SimplePaint

Query by Example Editor for the VizIR Project

by
Dalibor Mitrovic
mitrovic@ims.tuwien.ac.at

Motivation

Global integration of information systems with the ability for easy creation and digitization of visual content (images and video) have lead to the problem of how these vast amounts of data in collections or databases can be managed. One of the crucial success factors of all approaches addressing this problem is apparently the implementation of effective but easy to handle retrieval methods. Text retrieval on indexing terms has turned out insufficient for two reasons:

- The indexing process is time consuming and expensive because it has to be done by humans
- Visual content is – naturally – often difficult or impossible to describe.

Content-based retrieval of images and video (CBVR) is a rather new approach to overcome these problems by deriving features (or: descriptors; like color histograms, etc.) from the visual content and comparing visual objects by measuring the distance of features with distance functions. CBVR is a helpful addition to text retrieval systems. Its major advantages are fully automated indexing and description of visual content by visual features.
The VizIR project (Visual Information Retrieval) integrates the various directions of past and current research in an open-source, portable, extendible and well-documented class framework.
SimplePaint offers the necessary classes to enable query by example. In fact SimplePaint is a small painting application that can be used to produce multi-layered sketches used as examples to query the database. However it does not provide any querying functionality at all. Its sole purpose is the on-the-fly creation of paintings.

Architecture

Figure 1 shows the Class Diagram of SimplePaint. All classes are implemented in the package "`org.vizir.sketching`".

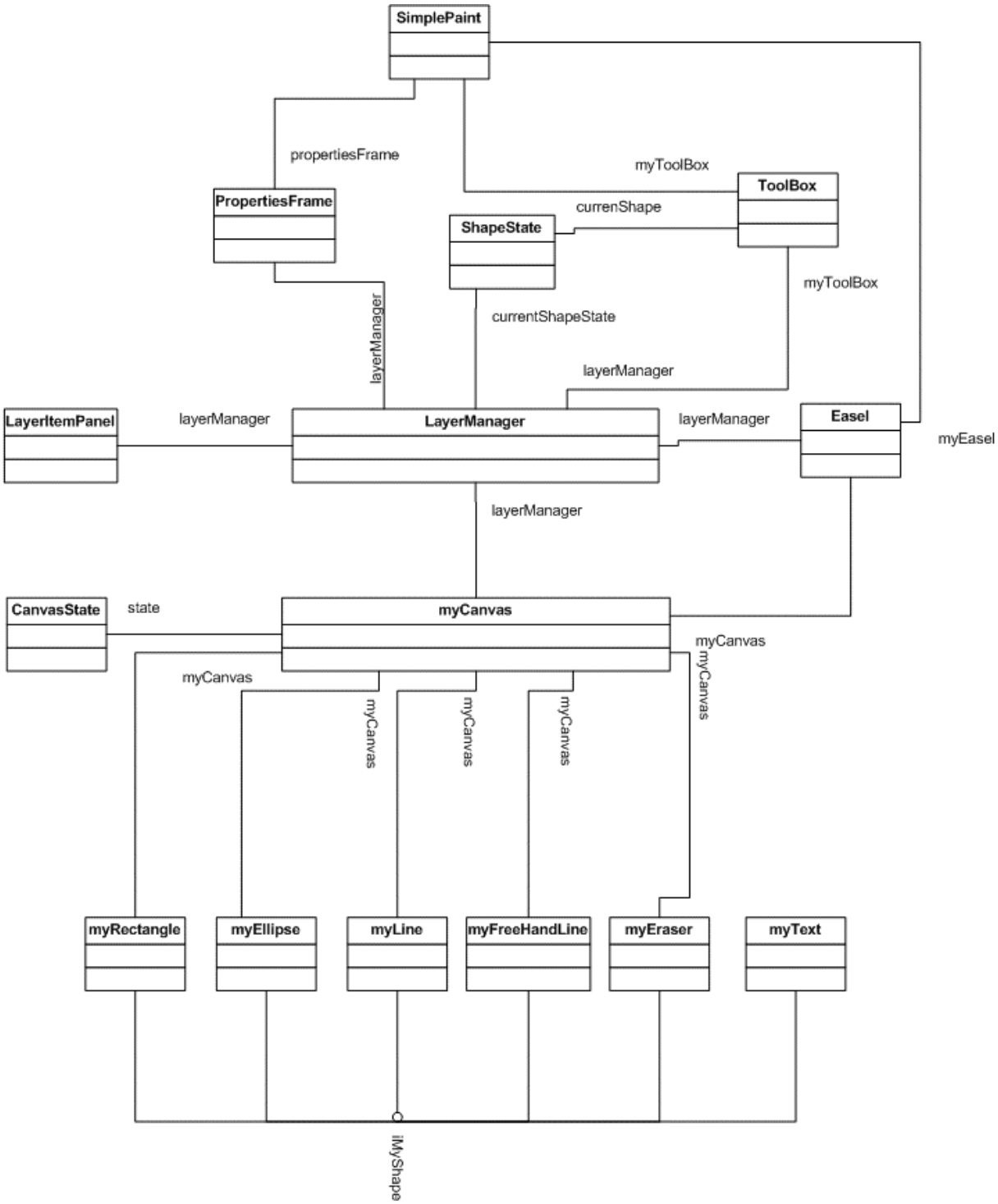

Figure 1 Class Diagram of SimplePaint

The SimplePaint:

The class SimplePaint is the main class, upon creation it creates the other necessary instances of the classes listed below. Since SimplePaint extends `JPanel` it can be simply added to any other Java container.

The ToolBox:
The class ToolBox provides the user-interface for the selection of tools. It holds eight tools and controls three painting parameters. The information which tool is used at the moment and what parameters are applied is stored in a ShapeState object. After a change has been made, the ShapeState object is updated with the new parameters and is passed to the LayerManager.

The LayerManager:
The class LayerManager is responsible for displaying the selected layer with its current alpha value. Furthermore it provides access to a ShapeState object holding the currently selected tool and parameters. During initialization the LayerManager creates all myCanvas objects. The LayerManager has two public methods that both return `BufferedImage` objects. The method `getActiveCanvasImage()` returns the contents of the active canvas, the method `getAllCanvasImage()` returns the contents of all canvases combined into one single canvas respecting each canvas' alpha value.

The myCanvas:
Each layer of the painting is stored as one myCanvas object. myCanvas stores all iMyShape objects painted on the layer in one Vector. myCanvas provides methods to add and remove iMyShape objects. The most noteworthy function in the myCanvas class is the creation of all iMyShape objects. When a Mouse Button is pressed a new iMyShape object is created, the parameters like the actual type of the shape (rectangle, oval, etc.), color, whether the object should be filled or not and the like. After creation it is added to the Vector storing all iMyShape objects of this layer.

The iMyShape:
The class iMyShape specifies two methods, namely `paint()` and `isSelected()`. Since this application tries to be as object oriented as possible every shape that can be drawn onto the canvas has to bring its own painting code. The method `isSelected()` is used to switch between the two drawing modes most shapes have. If a shape is selected it is drawn in a different highlighted way with a frame around it. The only exception is the myEraser object, it has no special highlighted mode.

Features

In this section we will have a look at the features SimplePaint offers. A selection of important features is listed below:

- **multiple layers**
  SimplePaint supports drawing to multiple independent layers. Each layer has an alpha channel and a background color.

- **alpha-compositing of layers**
  The user can change the opacity from zero (invisible) to 100 (fully opaque). The "all canvas" contains the image of all other layers combined taking into account their alpha channel value.

- **painting object oriented**
  SimplePaint's drawing primitives are stored as separate objects; this permits manipulation on an object level. In future releases it can be made possible to select every shape and for example scale it.

- **highlighting when shape is selected**
  When a drawing primitive is selected it is rendered in a highlighted way. A frame is rendered around the shape and if the shape is not filled the line-width is changed.

- **access to the `BufferedImage` and myCanvas objects**
  SimplePaint offers the developer access not only to the image data, but also to the myCanvas object. This way the developer has more power.

Limitations & known Issues

This section lists and explains the limitations of SimplePaint and some known bugs:

- **no possibility of filling a region with a specified color**
  It is not possible to fill a contiguous region with a certain color. This is due to the object oriented character of the painting. A region that should be filled would have to be separate iMyShape object.

- **no point and click selection**
  It is not possible the select a shape by clicking on it. Selection has to be performed via the selection combobox.

- **no shape manipulation possible**
  After a shape is created it is not possible to alter its appearance.

- **no save/load of sketches possible**
  It is not possible to save a painting. Furthermore one cannot load an image and manipulate it.

- **high coupling**
  Although SimplePaint was designed with object oriented principles in mind, the release version is highly coupled.

- **strange behavior of color chooser window**
  The color chooser window looses focus the moment the mouse leaves it. Furthermore if the application's main window is maximized the color chooser window is hidden the moment it is entered by the mouse cursor. One has to reduce the application's main window in order to access the color chooser window.

- **shape deletion of unselected shapes possible**
  It is possible to delete a shape that is not marked as selected. The shape deletion routine does not check whether a shape is selected or not prior to deleting it.

- **selection visible in `BufferedImage`**
  If a shape is selected and the LayerManager's methods `getActiveCanvasImage()` or `getAllCanvasImage()` are called the returned `BufferedImage` shows the selected shape rendered in its highlighted mode.

Future Work

For the future removing of the issues has highest priority. The two most important ones are the strange behavior of the color chooser window and that the selection is visible in the `BufferedImage`. Deferred manipulation of shapes could be achieved by simple means i.e. making shape's attributes accessible. Together with point and click selection this would significantly improve SimplePaint. Another way to improve PraktiPaint would be to enhance the user interface. Loading and saving of sketches could also be implemented, this way one could separate the process of constructing examples for the visual information retrieval system's query by example functionality from the actual execution.