Spare Time Development - A little guide line for software developing hobbyists

Stephan Strauss

January 8, 2020

Part I

Introduction

Chapter 1

Why this book

1.1 How it started

The work on this little book in hand started in the summer of 2012 as side product of a little private Java learning and development project. Although absorbed by the daily work in our company, my good friend Claudio Vaccani and my humble self started to work on it in our spare time.

1.2 Requirements to fulfill

It is a hard job to get a serious development project done as a hobbyist. Especially at least if you plan to read Thomas Mann, Marcel Proust and Adorno at the same time – just kidding. Therefore we decided not to reach for the stars but to develop a useful little utility, that fulfils several requirements:

- should be liked on facebook by Richard Wagner and Nietzsche.
- should be created by god and such heavy he could not pull himself.

Sorry, this was the wrong list, once again, that requirements should be fulfilled:

- should be based on pure java code (no native JNI code).
- should have minimal dependencies to other projects.
- should be designed first on paper.
- should have a foreseeable finalization.
- time constraints should be kept minimal - the learning effect maximal.

To be honest, we never believed that we could achieve a final product but we wanted to learn and as much as possible and therefore defined the requirements as such.

1.3 Learning by doing

I learned already some years before with another friend and former colleague Ralf Gautschi the basics of the java language using several books that I will list at the end of this chapter. Therefore this project as described in this book was more concerned about good design and patterns, about unit testing and re-factoring, about agile programming and other streams and temporary fashions - not so much about the java language itself.

With not much knowledge but with a unbreakable will we started and ended successfully the project: The resulting little story is worth telling and therefore written down in this little cheap Amazon book for Amazons.

Because the corresponding source code is probably not the best seen in software industry, the self-critical aspect is the center of this book. I think we learned a lot , especially by criticizing our own result.

Ralf Gautschi is now principle engineer at Microsoft, Zuerich. Claudio and me are still working for Synopsys AG. We are both located in Zurich and work in R&D.

The tool developed (it's working name was TEME) is used to measure precisely distances and aeras

Chapter 2

Drawables

2.1 The `Drawable` Interface

The basic properties of all objects that should be drawn in the `TEMView` are summarized essentially in two interfaces. The first interface is called `Drawable` (see listing 2.1) and implements a `paint` and a `contains` method. The paint method is used after implementation to place the `Drawable` into the `TEMView` after application of an `AffineTransform`

Listing 2.1: The Drawable Interface all `Drawables` are implemented against

```
1 public interface Drawable  {
2   public void paint(Graphics2D graphicsContext, AffineTransform transformation);
3   public boolean contains(int x, int y);
4 }
```

Listing 2.2: The transformation of the viewPort (`viewPortTransform: AffineTransform`) is used by the `paint` method to place the `Drawable` objects into TEMView's graphicsContext

```
1   @Override
2   public void paintComponent(Graphics g) {
3     super.paintComponent(g);
4     Graphics2D g2D = (Graphics2D) g;
5     ...
6     for (Drawable drawable : drawableList) {
7       drawable.paint(g2D, this.viewPortTransform);
8     }
9   }
```

2.2 The Reference Scale Drawable

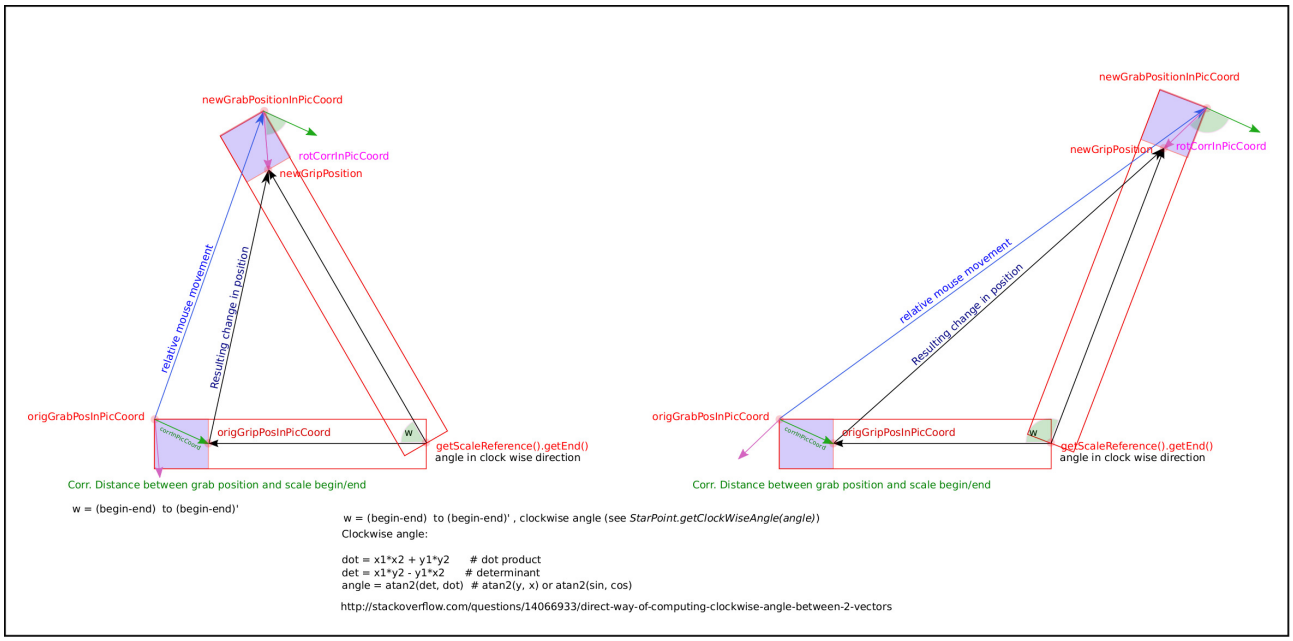

Figure 2.1: Calculation of a new grip position as function of a relative repositioning of the mouse pointer