The JHotDraw 7 Handbook

Version 0.4

Werner Randelshofer, 2011-10-01

Contents

Introduction to JHotDraw 7

JHotDraw 7 is a Java framework for structured drawing editors.

The purpose of a framework is the separation of domain-specific code from domain-neutral code. So that a body of domain-neutral code can be built and reused in many different cases. In this respect, a framework is similar to a library. However, unlike a library, which only provides domain-neutral functions, a framework also provides domain-neutral control flows.

JHotDraw 7 consists of two parts: the drawing framework and the application framework.

The drawing framework supports the creation of various kinds of 2D vector drawing editors, from simple doodle tools up to full fledged diagram editors.

The application frameworks supports the creation of document oriented applications which can follow a range of platform dependent style guides, such as Apple's Mac OS X Human Interface Guidelines [1] and Microsoft's Windows User Experience Interaction Guidelines [2].

JHotDraw 7 is a fork of the JHotDraw framework by Eric Gamma et al. JHotDraw 7 has been originally developed as a research project at the Lucerne University of Applied Sciences by Werner Randelshofer.

How to read this Handbook

What you should already know

JHotDraw 7 is not for beginners. You should have a good understanding of object-oriented programming, of the Java programming language, and of the Java Swing API.

If you are a student, I would say, you should have had at least two semesters of informatics. If you are an autodidact, you should have at least one year of experience with Java, including user interface programming on the client.

If this document does not answer your question

This document is work in progress, and will be extended as needed. If this document does not answer your question, please ask it at the JHotDraw forums at http://sourceforge.net/projects/jhotdraw/forums

Application Framework Overview

The JHotDraw 7 application framework is designed for document oriented applications which are implemented with the Swing API, and can be deployed on Windows and on Mac OS X.

Document oriented applications

A document oriented application works with documents.

A document is a media which holds structured content, for example a text, a spreadsheet, or a drawing.

Documents reside in a document store, for example on the file system.

With a document oriented application, a user can open a document from the document store, and view its content. If the application supports editing, the user can also create a new document, and change the contents of an existing document.

In addition to editing, a typically document oriented application provides clipboard functionality (cut, copy, paste) and undo functionality. We can also expect that an application remembers the most recent documents that we opened, and that it will warn us when we try to close a view which has unsaved changes.

Supporting platform specific user interfaces

On Windows, document oriented applications are typically realized with the single document interface (SDI) or the multiple document interface (MDI). Mac OS X, uses an application-centric interface (OSX).

An SDI application consists of a single window which holds a single document. An SDI application may allow to open more than one window, but for the user, each window acts as an application of its own. If the application supports editing, then there is one separate editor in each window. Each window has its own set of menu bars and toolbars. Closing a document window means quitting the application.

An MDI application consists of a main window which contains child windows for each opened document. If the application supports editing, then there is one editor for all child windows. Only the main window has menu bars and toolbars. Closing a document window does not quit the application.

An OSX application is very much like an MDI application. Instead of a main window, we have a main menu bar, which is attached at the top of the screen. Documents are opened in separate windows. Toolbars are on floating palette windows. If the application supports editing, then there is one editor for all document windows. All windows can be closed without quitting the application.

Creating an application with the plain Swing API

There is no explicit support for SDI-, MDI- or OSX-interfaces in the Swing API.

If we wanted to create a document oriented application that should run on Windows and Mac OS X with Swing, we would probably start with the platform-independent part of the application.

We could start with a JComponent for the domain-specific part of the user interface, for example a JPanel. On the JPanel, we could place all components needed for viewing and editing a document.

We could then write domain-neutral code, which creates the JFrame's, JToolBar's and JMenuBar's needed for a specific interface type. We could put this code into a class with a main() method.

Swing supports the abstraction of control flow into Action objects. For example, we could create reusable control flows for opening and closing documents. The control flow could invoke domain specific methods for reading and writing files.

Creating an application with the JHotDraw 7 framework

The JHotDraw 7 application framework provides the following key interfaces for an application:

- View – A JComponent which holds the domain-specific view to a document
- Application – Domain-neutral code which manages views, menus and toolbars
- ApplicationModel – Domain-specific code used by the application
- Action (from the Swing API) – Domain-neutral control flows.

These interfaces are defined in the package org.jhotdraw.app. The Action interface is defined in the javax.swing package.

We typically start by implementing the View interface. For example, by subclassing AbstractView. AbstractView extends from JPanel and allows us to place components for viewing and editing a document on it.

Next, we could implement the ApplicationModel interface by subclassing DefaultApplicationModel. Here we can add code which creates Action objects for use by the Application as well as menu items and toolbars. The ApplicationModel also serves as a factory for View objects.

And finally, we create a class with a main() method which creates the desired kind of Application, for example SDIApplication, and passes it our ApplicationModel.

Drawing Framework Overview

The JHotDraw 7 drawing framework is specifically designed for the creation of editors for two dimensional structured drawings and diagrams.

Structured Drawing Editors

A structured drawing is a drawing which is made up from figures.

A figure is a graphical shape, which can be individually added, changed and removed from the drawing. A figure can provide handles for editing.

A drawing, its figures and handles are presented in a drawing view. A drawing view can maintain a selection of figures for editing.

An editor provides editing functionality for drawing views. The editor can be in different editing states. The editing states are represented by Tool objects.

Creating a drawing editor with the plain Swing API

If we wanted to create a structured drawing editor with Swing, we could start with the drawing and the figures objects. We could implement them as JavaBeans. Drawing could provide a draw() method, which in turn invokes a draw() method on the figures.

Next we could implement the drawing view. This could be a subclass of JComponent. The drawing view could override method paintComponent() to invoke the draw() method of the drawing object.

The editor object could be another JavaBean, which maintains a list of drawing views. For each state of the editor, we could implement a Tool object, which registers mouse- and keyboard-listeners on the drawing views for editing.

Creating a drawing editor with the JHotDraw 7 framework

The JHotDraw 7 drawing framework provides the following key interfaces for an application:

- Drawing – Holds figures.
- Figure – Represents an editable shape of a drawing. Can create Handle objects.
- DrawingView – Provides a view for a drawing. Maintains a selection state.
- DrawingEditor – Manages DrawingView's and has an exchangeable state

represented by a Tool object.

- Tool – Listens on mouse- and keyboard-events on all drawing views of the editor and interprets them for editing a drawing or forwards the events to a Handle.
- Handle – Can be used by Tool objects to edit a Figure.

These interfaces are defined in the package org.jhotdraw.draw.

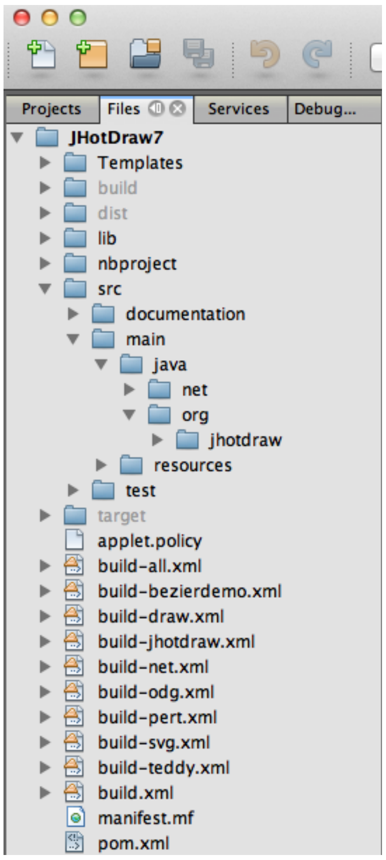
Getting Started with JHotDraw

Creating a NetBeans project which uses JHotDraw 7

Download and install the NetBeans IDE from http://www.netbeans.org
This is recommended even if you don't intend to develop your application with this IDE.

Download a release of JHotDraw 7 or checkout a snapshot from the repository at http://sourceforge.net/projects/jhotdraw/

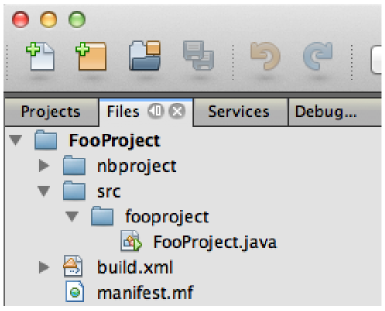
JHotDraw 7 is bundled as a NetBeans project. Open it in NetBeans and make sure that you can compile and run it without errors. Here is how JHotDraw 7 should look like in NetBeans:



```
JHotDraw7
  Templates
  build
  dist
  lib
  nbproject
  src
    documentation
    main
      java
        net
        org
          jhotdraw
      resources
    test
  target
  applet.policy
  build-all.xml
  build-bezierdemo.xml
  build-draw.xml
  build-jhotdraw.xml
  build-net.xml
  build-odg.xml
  build-pert.xml
  build-svg.xml
  build-teddy.xml
  build.xml
  manifest.mf
  pom.xml
```

You can now create a new project using your IDE of choice and include JHotDraw. If you use NetBeans, here are the recommended steps:

Start NetBeans and create a new Java Application project. Here is what you may get if your project has the name FooProject.



We now want to integrate the sources and libraries of JHotDraw 7 into this project, without intermingling the code with our new project.

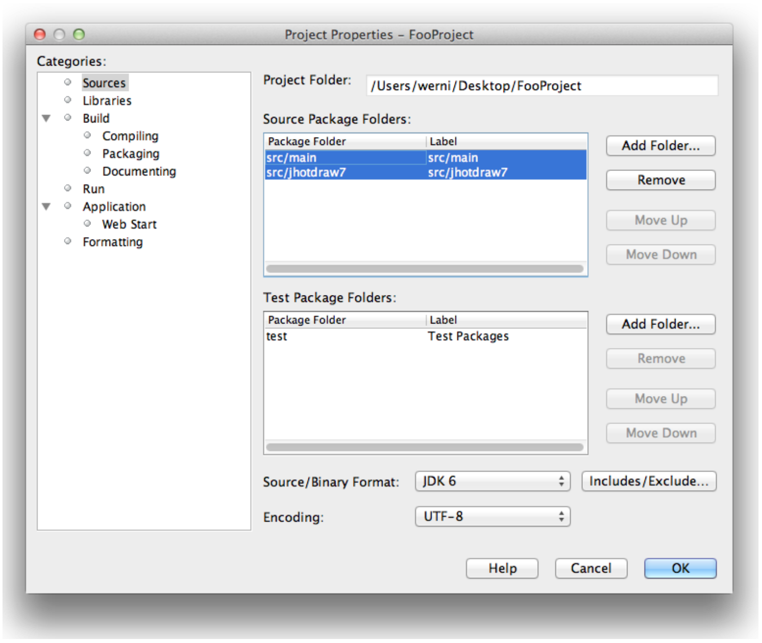
Make sure that the JHotDraw7 project is opened in the IDE.

Perform all the following copy operations in the IDE and not with Windows Explorer or Finder – unless you know what you are doing. This is because the IDE knows how to properly deal with hidden repository files. And that is the reason why copying files in the IDE is so much slower.

Create two new sub-folders in the src folder of your project: FooProject/src/jhotdraw7 and FooProject/src/main.

Copy the JHotDraw7/lib folder into your project, i.e. into FooProject/lib.

Open the Project Properties panel. In the category Sources, remove the src folder and add src/main and src/jhotdraw7 instead. This is how it should look like:


Project Properties – FooProject

Project Folder: /Users/werni/Desktop/FooProject

Source Package Folders:

Package Folder	Label
src/main	src/main
src/jhotdraw7	src/jhotdraw7

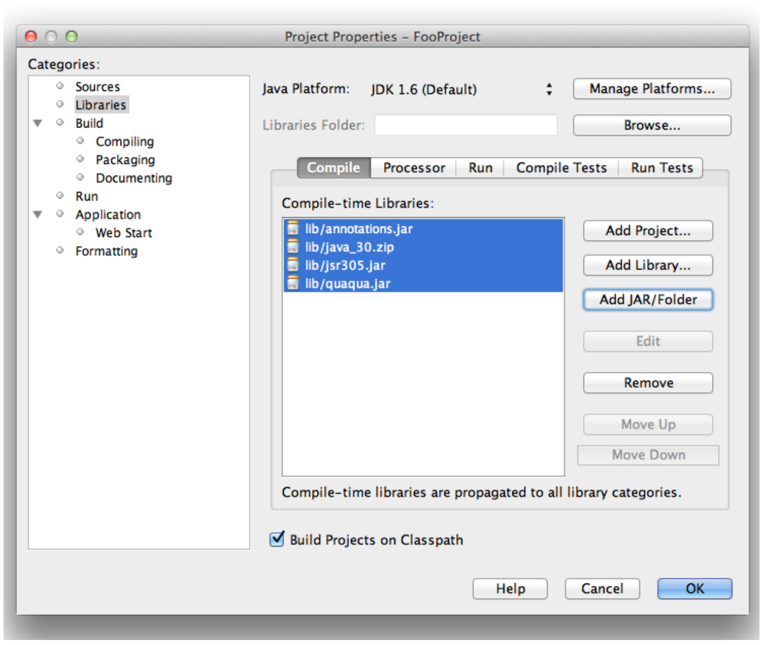
Test Package Folders:

Package Folder	Label
test	Test Packages

Source/Binary Format: JDK 6

Encoding: UTF-8

In the Libraries category, add all .jar files from the lib folder. This is how it should look like:


Project Properties – FooProject

Categories:
- Sources
- Libraries
- Build
  - Compiling
  - Packaging
  - Documenting
- Run
- Application
  - Web Start
- Formatting

Java Platform: JDK 1.6 (Default) — Manage Platforms...

Libraries Folder: — Browse...

Compile | Processor | Run | Compile Tests | Run Tests

Compile-time Libraries:
- lib/annotations.jar
- lib/java_30.zip
- lib/jsr305.jar
- lib/quaqua.jar

Add Project... | Add Library... | Add JAR/Folder | Edit | Remove | Move Up | Move Down

Compile-time libraries are propagated to all library categories.

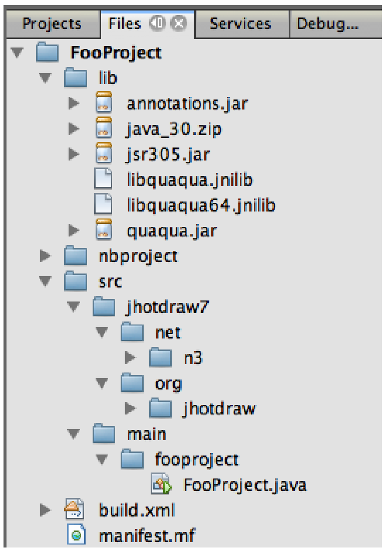
☑ Build Projects on Classpath

Help | Cancel | OK

Now, move all your project sources into FooProject/src/main **without refactoring.**

Copy the sources from the JHotDraw7/src/main/java folder into FooProject/src/jhotdraw7.

This is what it should look like:


Projects | Files | Services | Debug...

```
FooProject
  lib
    annotations.jar
    java_30.zip
    jsr305.jar
    libquaqua.jnilib
    libquaqua64.jnilib
    quaqua.jar
  nbproject
  src
    jhotdraw7
      net
        n3
      org
        jhotdraw
    main
      fooproject
        FooProject.java
  build.xml
  manifest.mf
```

Check if your project compiles.
You are now ready to write an application based on the framework.

Trying out the sample applications

JHotDraw 7 comes with a number of sample applications. They are located in the package org.jhotdraw.samples.

If you are new to the framework, you may want to start with the "mini"-samples, which are located in the org.jhotdraw.samples.mini package. Each of these examples consists of a single Java class only.

The "draw" sample in the org.jhotdraw.samples.draw package shows off all default implementations of the Figure interface. This sample is ideal if you want to try out a default Figure before you integrate it into your own drawing editor.

The "pert" sample in the org.jhotdraw.samples.pert package shows how to add simple behavior to figures.

If you only want to use the application framework, then you may want to try out the "teddy" sample in the org.jhotdraw.samples.teddy package. This example features a simple text editor.

The "svg" sample is the most complex example. It shows how a full-fledged SVG editor could be implemented with JHotDraw 7. The SVG editor can be used as a component in an application of your own. See the SVGDrawingPanelSample in the "mini"-samples package.

References

[1] Mac OS X Human Interface Guidelines.
Apple Inc.
http://developer.apple.com/library/mac/#documentation/UserExperience/Conceptual/AppleHIGuidelines/Intro/Intro.html

[2] Windows User Experience Interaction Guidelines
Microsoft Inc.
http://msdn.microsoft.com/en-us/library/aa511258.aspx