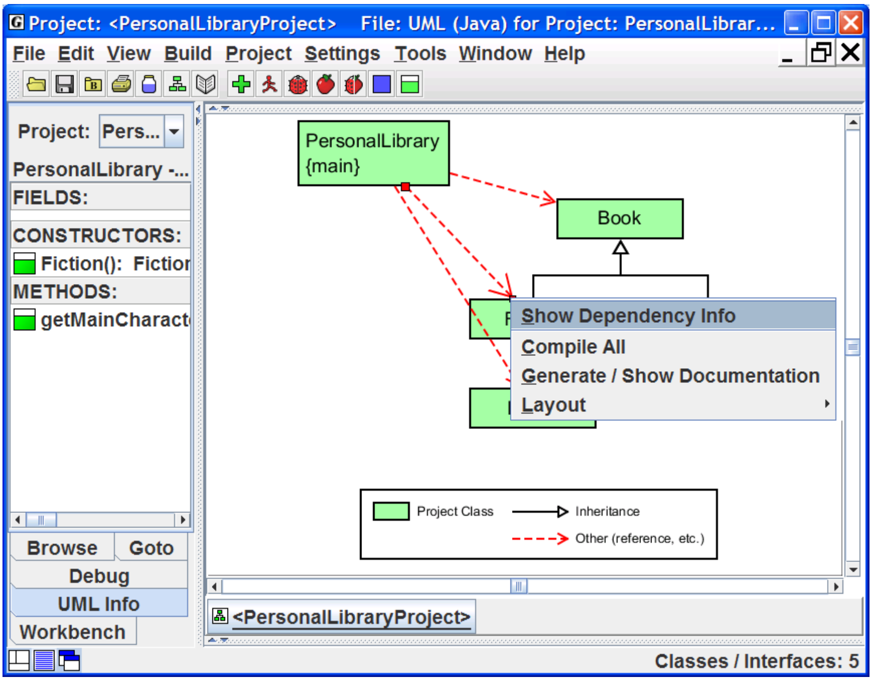


Figure 8-7. Displaying the dependencies between two classes

8.8 Navigating to Source Code via the Info Tab

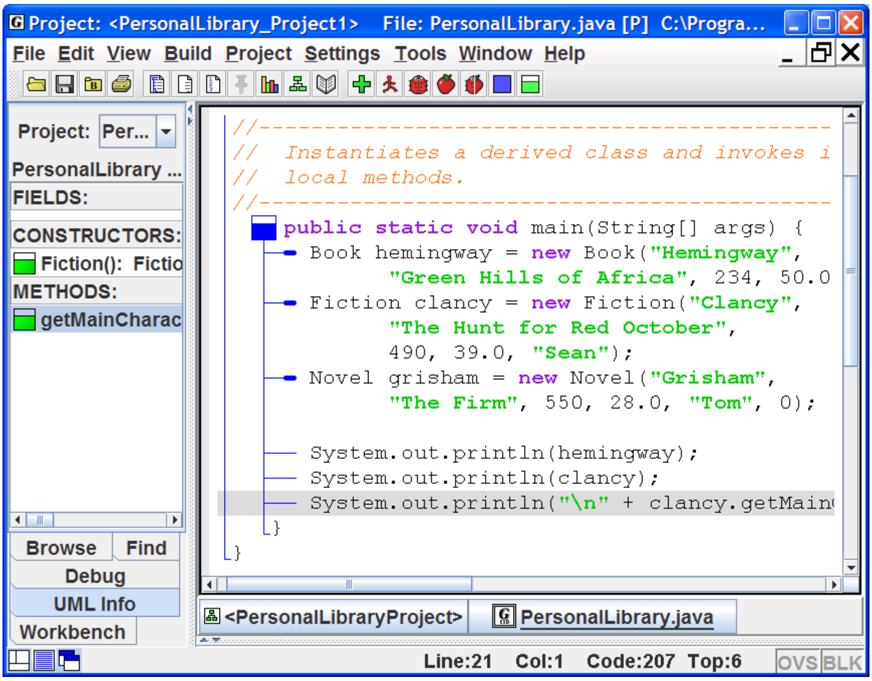
In the Info tab, a green symbol indicates that the item is defined or used in the class rather than inherited from a parent class. Double-clicking on a green item will take you to its definition or use in the source code. For example, clicking on getMainCharacer() in Figure 8-7 above will open PersonalLibrary in a CSD window with the line containing getMainCharacter() highlighted as shown in Figure 8-8 below.

8.9 Finding a Class in the UML Diagram

Since a UML diagram can contain many classes, it may be difficult to locate a particular class. In fact, the class may be off the screen. The **Goto** tab in the left pane provides the list of classes in the project. Clicking on a class in the list brings it to the center of the UML window.

8.10 Opening Source Code from UML

The UML diagram provides a convenient way to open source code files. Simply double-click on a class symbol, and the source code for the class is opened in a CSD window.



```
//--------------------------------------------
//  Instantiates a derived class and invokes i
//  local methods.
//--------------------------------------------
  public static void main(String[] args) {
     Book hemingway = new Book("Hemingway",
           "Green Hills of Africa", 234, 50.0
     Fiction clancy = new Fiction("Clancy",
           "The Hunt for Red October",
           490, 39.0, "Sean");
     Novel grisham = new Novel("Grisham",
           "The Firm", 550, 28.0, "Tom", 0);

     System.out.println(hemingway);
     System.out.println(clancy);
     System.out.println("\n" + clancy.getMain
  }
}
```

Figure 8-8. Navigating to where getMainCharacer is used in the CSD Window

8.11 Saving the UML Layout

When you close a project, change to another project, or simply exit jGRASP, your UML layout is automatically saved in the project file (.gpj). The next time you start jGRASP, open the project, and open the UML window, you should find your layout intact.

If the project file is created in the same directory as your program files (.java and .class files), and if you added the source files with *relative paths*, then you should be able to move, copy, or send the project and program files as a group (e.g., email them to your instructor) without losing any of your layout.

8.12 Printing the UML Diagram

With a UML window open, click on **File** > **UML Print Preview** to see how your diagram will look on the printed page. If okay, click the **Print** button in the lower left corner of the Print Preview window. Otherwise, if the diagram is too small or too large, you may want to go back and scale it using the scale factors near the top right of the UML window, and then preview it again.

For additional details see UML Class Diagrams in jGRASP **Help**.