JTourBus Experiment Documentation

This document complements the workspace images, which can be downloaded from the JTourBus homepage[1], and contains the full task descriptions including solutions given to the participations of the experiment. The JTourBus experiment took an unmodified version of JHotDraw and added only documentation information as required by the experiment setup.

Task Description

You want to participate in the Open Source project JHotDraw and thus have decided to take on some maintenance tasks from the bug tracker on Sourceforge related to the topic of Handlers.

As a first step you want to understand the software well enough to issue change requests which can be implemented afterwards.

A Change Request for this matter should look like this:

```
Class.Method(int,int): Change this method, so that...
```

For this the method signature can be copied from the source code.

The description of the change does not need to be a full specification of what needs to be done or which special cases to think about, but rather should describe in a short sentence which effect the change should have on the behavior of the program. For this task, no implementation or test cases are necessary, but rather a good understanding of the problem.

Before giving the tasks to you, we will first tell you a little bit about JHotDraw. Afterwards we will give you an example of such a change request.

For Group C(ontrol):

Unfortunately there is no special documentation apart the JavaDocs for the part of the software that your tasks will be in.

For Group J(TourBus) and Gruppe P(lain text):

Fortunately enough the software contains some documentation in the area where your tasks need to be done and you do not need to start from scratch to explore the software.

For Group J:

Documentation is given as tour-routes through the source code. Since you are probably not accustomed to this type of documentation, we will explain the idea a little bit:

> A Tour is a series of places in the source code that are related to a certain concern or part of the software. These places are marked by the special JavaDoc tag `@JTourBusStop`. Each stop on the route has a number, a tour name and a description. A colon separates the stop from further comments. In the code this might look like this:

[1] https://www.mi.fu-berlin.de/w/SE.JTourBus

```
/**
 * @JTourBusStop 1, Application-Startup-Tour, Main-Method:
 *
 * This application is started from the main method...
 */
public static void main(String[] args){
```

The comment beneath each stop will usually explain the stop and should not be understood as a regular JavaDoc-API comment. The stops are arranged so that if you visit them in order of their numbers, the tour should make the most sense, but each stop should also make sense individually.

To allow for easy navigation between stops you can use the JTourBus Routes View in Eclipse (Window → Show View → Other → Java Browsing → JTourBusRoutes). This view groups and sorts the stops and allows you to visit each stop in turn by navigation buttons or by double clicking. Initially you need to import the tours by using the Refresh-button (mark a project and press the refresh button in the JTourBusRoutes view).

For Group P:

The documentation is given as a txt-file in the project directory named "Handler-Doku.txt"

For Group J and P:

Please do not expect a direct answer to your tasks (the documentors did not know which tasks would be necessary to execute in the future), but still a lot of useful knowledge.

JHotDraw Introduction

A colleague of yours has been nice enough to give you a short overview about the software. His email and a couple of UML diagrams can be found in the project folder as 'Architecture Overview.txt' and 'Architecture Overview.ppt':

Architecture Overview

JHotDraw is a framework for developing graphics applications. As a framework JHotDraw is built around a set of interfaces which can be implemented in various ways. These central interfaces can be found in the package org.jhotdraw.framework.

Of all possible applications that can be developed, using JHotDraw we will only look at the following:

The JavaDrawApp

This application uses the framework to create a simple drawing editor and can be found in the package org.jhotdraw.samples.javadraw and contains a corresponding class as a main entry point.

The JavaDrawApp is a DrawApplication from the framework and responsible for the construction of the user interface (for instance the menu). Once the app has been started in the open-Method, it uses Swing-Events and not a dedicated event-loop to operate. A DrawApplication can display Drawings in DrawingViews (upper case words are interfaces/classes from JHotDraw). A Drawing can be edited using Tools. If for instance a Tool is selected using a ToolButton, then the mouse events captured by the DrawingView are sent to this tool and from this to a Handle.

A Drawing consists of Figures which are responsible for actual drawing and manage Handles and Connectors.

Two typical Tools are for instance the SelectionTool for selecting figures and the CreationTool for creating new Figures.

Now the promised **example**:

Let's assume the following task of cleaning up the menu bar in JHotDraw to only show the shapes Rectangle, Polygon, Triangle and Scribble, then the text above already provided a good hint to look at the JavaDrawApp. In this class, there is a method createTools which initializes the tools to create figures. Thus, the following would be a great answer:

```
JavaDrawApp.createTools(): Only create and add those tools to the
palette that are wanted
```

This is all that is needed for a perfectly fine answer, even though if you had looked around a little bit more, you would have found:

```
MDI_DrawApplication.createTools(): Remove Drag and Drop Tool
```

To sum up: Your first idea on how to solve the task is what we want.

Frequently asked questions

How do I start JHotDraw?	org.jhotdraw.samples.javadraw.JavaDrawApp ➔ Right click ➔ Run As... ➔ Java Application
How can I find the declaration of a method / class / attribute?	CTRL-click on the name of the method/class/attribute.
How can I determine which class is implementing an interface or which class is a subclass of another?	Right-click the class name and then "Open Type Hierachy".
How do I find all callers of a method?	Right-click the method and "Open Call Hierarchy".
How do I determine all places where a method is implemented or overridden?	Right-click the method ➔ „Open Type Hierarchy“ ➔ „Lock View and Show Members“
How can I find all places in which a method/class/attribute occurs?	Right-click the method/class/attribute ➔ "Find all Occurrences".

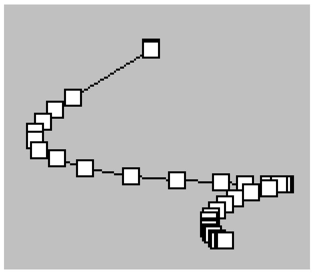
Group J only:

How can I see the JTourBus routes?	Window -> Show View -> Other -> Java Browsing -> JTourBusRoutes
How do I explore a tour?	Select the project and press the Refresh-button in the JTourBusRoutes view.

Tasks

Task 1

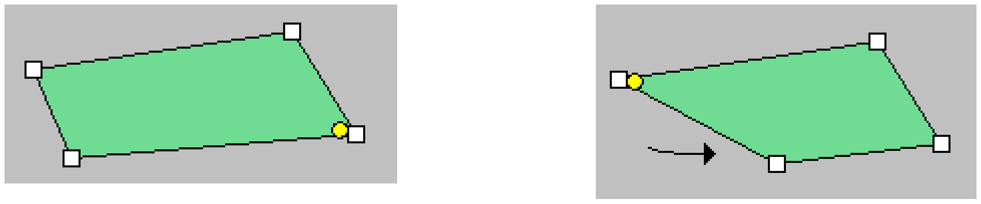
Using the ScribbleTool you can draw a connected series of line segments. Unfortunately such a series of line segments cannot be enlarged in the same way as for instance a triangle using the typical 8 resize-handles. Can you identify the location/s in the code which need to be adapted to allow resizing?



Task 2

A polygon has a little yellow handle for rotating the polygon. Unfortunately there is a bug in the current implementation which makes the little yellow handle jump to another corner, when changing the polygon using the corner handles.

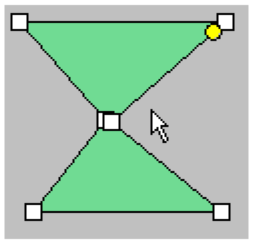
The following screenshot shows what is happening if the lower left corner is moved to the left:



Which location/s in the code need to be changed to avoid this behavior?

Task 3

If several handles of the same figure are drawn on the same spot, the mouse always selects the one drawn at bottom. It would be preferred to always move the handle that is also drawn on top of the others. Shown below is an example situation where the handles overlap:



Which location/s in the code need to be adapted, so that the handle on the top can be moved?

Task 4

New figures can be inserted into an area marked by the mouse. From other graphic editors it is a well-known feature to insert new figures centered around the first click, if CTRL is being held. Which location in the code needs to be adapted, so that such a change is possible for ellipses?

Task 5

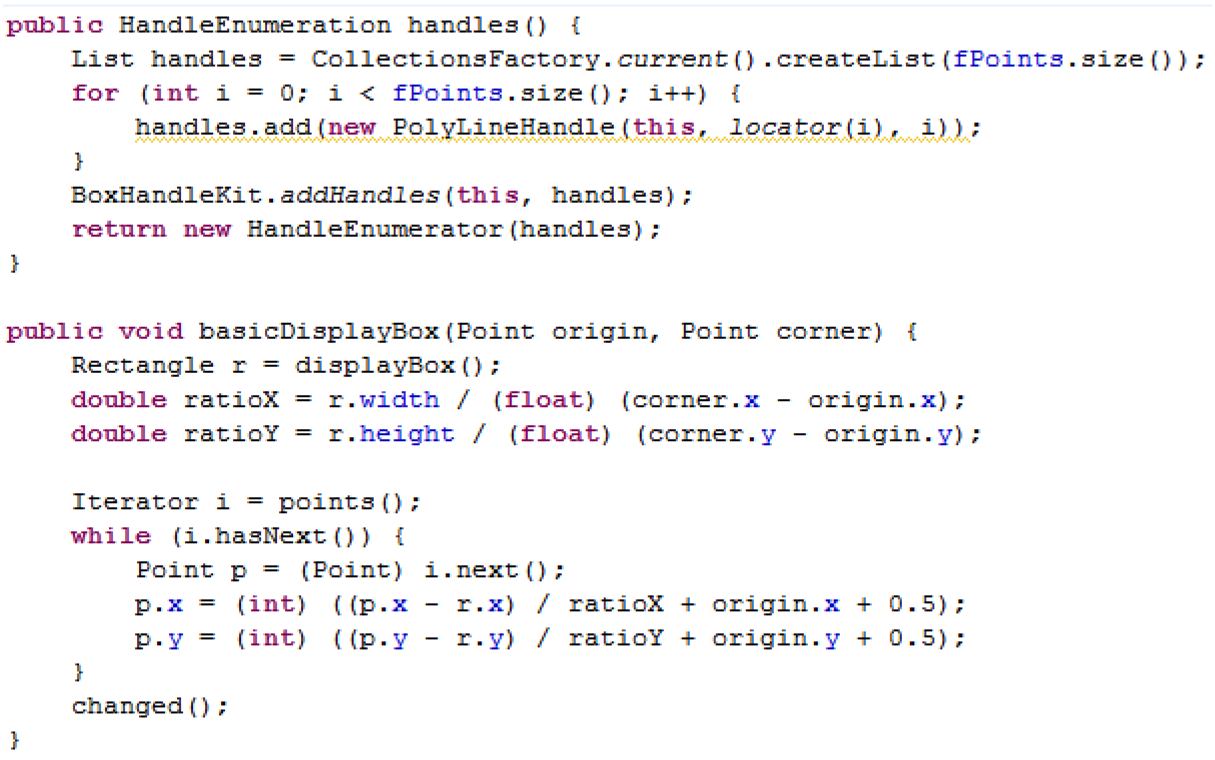
A last problem of JHotDraw is that you cannot abort a drag-action (for instance using escape). Which location in the code needs to be changed to enable this feature?

Solutions

Task 1 (Make scribbles resizable)

PolyLineFigure.handles(): Add BoxHandleKit to the Handles-Enumeration.

PolyLineFigure.basicDisplayBox(): Modify in such a way that the figures are always scaled to the given rectangle.



```
public HandleEnumeration handles() {
    List handles = CollectionsFactory.current().createList(fPoints.size());
    for (int i = 0; i < fPoints.size(); i++) {
        handles.add(new PolyLineHandle(this, locator(i), i));
    }
    BoxHandleKit.addHandles(this, handles);
    return new HandleEnumerator(handles);
}

public void basicDisplayBox(Point origin, Point corner) {
    Rectangle r = displayBox();
    double ratioX = r.width / (float) (corner.x - origin.x);
    double ratioY = r.height / (float) (corner.y - origin.y);

    Iterator i = points();
    while (i.hasNext()) {
        Point p = (Point) i.next();
        p.x = (int) ((p.x - r.x) / ratioX + origin.x + 0.5);
        p.y = (int) ((p.y - r.y) / ratioY + origin.y + 0.5);
    }
    changed();
}
```

Note: This certainly is no perfect solution, as repeated scaling operations will distort the figure.

Task 2 (Jumping yellow handle)

PolygonScaleHandle.getOrigin(): Modify in such a way that the calculation is stable with regards to changes to the polygon.

Task 3 (Overlapping handles)

StandardDrawingView.findHandle(): Change the ordering of selectionHandles() before searching.

Task 4 (CTRL mouse)

CreationTool.mouseDrag(...) and CreationTool.mouseDown(...): Adapt calls to getDisplayBox in such a way that the DisplayBox is changed into the other direction, if CTRL is being held:

```java
if ((e.getModifiersEx() & InputEvent.CTRL_DOWN_MASK) == 0) {
    getAddedFigure().displayBox(new Point(getAnchorX(), getAnchorY()),
            new Point(x, y));
} else {
    getAddedFigure().displayBox(
            new Point(2 * getAnchorX() - x, 2 * getAnchorY() - y),
            new Point(x, y));
}
```

Task 5 Cancelling

SelectionTool.keyDown() (overrides AbstractTool.keyDown()): Catch ESCAPE and undo action

```java
/**
 * Handles key down events in the drawing view.
 */
public void keyDown(KeyEvent evt, int key) {
    if (getDelegateTool() != null) {
        if (key == KeyEvent.VK_ESCAPE) {
            getDelegateTool().mouseUp(null, 0, 0);
            getDelegateTool().deactivate();

            Undoable lastUndoable = view().editor().getUndoManager()
                    .popUndo();
            lastUndoable.undo();
            lastUndoable.getDrawingView().checkDamage();

            editor().figureSelectionChanged(lastUndoable.getDrawingView());

            setDelegateTool(null);
            if (view() != null) {
                view().unfreezeView();
                editor().figureSelectionChanged(view());
            }
        } else {
            getDelegateTool().keyDown(evt, key);
        }
    }
}
```

Unfortunately the undo concern is not implemented very well in JHotDraw. The proposed change thus only works for handles, because these are wrapped as UndoableHandles.